CORA Cork Open Research Archive
Cartlann Taighde Oscailte Chorcaí

Title	Exploring Field Programmable Gate Array Architectures for Pascal's Simplex-based multinomial option pricing
Authors	O'Mahony, Aidan T.
Publication date	2025
Original Citation	O'Mahony, A. T. 2025. Exploring Field Programmable Gate Array Architectures for Pascal's Simplex-based multinomial option pricing. PhD Thesis, University College Cork.
Type of publication	Doctoral thesis
Rights	© 2025, Aidan Timothy O'Mahony. - https://creativecommons.org/licenses/by/4.0/
Download date	2026-11-15 16:16:24
Item downloaded from	https://hdl.handle.net/10468/17422



UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Ollscoil na hÉireann, Corcaigh

National University of Ireland, Cork



Exploring Field Programmable Gate Array Architectures for Pascal's Simplex-Based Multinomial Option Pricing

Thesis presented by

Aidan Timothy O'Mahony, BSc MSc

0000-0001-8413-9656

for the degree of

Doctor of Philosophy

University College Cork

Department of Electrical Engineering

Head of School/Department: Prof. Jorge Oliveira/Prof. Peter Parbrook

Supervisor(s): Dr. Emanuel Popovici, Prof. Bernard Hanzon

2025

Contents

List of Figures **1**

List of Tables **1**

Abstract iv

Associated Publications vi

Acronyms vii

Acknowledgements vii

1 Introduction **1**

1.1 Contributions of this Thesis 4

1.2 Thesis Outline 6

2 Background **7**

2.1 Modern Option Types 8

2.1.1 Vanilla Options 8

2.1.2 Exotic Options 10

2.2 Methods of Valuing Options 15

2.2.1 Analytical Methods 15

2.2.2 Lattice Models 26

2.2.3 Risk-neutral Valuation 33

2.3 Introduction to Hardware Acceleration 34

2.3.1 Different Types of Hardware Accelerators 35

2.3.2 Challenges Associated with Hardware Acceleration . . . 37

2.4 Field Programmable Gate Arrays (FPGAs) 39

2.4.1 Introduction to FPGAs 39

2.5 Parallel Computing Platforms 42

2.5.1 Introduction to the importance of parallel computing in financial modeling 42

2.5.2 OpenCL: Origins, Architecture, and Its Significance in the Context of this Research 43

2.5.3 OneAPI: Introduction, its advantages, and relevance for option pricing applications . . . 45
2.6 Conclusion . . . 46

3 State-of-the-art in FPGA Acceleration of Option Pricing 48
3.1 Introduction . . . 48
3.2 Methodology . . . 48
3.2.1 Data Extraction . . . 50
3.3 Comparison of FPGA, CPU, and GPU . . . 50
3.4 Review of FPGA Implementations for Option Pricing . . . 51
3.4.1 Black–Scholes Model . . . 51
3.4.2 Binomial and Trinomial Tree Methods . . . 57
3.4.3 Monte Carlo Simulations . . . 62
3.4.4 Finite Difference . . . 69
3.4.5 Finite Difference on FPGA . . . 69
3.4.6 Heston Model . . . 72
3.4.7 Quadrature Methods . . . 74
3.4.8 Miscellaneous Models . . . 77
3.5 Challenges and Future Directions . . . 83
3.5.1 Challenges . . . 83
3.5.2 Future Directions . . . 83
3.6 Conclusions . . . 84

4 Implementation of a Multinomial Tree Method for Option Pricing 86
4.1 Introduction . . . 86
4.2 Theoretical Framework . . . 87
4.2.1 The Sierag-Hanzon Model . . . 88
4.2.2 Key Concepts and Definitions . . . 89
4.2.3 Models and Equations . . . 91
4.2.4 Multinomial Tree Methodology . . . 102
4.3 Implementation Strategy . . . 105
4.3.1 Binomial Option Pricing Model for European Options . . . 105

4.3.2 Extension to Multinomial Option Pricing 106
4.3.3 Simplified multiasset option pricing 107
4.3.4 Recursive Factor Adjustment Method 110
4.3.5 Monomial Ordering and FIFOs 111
4.3.6 Optimization of Input Parameters 113
4.3.7 Computation of d and Q 114
4.3.8 Data Structures and Storage 119
4.4 Code Implementation 125
4.4.1 Code Structure and Design 126
4.5 Conclusion 128

5 Two-asset European option pricing 130
5.1 Introduction 130
5.2 Related Work 131
5.3 Financial Options 132
5.3.1 Binomial Option Price Model 133
5.3.2 Recombining Multinomial Trees Based on Pascal's Simplex 134
5.3.3 Comparison against using Binomial Trees for Two Asset Computation 138
5.4 Architecture 138
5.4.1 Computation of the Direction Vectors and Risk-Neutral Probability Vector 139
5.4.2 Pipelined Computation of Final Option Value 140
5.4.3 OpenCL Kernels for Two Asset European Option Pricing 141
5.5 Results 143
5.6 Conclusion 145

6 Multi-asset option pricing of European Options 146
6.1 Introduction 147
6.2 Related Work 148
6.3 Financial Options 149

6.3.1 Recombining Multinomial Trees Based on Pascal's Simplex . . . 151
6.4 Proposed FPGA architecture for financial options . . . 153
6.4.1 High-level system design exploration . . . 154
6.4.2 Multi-bank ping-pong access . . . 156
6.4.3 System implementation details . . . 159
6.5 Results and performance analysis . . . 161
6.5.1 Hardware and software specifications . . . 161
6.5.2 Metrics and key performance indicators . . . 162
6.6 Conclusions . . . 170

7 Multi asset option pricing of American Options 171
7.1 Introduction . . . 171
7.2 Related Work . . . 172
7.2.1 Differences between models . . . 172
7.2.2 American Binomial Option Price Model . . . 173
7.2.3 Recombining Multinomial Trees Based on Pascal's Simplex . . . 173
7.3 Proposed FPGA architecture for financial options . . . 174
7.3.1 Intel OneAPI and DPC++ . . . 174
7.3.2 Design of the Multinomial Option Pricer . . . 174
7.4 Results . . . 179
7.4.1 Accuracy Analysis . . . 179
7.4.2 Latency Analysis . . . 179
7.4.3 Resource Utilization Analysis . . . 180
7.4.4 Comparative Analysis . . . 181
7.5 Conclusion . . . 182

8 Conclusion and Future Work 184

References 187

Declaration of Work

I, Aidan O Mahony, hereby declare that this Ph.D. thesis entitled

“Exploring Field Programmable Gate Array Architectures for Pascal’s Simplex-Based Multinomial Option Pricing”

has been written entirely by me under the supervision of Dr. Emanuel Popovici and Prof. Bernard Hanzon.

This thesis has not previously been submitted for any other award at this university or at any other institution.

_____, 27 March 2025

List of Figures

1.1 Chicago Board Options Exchange 2

2.1 Generic FPGA Architecture 40

3.1 Bar chart of yearly counts of publications. 49

4.1 A portion of Pascal's Triangle. 90
4.2 A binomial lattice with entries from Pascal's Triangle. 91
4.3 Visualization of FIFO Queues for Different Variables 109

5.1 Example of 3-dimensional to 2-dimensional projection 135
5.2 The block diagram of the proposed system architecture 137

6.1 Visual example of a three-dimensional projection to two dimensions 153
6.2 High-level architecture of the multi-asset option pricer. 155
6.3 Multi-bank ping-pong access block diagram. 157
6.4 The compute block. 162
6.5 Schematic Diagram of the Intel FPGA SDK for OpenCL Programming Model 163
6.6 Stratix 10 Programmable Acceleration Card 163
6.7 Precision of the two-asset option pricer. 165
6.8 Precision of the four-asset option pricer. 168

7.1 High-Level Architecture of the American Option Pricer 176
7.2 Comparison of pricing methods for an option on three assets. . 180
7.3 Latency Comparison for American Option Pricing with Two to Seven Underlying Assets 180

List of Tables

3.1 Comparison between FPGA, CPU, and GPU across various aspects. 51
3.2 Summary of notable findings from various studies on FPGA implementations of the Black-Scholes model. 55
3.3 Summary of notable findings from various studies on FPGA implementations of binomial and trinomial tree methods. . . . 61
3.4 Summary of notable findings from various studies on FPGA implementations of Monte Carlo simulations. 67
3.5 Summary of notable findings from various studies on FPGA implementations of finite difference methods. 71
3.6 Summary of notable findings from various studies on FPGA implementations of the Heston Model. 74
3.7 Summary of notable findings from various studies on FPGA implementations of Quadrature Methods. 77
3.8 Summary of notable findings from various studies on FPGA implementations of miscellaneous models. 81

4.1 Precomputed OpenCL kernel inputs 119

5.1 OpenCL kernels for two asset option pricing 140
5.2 Input to two-asset option pricer 143
5.3 Latency performance and pipelining impact 144
5.4 Performance-area results 144

6.1 OpenCL kernel descriptions for multinomial option pricing . . . 156
6.2 OpenCL kernel inputs for multinomial option pricing 158
6.3 OpenCL kernel utilization for multinomial option pricing 159
6.4 Option pricer parameter values 164
6.5 Latency performance and pipelining impact as measured in seconds per option price 166

6.6 Stratix 10 resource utilization and maximum frequency (fMAX) 167

7.1 OneAPI FPGA Inputs for American Option Pricing 177
7.2 Detailed Overview of Kernels Employed in the Option Pricing Algorithm 178
7.3 Resource Utilization (as a Percentage of Total) for Option Pricing Across Different Numbers of Underlying Assets on the Intel FPGA PAC Stratix 10 SX D5005. 181

List of Algorithms

1 Pseudocode for Binomial Model 106
2 Pseudocode for Multinomial Model 106
3 Option Pricing Process with FIFOs 107

Abstract

In financial engineering, the speed and accuracy of option pricing models are critical for decision-making in fast-paced markets. This thesis explores the application of Field Programmable Gate Arrays (FPGAs) to accelerate the computation of option pricing models, specifically focusing on Pascal's Simplex method. Pascal's Simplex is chosen for its scalability in handling multi-asset options, which becomes increasingly vital as the complexity of financial instruments grows.

FPGAs are selected due to their ability to execute parallel computations efficiently, offering significant speedups compared to traditional CPU-based systems while maintaining computational accuracy. This work introduces a novel FPGA architecture for Pascal's Simplex-based multinomial option pricing and presents a detailed comparative analysis of FPGA and CPU implementations. The results show that FPGAs provide substantial performance improvements in terms of speed and scalability, both for European and American style options, especially for pricing models with a high number of underlying assets.

This research demonstrates that FPGAs can offer up to 43 times speedup over traditional methods while preserving accuracy, making them a powerful tool for real-time financial analytics. Furthermore, the architecture supports generalization for a varying number of assets underlying each option, and implementations on state of the art hardware are presented for up to 7 assets. By introducing a novel FPGA architecture for Pascal's Simplex-based multinomial option pricing and demonstrating up to 43 times speedup over traditional CPU methods while preserving accuracy, this thesis significantly contributes to advancing real-time financial analytics for complex multi-asset options.

To my wife Aileen, for her love, patience, and encouragement, and to my children Ash, Alex, and AJ, who bring light and laughter to my days.

Associated Publications

- Mahony, Aidan O., Gil Zeidan, Bernard Hanzon, and Emanuel Popovici. **A Parallel and Pipelined Implementation of a Pascal-Simplex Based Two Asset Option Pricer on FPGA using OpenCL.** In 2020 IEEE Nordic Circuits and Systems Conference (NorCAS), pp. 1-6. IEEE, 2020.
- Mahony, Aidan O., Gil Zeidan, Bernard Hanzon, and Emanuel Popovici. **A Parallel and Pipelined Implementation of a Pascal-Simplex Based Multi-Asset Option Pricer on FPGA using OpenCL.** , pp. 1-6, Microprocessors and Microsystems (2022).
- A. O. Mahony, G. Zeidan, B. Hanzon and E. Popovici, **American Multinomial Option Pricing on FPGA using OneAPI**, 2023 IEEE 66th International Midwest Symposium on Circuits and Systems (MWSCAS), Tempe, AZ, USA, 2023, pp. 831-835.
- O Mahony, Aidan, Bernard Hanzon, and Emanuel Popovici. **The Role of FPGAs in Modern Option Pricing Techniques: A Survey.** Electronics 13.16 (2024), pp. 3186-3211.

Other Publications

- Mahony, Aidan O., and Emanuel Popovici. **Power analysis of sorting algorithms on FPGA using OpenCL.** In 2018 29th Irish Signals and Systems Conference (ISSC), pp. 1-6. IEEE, 2018.
- Mahony, Aidan O., and Emanuel Popovici. **A systematic review of blockchain hardware acceleration architectures.** In 2019 30th Irish Signals and Systems Conference (ISSC), pp. 1-6. IEEE, 2019.

Acronyms

FPGA	Field-Programmable Gate Array
MC-Tree	Monte Carlo-Tree Method
CVA	Credit Valuation Adjustment
GPP	General Purpose Processor
GPU	Graphics Processing Unit
CPU	Central Processing Unit
OpenCL	Open Computing Language
HDL	Hardware Description Language
HFT	High-Frequency Trading
API	Application Programming Interface
ASIC	Application-Specific Integrated Circuit
RTL	Register-Transfer Level
SoC	System on Chip
EIT	Electro Information Technology
FCCM	Field-Programmable Custom Computing Machines
CBOE	Chicago Board Options Exchange
ALU	Arithmetic Logic Unit
CGRAs	Coarse-Grained Reconfigurable Arrays
PIM	Processing-In-Memory
MVGRNG	Multivariate Gaussian Random Number Generator
NRE	Non-Recurring Engineering
PCIe	Peripheral Component Interconnect Express
AXI	Advanced eXtensible Interface
DDR	Double Data Rate
DPC++	Data Parallel C++
ISLs	Iterative Stencil Loops
HPC	High Performance Computing
VaR	Value-at-Risk
GCM	Gaussian Copula Model
OTC	Over-The-Counter
LSMC	Least-Squares Monte Carlo
IP	Intellectual Property

BGM	Brace Gatarek Musiela
FPGA PAC	FPGA Programmable Acceleration Card
QEI	Query Engine Interface
XCI	Xilinx Contextual Interface
CUDA	Compute Unified Device Architecture

Acknowledgements

I extend my deepest gratitude to my supervisors, Dr. Emanuel Popovici and Professor Bernard Hanzon, for their invaluable guidance, patience, and wisdom throughout this journey. Special thanks to Gil Zeidan for his assistance and support, which have been crucial to my research. I am profoundly grateful to Dell Technologies for sponsoring my studies, to Intel Corporation for the essential hardware and software, and to the Insight Centre for Data Analytics for their support of my research endeavors.

Chapter 1

Introduction

The world of finance is a complex and rapidly evolving field. With the advent of new technologies and the increasing complexity of financial instruments, the need for sophisticated computational models has never been greater.

It is useful at this point to define what an options contract is. An options contract is a financial agreement that grants the buyer the right, but not the obligation, to buy or sell a particular asset (like a stock) at a preset price within a given period. This component of financial markets has a rich history, tracing its roots back to ancient times [1], evolving through the ages, and culminating in the sophisticated models of the present day. The study of option pricing represents our effort to understand financial markets, showcasing significant mathematical developments and our growing grasp of risk and uncertainty.

From the rudimentary options contract of Thales of Miletus in ancient Greece to the speculative trading of the Tulip Mania in 17th century Holland, we will examine the origins and evolution of option pricing. We will explore the establishment of the Chicago Board Options Exchange, shown in Figure 1.1, and the advent of the revolutionary Black-Scholes-Merton model, which marked the birth of modern option pricing. Moving into the present, we will examine the various types of options that exist today, from vanilla options to exotic variants, and their role in financial markets. We will also inspect the current techniques for valuing options, exploring the mathematical models and computational methods that traders and financial institutions use to price these complex financial instruments. This introduction serves as a foundation into to the world of option pricing, providing the basic knowledge needed to navigate this complex field. The exploration of the history of option pricing ex-

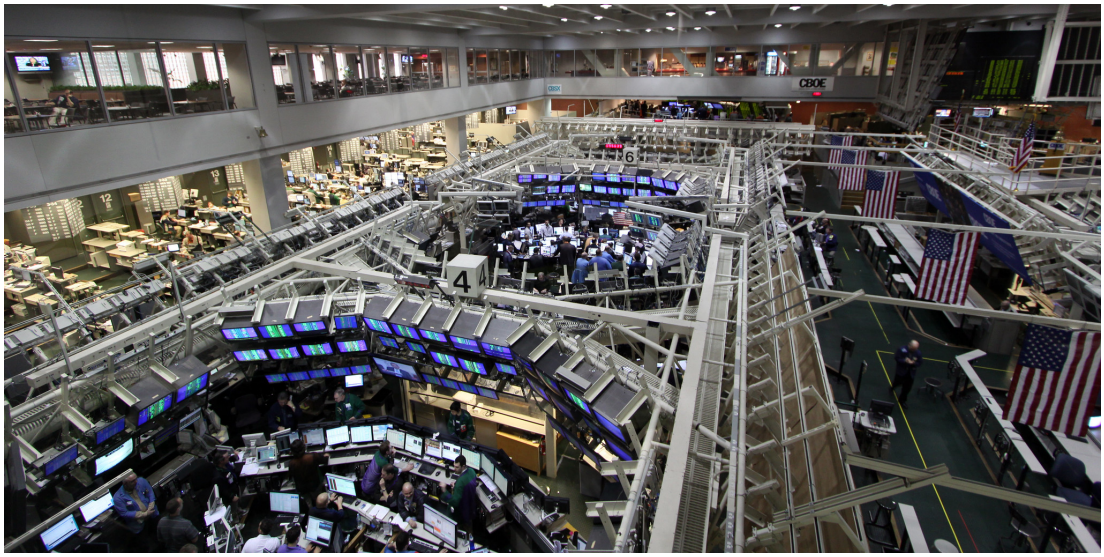


Figure 1.1: CBOE Floor 1, Chicago Board Options Exchange. Image licensed under CC BY-NC 2.0.

tends further back then you might expect, tracing the evolution of financial markets and the mathematical models that underpin them. Thales of Miletus, a pre-Socratic philosopher from ancient Greece, is often credited with creating the first known options contract [2]. Thales was not only a philosopher but also an astronomer and mathematician, and he used his knowledge of the seasons to predict that the upcoming olive harvest would be particularly bountiful. According to Aristotle, who recounted this story in his work "Politics", Thales secured the right to use a number of olive presses the following spring. He did this during the winter when demand for the presses was low, and thus he was able to secure the rights at a low cost. This is effectively an options contract, as Thales had the right, but not the obligation, to use the olive presses at a future date.

When spring came and the olive harvest was indeed abundant, there was high demand for the use of olive presses. Thales, holding the rights to use the presses, was able to rent them out at a much higher price than what he had initially paid. This clever move brought him considerable profit and, more importantly, demonstrated the basic principle of an options contract: securing the right to buy or sell an asset at a predetermined price, betting on the future movement of that asset's price. This anecdote, whether fact or legend, represents the earliest known example of an options contract and the rudimentary beginnings of option pricing.

In the Mediterranean, merchants in the 12th and 13th centuries used con-

tracts similar to options for transactions involving commodities. These contracts, known as "qirad" and "commenda" in the Islamic and Christian worlds respectively, allowed a financier to fund a merchant's voyage with the agreement that they would share in the profits upon the merchant's return [3]. This is not an option in the modern sense, but it does represent a form of contingent claim [1], which is a broader category that includes options.

However, it wasn't until the establishment of formal exchanges for trading commodities and securities, such as the Amsterdam Stock Exchange in the 17th century, that we see the widespread use of sophisticated derivative contracts, including options [4]. The Tulip Mania of the 1630s is one of the most famous early examples of this kind of trading.

The Tulip Mania that took place in the Dutch Golden Age during the 1630s provides one of the earliest examples of speculative trading involving options. Tulips had recently been introduced to Europe from the Ottoman Empire and quickly became a status symbol due to their vibrant colors and unique patterns. As demand for tulips grew, prices began to rise, and a speculative bubble formed.

During this period, traders began to use contracts for the future delivery of tulip bulbs, effectively creating a form of options contract. These contracts gave the holder the right, but not the obligation, to buy tulip bulbs at a predetermined price at a future date. Traders would buy these contracts hoping that the price of tulip bulbs would continue to rise. If the market price of tulip bulbs was higher than the contract price when the contract matured, the holder could buy bulbs at the contract price and then sell them at the market price for a profit.

However, when the tulip market collapsed in February 1637, the market price of tulip bulbs fell below the contract price of many of these options contracts. Holders of these contracts found themselves with the right to buy tulip bulbs at a price higher than the market price, making the contracts worthless. This led to significant financial losses for many traders and marked the end of the Tulip Mania.

This event serves as an early example of speculative trading involving options

[1] In finance, a contingent claim refers to a derivative with a future payoff contingent upon the valuation of a distinct "underlying" asset. More broadly, its payoff is predicated on the occurrence of a specific uncertain future event. Such claims are termed 'contingent' due to the conditional nature of their payoffs.

and illustrates the potential risks of such trading. It also highlights the fundamental principle of option pricing, i.e., the price of an option reflects the market's expectations about future price movements and the degree of uncertainty or risk associated with these movements.

The Chicago Board Options Exchange (CBOE) is a significant institution in the history of option pricing [5]. Established in 1973, the CBOE was the first organized exchange for trading options, marking a significant milestone in the development of modern financial markets.

Before the establishment of the CBOE, options were traded in over-the-counter (OTC) markets, which were less regulated and more opaque [6]. The creation of the CBOE brought standardization and transparency to options trading. It introduced standardized contracts, which specified the terms of the option such as the underlying asset, the strike price[2], and the expiration date. This standardization made it easier for traders to compare and trade options.

The CBOE also provided a centralized marketplace where buyers and sellers could come together to trade options. This improved liquidity, which is the ease with which an asset can be bought or sold without affecting its price. Improved liquidity makes markets more efficient and reduces the cost of trading.

Furthermore, the establishment of the CBOE coincided with the development of the Black-Scholes-Merton model, a mathematical model for pricing options. This model, which was published in the same year the CBOE was established, provided a theoretical framework for pricing options and has been instrumental in the growth and development of options trading.

Today, the CBOE is one of the largest options exchanges in the world, offering options on a wide range of assets including stocks, indices, and exchange-traded funds. It continues to play a key role in the world of finance and in the ongoing evolution of option pricing theory and practice.

1.1 Contributions of this Thesis

The computational intensity of financial models, especially when applied to large-scale financial data, poses significant challenges. Traditional computational platforms often struggle to deliver the required performance, lead-

[2]The strike price (also called the exercise price) is the predetermined price at which the underlying asset can be bought or sold when the option is exercised.

ing to delays and potential missed opportunities in fast-paced financial markets. Field Programmable Gate Array (FPGA) architectures have emerged as a promising solution. Their reprogrammable nature, high degree of parallelism, and energy efficiency make them an attractive choice for high-performance computing applications, including financial modeling.

Despite the potential advantages of FPGA architectures, their application in Pascal's Simplex-based multinomial option pricing remains largely unexplored. There is a need to investigate the design, implementation, and performance of FPGA architectures in this context, and to evaluate their potential benefits and challenges.

The main objectives of this research are to explore the design and implementation of FPGA architectures for Pascal's Simplex-based multinomial option pricing, evaluate the performance of these FPGA architectures in terms of speed, accuracy, and energy efficiency, and compare the performance of FPGA architectures with traditional computational platforms in the context of option pricing.

This research aims to answer several questions, including how FPGA architectures can be designed and implemented for Pascal's Simplex-based multinomial option pricing, what the performance of the implemented FPGA architectures is in terms of speed, accuracy, and energy efficiency, and how the performance of FPGA architectures compares with traditional computational platforms in the context of option pricing.

We believe this study has the potential to make significant contributions to the field of computational finance. By exploring the use of FPGA architectures for option pricing, it could offer new insights into the potential of FPGA architectures for complex financial modeling. The findings could also inform the development of more efficient and effective computational platforms for financial applications.

While this study focuses on the design, implementation, and performance evaluation of FPGA architectures for Pascal's Simplex-based multinomial option pricing, it does not cover other types of financial models or other potential applications of FPGA architectures. The performance evaluation is based on a specific set of benchmarks and may not reflect the performance under all possible scenarios.

1.2 Thesis Outline

The remainder of this thesis is organized as follows: Chapter 2 looks into the background, touching on option pricing, FPGA's role, and parallel computing frameworks like OpenCL and OneAPI. Chapter 3 surveys the current FPGA implementations in option pricing, highlighting the field's challenges and prospects. This survey also identifies the absence of FPGA architectures for Pascal's Simplex-based multinomial option pricing. The implementation of multinomial tree methods for option pricing is detailed in Chapter 4, extending from binomial models to multinomial ones, and emphasizing optimization and First-in First-out (FIFO) queue utilization. Chapters 5 and 6 explore the implementation of two-asset and multi-asset option pricing using OpenCL, respectively, presenting architectures, results, and analyses. Chapter 7 shifts focus to American option pricing with OneAPI, offering insights into FPGA architecture and performance. The thesis wraps up in Chapter 8, summarizing findings, discussing market implications, and proposing future research avenues.

Chapter 2

Background

This chapter outlines key foundational ideas that underpin our research. We aim to provide clarity on essential concepts, setting a base for the detailed discussions in subsequent chapters.

We begin by introducing modern option pricing. This will allow us to see its development over time, highlighting major achievements and identifying areas that presented challenges such as complex pricing models. Modern option pricing plays a key role in the later chapters, serving as a benchmark to compare against the model presented in our research.

We will then introduce hardware acceleration and Field Programmable Gate Arrays (FPGAs). These are crucial in computational processes and have significantly influenced the way calculations are carried out in various fields, including finance.

Next, the concept of parallel computing platforms will be explored. Specifically, we'll discuss OpenCL and OneAPI which are used in this research. These platforms have changed the dynamics of computational tasks by allowing multiple processes to run simultaneously, leading to more efficient results.

The combination of modern option pricing, hardware acceleration, FPGAs, and frameworks such as OpenCL and OneAPI fundamentally underpin the main contributions of this thesis and understanding them is crucial.

2.1 Modern Option Types

In this section, we will explore two primary categories: vanilla and exotic options. Vanilla options include the straightforward call and put European/American options. They have standard terms and are commonly traded. Exotic options, on the other hand, have more complex structures and are designed for specific purposes.

2.1.1 Vanilla Options

Vanilla options are the most basic type of options, providing fundamental strategies for investors to manage financial risks and speculate on asset price movements. To understand vanilla options, the reader needs to understand the difference between call/put options (related, respectively, to buying and selling an underlying asset) and European/American options (related to when the option can be exercised). These are described in this subsection as well as core terms for option pricing.

Key Terms in Option Contracts Understanding option contracts requires familiarity with the following terms:

Strike Price: The predetermined price at which the holder of an option can buy (in the case of a call option) or sell (in the case of a put option) the underlying asset. The strike price is agreed upon when the option contract is created and remains fixed throughout the option's lifetime. The profitability of an option is often determined by the relationship between the strike price and the market price of the underlying asset at expiration or exercise.

Exercise Date: The specific date or period during which the option can be exercised. For European options, this is limited to a single date at expiration. For American options, the exercise can occur at any time up to and including the expiration date, providing greater flexibility.

Expiration Date: The last date on which an option contract is valid. If the option is not exercised by this date, it expires worthless, and the holder loses the premium paid.

Premium: The price paid by the buyer of the option to the seller (writer) for the rights conferred by the option. The premium reflects the value of the option based on factors such as the underlying asset's price, strike price, time to

expiration, and market volatility.

Discounting: In financial options, discounting refers to the process of adjusting future cash flows to their present value using a discount rate. This concept is critical in option pricing, as the value of an option depends not only on the expected payoff at expiration but also on the time value of money. Discounting accounts for the fact that money available today is worth more than the same amount in the future due to earning potential, inflation, and risk factors.

Call Option A *European call option* gives its holder the right, but not the obligation, to purchase an asset at a predetermined price, known as the strike price. Investors buy call options anticipating that the asset's price will increase beyond the strike price before the option expires. If this occurs, the investor can either exercise the option to buy the asset at the lower strike price or sell the option contract itself for a profit. Conversely, if the asset's price remains below the strike price at expiration, the option becomes worthless, leading to a loss of the premium paid.

For example, suppose an investor buys an American call option for a stock with a strike price of €100, paying a premium of €5. If the stock's price rises to €120 before expiration, the investor can exercise the option, buying the stock at €100 and potentially selling it at €120, earning a profit of €15 (€20 price increase minus €5 premium), if we ignore the effect of discounting.

Put Option An *American put option* provides its holder the right to sell an asset at a predetermined strike price within a set timeframe (details on types of options are discussed in the next paragraph). An investor buys a put option if they anticipate a decrease in the asset's price below the strike price before expiration. In this scenario, they can either sell the asset at the higher strike price or sell the option contract if it gains value. However, if the asset's price stays above the strike price at expiration, the put option becomes worthless, and the investor loses the premium.

For example, suppose an investor buys a put option for a stock with a strike price of €100, paying a premium of €5. If the stock's price falls to €80 before expiration, the investor can exercise the option to sell the stock at €100, earning a profit of €15 (€20 price drop minus €5 premium), if we ignore the effect of discounting.

European Options European options[1] are a type of vanilla option that can only be exercised at the expiration date.

For example, suppose an investor purchases a European call option on a stock with a strike price of €100, paying a premium of €5. If the stock price rises to €120 by the option's expiration date, the investor can exercise the option on that date, purchasing the stock at €100 and potentially selling it at the market price of €120, earning a profit of €15 (€20 price difference minus €5 premium). However, if the stock price remains below €100 at expiration, the option expires worthless, and the investor loses the premium paid.

American Options American options differ from European options in that they can be exercised at any time up to the expiration date. This gives the holder more flexibility and makes American options potentially more valuable than European options. For example, suppose an investor purchases an American call option on a stock with a strike price of €100, paying a premium of €5. If the stock price rises to €120 before the expiration date, the investor has the option to exercise the call early, buying the stock at €100 and selling it at the current market price of €120, making a profit of €15 (€20 price increase minus €5 premium). Alternatively, the investor can hold onto the option until closer to expiration to see if the stock price rises further.

On the other hand, if the stock price remains below the strike price of €100, the investor may choose not to exercise the option, allowing it to expire, in which case the investor would only lose the premium paid.

The key advantage of the American option is the flexibility to exercise at any point during the option's lifetime, potentially allowing for early profit realization in favorable market conditions.

2.1.2 Exotic Options

Exotic options[8] offer more complex features compared to vanilla options, catering to sophisticated investors seeking tailored financial strategies.

[1] One possible origin for the name "European" is put forward in [7] where the economist Paul Samuelson stated "You are wasting your time - it takes a European mind to understand options.". Samuelson later clarified that he chose the term "European" for the relatively simple(-minded)-to-value option contract that can only be exercised at expiration and "American" for the considerably more-complex-to-value option contract.

Barrier Options A barrier option [9] is a type of exotic option where the option is activated or deactivated when the price of the underlying asset crosses a predefined barrier level. There are two main types of barrier options: knock-in options (which only become active if the asset price hits the barrier) and knock-out options (which expire worthless if the asset price hits the barrier).

For example, suppose an investor purchases an up-and-out American call option on a stock with a strike price of €100, a barrier level of €130, and a premium of €5. If the stock price rises to €120, the investor can exercise the option and buy the stock at €100 for a potential profit. However, if the stock price continues rising and hits the barrier level of €130, the option becomes worthless, and the investor loses the premium paid.

In this case, the barrier level acts as a "cap" on potential gains. Barrier options are typically less expensive than standard options due to this added complexity. Additionally, while not explicitly investigated in this thesis, the multinomial tree approach can be applied to Barrier Options.

Binary Options Binary options, also known as digital options [10], is a type of option where the payoff is a fixed amount or nothing at all, depending on whether the underlying asset's price is above or below a specific threshold (the strike price) at expiration. The payoff structure is "all or nothing," making it different from traditional options.

For example, suppose an investor purchases a binary call option on a stock with a strike price of €100 and a fixed payoff of €50, paying a premium of €10. If, at expiration, the stock price is above €100 (even by just one cent), the investor receives the full €50 payoff, resulting in a net profit of €40 (€50 payoff minus €10 premium). However, if the stock price is below €100 at expiration, the option expires worthless, and the investor loses the entire premium of €10.

Binary options offer simplicity in terms of potential outcomes: either the option finishes "in the money" with a fixed payout or "out of the money" with no payout, regardless of how much the price moves beyond the strike price.

Asian Options Asian options, also known as average options [11], is a type of exotic option where the payoff is based on the average price of the underlying asset over a specified period, rather than the price at a specific point in time. This averaging feature makes Asian options less volatile than stan-

dard European or American options, as the effect of sharp price movements is smoothed out.

For example, suppose an investor purchases an Asian call option on a stock with a strike price of €100. The option's payoff is based on the average stock price over the last 30 days before expiration. If the average stock price over this period is €110, the investor can exercise the option and receive a payoff of €10 (€110 average price minus €100 strike price). If the average price is below €100, the option expires worthless, and the investor loses the premium paid.

The key feature of Asian options is the use of an average price, which reduces the impact of short-term price volatility and provides a smoother, less risky payoff compared to traditional options.

Lookback Options A lookback option [12] is a type of exotic option where the payoff depends on the maximum or minimum price of the underlying asset during the option's life. This feature allows the holder to "look back" over the period and choose the most favorable price, making lookback options more flexible and potentially more profitable than standard options.

For example, suppose an investor purchases a lookback call option on a stock with a strike price of €100. Over the life of the option, the stock's price fluctuates, reaching a minimum of €90 and a maximum of €130. At expiration, the investor can "look back" and choose the lowest price of €90 as the effective strike price, allowing them to buy the stock at €90 and sell it at the maximum price of €130 (if the investor chooses to sell at €130), yielding a profit of €40 per share.

Lookback options provide significant advantages by allowing the investor to optimize the payoff based on historical price movements during the option's life. This eliminates the risk of poorly timed trades and can lead to greater profits than traditional options.

Rainbow Options Rainbow options [13] are an exotic option that derives its value from two or more underlying assets, often referred to as "colors" in the context of the option. The payoff depends on the performance of these multiple underlying assets, making it a popular instrument for investors looking to hedge or speculate on correlations between different assets.

For example, suppose an investor purchases a rainbow call option on two

stocks: Stock A and Stock B. The strike price for both stocks is set at €100, and the premium paid for the option is €10. At the option's expiration, Stock A is priced at €120, and Stock B is priced at €90.

The rainbow option's payoff could be based on the better-performing asset. In this case, since Stock A is trading at €120, the investor would exercise the option based on Stock A, yielding a profit of €20 (€120 - €100 strike price) minus the premium paid of €10, for a net profit (excluding the impact of the rate of interest) of €10.

Rainbow options provide the flexibility to benefit from multiple assets, and they are often used to hedge against unfavorable price movements in correlated assets or to speculate on the outperformance of one asset over another.

Compound Options Compound options [14] are an exotic option where the underlying asset is another option. This type of option gives the holder the right, but not the obligation, to buy or sell another option at a predetermined price. There are four main types of compound options: call on a call, call on a put, put on a call, and put on a put.

For example, consider a "call on a call" compound option. An investor buys a compound call option that gives them the right to purchase a standard European call option on a stock. The strike price of the compound option is €5, and the strike price of the underlying European call option is €100. The investor pays a premium of €2 for the compound option.

At the expiration of the compound option, the investor has the right to buy the European call option for €5. If the price of the underlying stock has risen to €110, the investor exercises the compound option, purchasing the European call for €5. The investor now holds a European call option that allows them to buy the stock at €100, which they can exercise at a suitable time before the expiry date for American options (or at the expiry date for European options) for a profit of €10 (€110 stock price - €100 strike price), minus the €5 paid for the European call and the €2 paid for the compound option, resulting in a net profit of €3.

Compound options are particularly useful in situations where there is significant uncertainty about the future value of the underlying option, allowing the investor to delay committing to a full position.

Options on Futures Options on futures [15] gives the holder the right, but not the obligation, to enter into a futures contract at a predetermined price (the strike price) before the option expires. These options are commonly used by investors to hedge or speculate on the future price of commodities, interest rates, or financial indices.

For example, suppose an investor purchases a call option on a crude oil futures contract with a strike price of €60 per barrel, expiring in three months. The premium paid for the option is €5. If the price of crude oil futures rises to €70 per barrel before the option's expiration, the investor can exercise the option and enter into the crude oil futures contract at the lower price of €60 per barrel.

Since the market price of the futures is €70, the investor can either hold the futures contract or immediately sell it at the current market price, making a profit of €10 per barrel (€70 - €60 strike price), minus the €5 premium, resulting in a net profit [2] of €5 per barrel. If the price of crude oil futures remains below €60, the option expires worthless, and the investor loses the premium paid.

Options on futures are a useful tool for investors who want to hedge against price movements in the futures market or speculate on the future direction of prices without taking an immediate futures position.

Employee Stock Options Employee stock options [16] are a form of compensation where a company grants its employees the right, but not the obligation, to purchase the company's stock at a predetermined price (the strike price) after a certain period, known as the vesting period. The value of these options depends on several factors, including the stock price, the strike price, and the option's expiration date.

For example, suppose an employee receives 1,000 stock options from their company, with a strike price of €50 per share, a vesting period of 3 years in the future from the time of granting, and an expiration date of 10 years from the grant date. The current market price of the stock at the time of the option being given is €45.

After 3 years, if the stock price has risen to €70 per share, the employee can exercise the options, purchasing 1,000 shares at the strike price of €50, and

[2] For the purposes of simplicity we assume an interest rate of 0%. In reality, the interest rate would need to be taken into account when calculating the net profit.

selling them at the market price of €70. The profit per share would be €20 (€70 market price - €50 strike price), resulting in a total profit of €20,000 before taxes.

However, if the stock price remains below €50 during the time the employee is eligible to exercise the options, the employee may choose not to exercise the options, which would result in the options expiring worthless. The value of employee stock options, therefore, depends on the performance of the company's stock and is typically valued using models such as the Black-Scholes model or binomial options pricing models, which account for factors like stock volatility, time to expiration, and risk-free interest rates.

2.2 Methods of Valuing Options

The valuation of options is influenced by various factors, including the underlying asset's price, volatility, time to expiration, and interest rates. This subsection presents the primary methodologies and models used in the financial world to determine the fair value of options.

2.2.1 Analytical Methods

Analytical methods are useful for accurately valuing options and other financial derivatives, providing structured approaches to estimate their prices based on market dynamics and underlying asset behaviors.

Black-Scholes Model The Black-Scholes model [17] is a widely-used method for valuing European call and put options. It assumes that the price of the underlying asset follows a geometric Brownian motion with constant volatility and interest rates.

Formula for European Call Option

$$C = S_0\Phi(d_1) - Xe^{-rT}\Phi(d_2) \tag{2.1}$$

where:

$$d_1 = \frac{\ln(S_0/X) + (r + \sigma^2/2)T}{\sigma\sqrt{T}}$$
$$d_2 = d_1 - \sigma\sqrt{T}$$

Here,

- C is the call option price,
- S_0 is the current stock price,
- X is the strike price,
- r is the risk-free interest rate,
- T is the time to maturity,
- σ is the volatility of the underlying asset,
- Φ is the cumulative distribution function of the standard normal distribution.

Example

Assume the current stock price (S_0) is €100, the strike price (X) is €90, the risk-free rate (r) is 5%, the volatility (σ) is 20%, and the time to expiration (T) is 1 year. Using the Black-Scholes formula, the value of the European call option can be computed as follows:

$$C = 100 \cdot \Phi(d_1) - 90 \cdot e^{-0.05} \cdot \Phi(d_2)$$

with d_1 and d_2 calculated using the respective formulas above. To solve for the value of the European call option using the Black-Scholes formula, we'll follow these steps:

Given Parameters:

- Current stock price (S_0) = €100
- Strike price (X) = €90
- Risk-free rate (r) = 5% or 0.05
- Volatility (σ) = 20% or 0.2
- Time to expiration (T) = 1 year

Step 1: Calculate d_1 and d_2

Compute d_1:

$$\begin{aligned} d_1 &= \frac{\ln\left(\frac{100}{90}\right) + \left(0.05 + \frac{1}{2} \times 0.2^2\right) \times 1}{0.2 \times \sqrt{1}} \\ &= \frac{\ln(1.1111) + (0.05 + 0.02)}{0.2} \\ &= \frac{0.10536 + 0.07}{0.2} \\ &= \frac{0.17536}{0.2} \\ &= 0.8768 \end{aligned}$$

Compute d_2:

$$\begin{aligned} d_2 &= d_1 - \sigma\sqrt{T} \\ &= 0.8768 - 0.2 \\ &= 0.6768 \end{aligned}$$

Step 2: Calculate $\Phi(d_1)$ and $\Phi(d_2)$

Using standard normal cumulative distribution function tables or a calculator:

Compute $\Phi(d_1)$:

$$\Phi(0.8768) \approx 0.8092$$

Compute $\Phi(d_2)$:

$$\Phi(0.6768) \approx 0.7494$$

Step 3: Calculate the Present Value of the Strike Price

$$\begin{aligned} \text{PV}(X) &= X \cdot e^{-rT} \\ &= 90 \cdot e^{-0.05} \\ &\approx 90 \cdot 0.95123 \\ &\approx 85.611 \end{aligned}$$

Step 4: Calculate the Option Price C

$$\begin{aligned} C &= 100 \cdot 0.8092 - 85.611 \cdot 0.7494 \\ &= 80.92 - 64.098 \\ &\approx 16.822 \end{aligned}$$

Answer:

The value of the European call option is approximately **€16.82**.

Binomial Option Pricing Model The binomial model [18] calculates option prices by simulating different possible price paths the underlying asset can take, using discrete time steps.

Key Formula for a European option

$$C = \frac{1}{(1+r)^N} \sum_{i=0}^{N} \binom{N}{i} p^i (1-p)^{N-i} C_i \tag{2.2}$$

where:

- C is the option price,
- N is the number of steps to maturity,
- C_i is the payoff at the i-th node at maturity,
- p is the risk-neutral probability of an upward movement.
- X is the strike price,
- r is the risk-free rate,

Example

Consider a European call option on a stock with a current price of €50, a strike price of €52, 2 time steps, and a 10% risk-free rate. At each time step, the stock can move up by 10% or down by 10%. We can use the binomial model to calculate the option price by constructing the binomial tree, determining the payoffs at each node, and working backwards. **Note:** The time steps in this example are relatively large for illustrative purposes. In practice, smaller and more frequent time steps (e.g., daily or weekly) would provide a

more accurate model for option pricing. The time step Δt is generally calculated as:

$$\Delta t = \frac{T}{N}$$

where T is the total time to maturity and N is the number of time steps.

We will use the binomial model to calculate the option price by constructing the binomial tree, determining the payoffs at each final node, and working backwards.

Step 1: Calculate the Risk-Neutral Probability p

Assuming each time step represents half a year (since total time is not specified), the per-period risk-free rate is calculated as:

$$R = e^{r\Delta t} = e^{0.10 \times 0.5} = e^{0.05} \approx 1.05127$$

The risk-neutral probability p is:

$$p = \frac{R - d}{u - d} = \frac{1.05127 - 0.90}{1.10 - 0.90} = \frac{0.15127}{0.20} = 0.75635$$

Step 2: Construct the Binomial Tree for Stock Prices

- **Time 0 ($t = 0$):**

$$S_0 = 50$$

- **Time 1 ($t = 1$):**

$$S_u = S_0 \times u = 50 \times 1.10 = 55$$
$$S_d = S_0 \times d = 50 \times 0.90 = 45$$

- **Time 2 ($t = 2$):**

$$S_{uu} = S_u \times u = 55 \times 1.10 = 60.50$$
$$S_{ud} = S_u \times d = 55 \times 0.90 = 49.50$$
$$S_{du} = S_d \times u = 45 \times 1.10 = 49.50$$
$$S_{dd} = S_d \times d = 45 \times 0.90 = 40.50$$

Step 3: Calculate Option Payoffs at Maturity (Time 2)

- **At** $S_{uu} = 60.50$**:**

$$C_{uu} = \max(S_{uu} - X, 0) = \max(60.50 - 52, 0) = 8.50$$

- **At** $S_{ud} = S_{du} = 49.50$**:**

$$C_{ud} = C_{du} = \max(49.50 - 52, 0) = 0$$

- **At** $S_{dd} = 40.50$**:**

$$C_{dd} = \max(40.50 - 52, 0) = 0$$

Step 4: Work Backwards to Calculate Option Prices at Earlier Nodes

At Time 1:

- **At Node** $S_u = 55$**:**

$$\begin{aligned} C_u &= e^{-r\Delta t}[p \times C_{uu} + (1-p) \times C_{ud}] \\ &= e^{-0.05}[0.75635 \times 8.50 + (1 - 0.75635) \times 0] \\ &= 0.95123 \times (0.75635 \times 8.50) \\ &= 0.95123 \times 6.429 \\ &= 6.116 \end{aligned}$$

- **At Node** $S_d = 45$**:**

$$\begin{aligned} C_d &= e^{-r\Delta t}[p \times C_{du} + (1-p) \times C_{dd}] \\ &= e^{-0.05}[0.75635 \times 0 + (1 - 0.75635) \times 0] \\ &= 0 \end{aligned}$$

At Time 0:

$$\begin{aligned} C_0 &= e^{-r\Delta t}[p \times C_u + (1-p) \times C_d] \\ &= e^{-0.05}[0.75635 \times 6.116 + (1 - 0.75635) \times 0] \\ &= 0.95123 \times (0.75635 \times 6.116) \\ &= 0.95123 \times 4.626 \\ &= 4.400 \end{aligned}$$

Answer

The value of the European call option is approximately **€4.40**.

2.2.1.1 Monte Carlo Methods

Monte Carlo simulation [19] is a numerical method that uses random sampling to estimate the value of an option by simulating different price paths of the underlying asset.

Key Formula for a European call option

$$C = e^{-rT} \frac{1}{N} \sum_{i=1}^{N} \max(S_T^{(i)} - X, 0) \tag{2.3}$$

where:

- N is the number of simulations,
- $S_T^{(i)}$ is the simulated asset price at maturity,
- X is the strike price,
- r is the risk-free rate,
- T is the time to maturity.

Example

Consider a European call option on a stock with a strike price of €100 and a time to maturity of 1 year. The stock price follows a geometric Brownian motion with an initial price of €100, volatility of 30%, and a risk-free rate of 5%. Using Monte Carlo simulation, we generate 10,000 possible price paths

for the stock and compute the payoff for each path. The average payoff is then discounted back to the present value to get the option price.

Example

Consider a European call option on a stock with:

- Strike price (X) = €100
- Time to maturity (T) = 1 year
- Initial stock price (S_0) = €100
- Volatility (σ) = 30% or 0.30
- Risk-free rate (r) = 5% or 0.05
- Number of simulations (N) = 10,000

Step 1: Simulate Stock Price Paths

The stock price follows a geometric Brownian motion given by:

$$S_T = S_0 \cdot e^{\left(r - \frac{1}{2}\sigma^2\right)T + \sigma W_T} \tag{2.4}$$

where W_T is a random variable following a normal distribution with mean 0 and variance T.

For each simulation i, generate a random standard normal variable Z_i and compute:

$$S_T^{(i)} = S_0 \cdot e^{\left(r - \frac{1}{2}\sigma^2\right)T + \sigma\sqrt{T}Z_i} \tag{2.5}$$

Step 2: Calculate Payoff for Each Simulation

For each simulated stock price $S_T^{(i)}$, compute the payoff:

$$\text{Payoff}_i = \max(S_T^{(i)} - X, 0) \tag{2.6}$$

Step 3: Estimate the Option Price

Calculate the average payoff and discount it back to present value:

$$\text{Average Payoff} = \frac{1}{N}\sum_{i=1}^{N}\text{Payoff}_i$$

$$C = e^{-rT} \times \text{Average Payoff}$$

Step 4: Numerical Result

Since performing 10,000 simulations manually is impractical, we can illustrate with a smaller number of simulations, say $N = 5$.

Simulation Details:

- Generate 5 random standard normal variables Z_i.
- Calculate $S_T^{(i)}$ for each i.
- Compute the payoffs and average them.

Sample Calculations:

Assuming the following random numbers (from a standard normal distribution):

$$Z_1 = 0.5$$
$$Z_2 = -0.3$$
$$Z_3 = 1.2$$
$$Z_4 = -1.0$$
$$Z_5 = 0.0$$

Compute $S_T^{(i)}$ for each i:

$$\begin{aligned} S_T^{(1)} &= 100 \times e^{\left(0.05-\frac{1}{2}\times 0.3^2\right)\times 1+0.3\times 1\times 0.5} \\ &= 100 \times e^{(0.05-0.045)+0.15} \\ &= 100 \times e^{0.155} \approx 100 \times 1.167 \approx 116.7 \end{aligned}$$

$$\begin{aligned} S_T^{(2)} &= 100 \times e^{(0.05-0.045)+0.3\times(-0.3)} \\ &= 100 \times e^{0.005-0.09} \\ &= 100 \times e^{-0.085} \approx 100 \times 0.918 \approx 91.8 \end{aligned}$$

$$\begin{aligned} S_T^{(3)} &= 100 \times e^{0.005+0.3\times 1.2} \\ &= 100 \times e^{0.005+0.36} \\ &= 100 \times e^{0.365} \approx 100 \times 1.441 \approx 144.1 \end{aligned}$$

$$\begin{aligned} S_T^{(4)} &= 100 \times e^{0.005+0.3\times(-1.0)} \\ &= 100 \times e^{0.005-0.30} \\ &= 100 \times e^{-0.295} \approx 100 \times 0.745 \approx 74.5 \end{aligned}$$

$$\begin{aligned} S_T^{(5)} &= 100 \times e^{0.005+0.3\times 0} \\ &= 100 \times e^{0.005} \approx 100 \times 1.005 \approx 100.5 \end{aligned}$$

Compute the payoffs:

$$\begin{aligned} \text{Payoff}_1 &= \max(116.7 - 100, 0) = 16.7 \\ \text{Payoff}_2 &= \max(91.8 - 100, 0) = 0 \\ \text{Payoff}_3 &= \max(144.1 - 100, 0) = 44.1 \\ \text{Payoff}_4 &= \max(74.5 - 100, 0) = 0 \\ \text{Payoff}_5 &= \max(100.5 - 100, 0) = 0.5 \end{aligned}$$

Calculate the average payoff:

$$\text{Average Payoff} = \frac{16.7 + 0 + 44.1 + 0 + 0.5}{5} = \frac{61.3}{5} = 12.26$$

Discount the average payoff to present value:

$$C = e^{-0.05 \times 1} \times 12.26 = e^{-0.05} \times 12.26 \approx 0.95123 \times 12.26 \approx 11.67$$

Answer

The estimated value of the European call option using Monte Carlo simulation is approximately **€11.67** (based on 5 simulations).

Note that with 10,000 simulations, the estimate is **€14.41** [3]. Furthermore, if the number of simulations approaches infinity, the Monte Carlo price will converge to the Black-Scholes price (introduced in the next subsection). When we compute the option price using the Black-Scholes formula the value is **€14.19** which is much closer to the figure computed by running 10,000 simulations compared to 5 simulations.

2.2.1.2 Finite Difference Methods

Finite Difference Methods are numerical techniques used to solve partial differential equations (PDEs), like the Black-Scholes PDE, for option pricing [20].

Black-Scholes PDE

$$\frac{\partial C}{\partial t} + rS\frac{\partial C}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 C}{\partial S^2} = rC \tag{2.7}$$

FDM discretizes this PDE in both time and stock price, and solves it numerically by working backwards from the expiration date.

Example

Assume we are pricing a European call option with a strike price of €50, a volatility of 25%, a risk-free rate of 5%, and a time to expiration of 1 year. We can use FDM to discretize the time and stock price space, applying boundary

[3] This figure was carried out by running the simulation in Python.

conditions, and solving for the option price iteratively using either explicit, implicit, or Crank-Nicolson methods.

The workings for this method are quite long so we just provide the result. To obtain the option price, we numerically solved the Black-Scholes partial differential equation (PDE) using the finite difference method. By discretizing the stock price and time domains into a grid, we applied the Crank-Nicolson scheme for its stability and accuracy. Appropriate boundary and initial conditions were established based on the option's payoff at expiration and behavior at extreme stock prices. This transformed the PDE into a system of linear equations, which we solved iteratively. After performing all iterations and interpolations, the estimated value of the European call option is approximately **€52.45**.

2.2.2 Lattice Models

Lattice models, such as the binomial and trinomial tree models [21], are used to value options by discretizing the time to expiration into a finite number of steps. These models work by simulating the different possible paths that the price of the underlying asset could take and calculating the value of the option at each point in the lattice.

In a binomial tree, the underlying asset can move up or down by specific factors at each step, while a trinomial tree, for the case of one asset, allows for three possible movements: up, down, or remaining the same.

Example: Binomial Tree Model

Using the binomial tree model, we first calculate the possible stock prices at each node of the tree. At expiration (after two time steps), we determine the option's payoff at each node and then work backwards through the tree to calculate the option's value at the current time.

The value of the European call option is determined by discounting the expected payoffs at each node back to the present, using the risk-free rate.

Consider a European call option on a stock with the following parameters:

- **Current stock price** (S_0): €50
- **Strike price** (X): €52

- **Time to expiration** (T): 1 year
- **Risk-free rate** (r): 5% or 0.05
- **Upward movement factor** (u): 1.10 (10% up)
- **Downward movement factor** (d): 0.90 (10% down)
- **Number of time steps** (n): 2

We will use the binomial tree model to calculate the option price by constructing the binomial tree, determining the payoffs at each node, and working backwards.

Step 1: Calculate the Risk-Neutral Probability p

Each time step is:

$$\Delta t = \frac{T}{n} = \frac{1}{2} = 0.5 \text{ years}$$

The per-period risk-free rate is:

$$R = e^{r\Delta t} = e^{0.05\times 0.5} = e^{0.025} \approx 1.02532$$

The risk-neutral probability p is:

$$p = \frac{R - d}{u - d} = \frac{1.02532 - 0.90}{1.10 - 0.90} = \frac{0.12532}{0.20} = 0.6266$$

Step 2: Construct the Binomial Tree for Stock Prices

At Time 0 ($t = 0$):

$$S_0 = 50$$

At Time 1 ($t = 1$):

$$S_u = S_0 \times u = 50 \times 1.10 = 55$$
$$S_d = S_0 \times d = 50 \times 0.90 = 45$$

At Time 2 ($t = 2$):

$$S_{uu} = S_u \times u = 55 \times 1.10 = 60.50$$
$$S_{ud} = S_u \times d = 55 \times 0.90 = 49.50$$
$$S_{du} = S_d \times u = 45 \times 1.10 = 49.50$$
$$S_{dd} = S_d \times d = 45 \times 0.90 = 40.50$$

Step 3: Calculate Option Payoffs at Maturity (Time 2)

At $S = 60.50$:

$$C_{uu} = \max(S_{uu} - X, 0) = \max(60.50 - 52, 0) = 8.50$$

At $S = 49.50$:

$$C_{ud} = C_{du} = \max(49.50 - 52, 0) = 0$$

At $S = 40.50$:

$$C_{dd} = \max(40.50 - 52, 0) = 0$$

Step 4: Work Backwards to Calculate Option Prices at Earlier Nodes

At Time 1:

Node at $S = 55$:

$$\begin{aligned} C_u &= e^{-r\Delta t}[p \times C_{uu} + (1-p) \times C_{ud}] \\ &= e^{-0.05\times 0.5}[0.6266 \times 8.50 + (1 - 0.6266) \times 0] \\ &= e^{-0.025}[5.3261] \\ &= 0.97531 \times 5.3261 \\ &\approx 5.194 \end{aligned}$$

Node at $S = 45$:

$$\begin{aligned} C_d &= e^{-r\Delta t}[p \times C_{du} + (1-p) \times C_{dd}] \\ &= e^{-0.025}[0.6266 \times 0 + 0.3734 \times 0] \\ &= 0 \end{aligned}$$

At Time 0:

$$\begin{aligned} C_0 &= e^{-r\Delta t}[p \times C_u + (1-p) \times C_d] \\ &= e^{-0.025}[0.6266 \times 5.194 + 0.3734 \times 0] \\ &= 0.97531 \times 3.2544 \\ &\approx 3.173 \end{aligned}$$

Answer

The value of the European call option using the binomial tree model is approximately **€3.17**.

Example: Trinomial Tree Model[4]

Suppose we have the same European call option, but now we use a trinomial tree model where the stock price can move up by 10%, down by 10%, or remain the same at each step.

At each node, there are three possible future stock prices, and we calculate the option's value at each node by working backwards from the option's expiration, similar to the binomial tree model. The trinomial tree provides a more detailed estimate of the option's value by allowing for more possible price paths.

Suppose we have the same European call option, but now we use a trinomial tree model where the stock price can move up by 10%, remain the same, or down by 10% at each step (the other parameters remain the same as the previous example).

Step 1: Define Movement Factors

Movement Factors:

- Upward movement (u): 1.10
- No movement (m): 1.00

[4]This approach could also be referred to as a trinary tree, as there are three different paths the price can take at each node. However, later in the thesis, we will explore a trinomial tree model, which instead models the movement of two assets.

- Downward movement (d): 0.90

Step 2: Calculate Risk-Neutral Probabilities

Substitute $u = 1.10$, $m = 1.00$, $d = 0.90$, and $R = e^{r\Delta t} = e^{0.025} \approx 1.02532$.

Set up the equations:

$$p_u + p_m + p_d = 1 \quad (1)$$
$$1.10p_u + 1.00p_m + 0.90p_d = 1.02532 \quad (2)$$

Assuming $p_m = p_d$ (for the purposes of this worked example only), we have:

$$p_u + 2p_m = 1 \quad \text{(from equation 1)}$$

$$1.10p_u + 1.90p_m = 1.02532 \quad \text{(from equation 2)}$$

Express p_u in terms of p_m:

$$p_u = 1 - 2p_m$$

Substitute p_u into the second equation:

$$\begin{aligned} 1.10(1 - 2p_m) + 1.90p_m &= 1.02532 \\ 1.10 - 2.20p_m + 1.90p_m &= 1.02532 \\ 1.10 - 0.30p_m &= 1.02532 \\ -0.30p_m &= 1.02532 - 1.10 \\ p_m &= \frac{1.10 - 1.02532}{0.30} \\ p_m &= \frac{0.07468}{0.30} \approx 0.24893 \end{aligned}$$

Calculate p_u and p_d:

$$p_u = 1 - 2p_m = 1 - 2 \times 0.24893 \approx 0.50214$$
$$p_d = p_m \approx 0.24893$$

Step 3: Construct the Trinomial Tree for Stock Prices

At Time 0 ($t = 0$):

$$S_0 = 50$$

At Time 1 ($t = 1$):

$$S_u = S_0 \times u = 50 \times 1.10 = 55$$
$$S_m = S_0 \times m = 50 \times 1.00 = 50$$
$$S_d = S_0 \times d = 50 \times 0.90 = 45$$

At Time 2 ($t = 2$):

- From S_u:

$$S_{uu} = 55 \times 1.10 = 60.50$$
$$S_{um} = 55 \times 1.00 = 55$$
$$S_{ud} = 55 \times 0.90 = 49.50$$

- From S_m:

$$S_{mu} = 50 \times 1.10 = 55$$
$$S_{mm} = 50 \times 1.00 = 50$$
$$S_{md} = 50 \times 0.90 = 45$$

- From S_d:

$$S_{du} = 45 \times 1.10 = 49.50$$
$$S_{dm} = 45 \times 1.00 = 45$$
$$S_{dd} = 45 \times 0.90 = 40.50$$

Step 4: Calculate Option Payoffs at Maturity (Time 2)

Calculate the option payoff C at each node at $t = 2$:

- Nodes where $S \geq 52$:

$$\begin{aligned} C_{uu} &= \max(60.50 - 52, 0) = 8.50 \\ C_{um} &= \max(55 - 52, 0) = 3.00 \\ C_{mu} &= \max(55 - 52, 0) = 3.00 \end{aligned}$$

- Nodes where $S < 52$:

$$\begin{aligned} C_{ud} &= \max(49.50 - 52, 0) = 0 \\ C_{du} &= \max(49.50 - 52, 0) = 0 \\ C_{mm} &= \max(50 - 52, 0) = 0 \\ C_{md} &= \max(45 - 52, 0) = 0 \\ C_{dm} &= \max(45 - 52, 0) = 0 \\ C_{dd} &= \max(40.50 - 52, 0) = 0 \end{aligned}$$

Step 5: Work Backwards to Calculate Option Prices at Earlier Nodes

At Time 1:

Node at $S = 55$:

$$\begin{aligned} C_u &= e^{-r\Delta t} \left[p_u \times C_{uu} + p_m \times C_{um} + p_d \times C_{ud} \right] \\ &= e^{-0.025} \left[0.50214 \times 8.50 + 0.24893 \times 3.00 + 0.24893 \times 0 \right] \\ &= 0.97531 \left[4.26819 + 0.74679 + 0 \right] \\ &= 0.97531 \times 5.01498 \\ &\approx 4.890 \end{aligned}$$

Node at $S = 50$:

$$\begin{aligned} C_m &= e^{-r\Delta t} \left[p_u \times C_{mu} + p_m \times C_{mm} + p_d \times C_{md} \right] \\ &= e^{-0.025} \left[0.50214 \times 3.00 + 0.24893 \times 0 + 0.24893 \times 0 \right] \\ &= 0.97531 \times 1.50642 \\ &\approx 1.469 \end{aligned}$$

Node at $S = 45$:

$$\begin{aligned} C_d &= e^{-r\Delta t} \left[p_u \times C_{du} + p_m \times C_{dm} + p_d \times C_{dd} \right] \\ &= e^{-0.025} \left[0.50214 \times 0 + 0.24893 \times 0 + 0.24893 \times 0 \right] \\ &= 0 \end{aligned}$$

At Time 0:

$$\begin{aligned} C_0 &= e^{-r\Delta t} \left[p_u \times C_u + p_m \times C_m + p_d \times C_d \right] \\ &= e^{-0.025} \left[0.50214 \times 4.890 + 0.24893 \times 1.469 + 0.24893 \times 0 \right] \\ &= 0.97531 \left[2.45473 + 0.36601 + 0 \right] \\ &= 0.97531 \times 2.82074 \\ &\approx 2.752 \end{aligned}$$

Answer

The value of the European call option using the trinomial tree model is approximately **€2.75**.

2.2.3 Risk-neutral Valuation

Risk-neutral valuation is a method used to value options by adjusting the probabilities of the different outcomes so that they reflect a "risk-neutral" world [22]. In a risk-neutral world, investors are indifferent to risk, and the expected return on all assets is the risk-free rate. The risk-neutral probabilities are used to calculate the expected payoff of the option, which is then discounted back to the present value using the risk-free rate.

Key Formula

For a binomial tree model, the risk-neutral probability of an upward movement (p) can be calculated as:

$$p = \frac{e^{r\Delta t} - d}{u - d} \tag{2.8}$$

where:

- r is the risk-free rate,
- Δt is the time step,
- u is the upward movement factor,
- d is the downward movement factor.

Example: Risk-neutral Valuation in a Binomial Tree

Consider the same European call option as in the binomial tree example above, but now apply risk-neutral valuation. Suppose the stock price can either move up by 10% or down by 10%, and the risk-free rate is 5%.

The risk-neutral probability of an upward movement (p) is calculated using the formula:

$$p = \frac{e^{0.05} - 0.9}{1.1 - 0.9}$$

Once the risk-neutral probability is determined, we can calculate the expected payoff of the option at each node, discount it back to the present using the risk-free rate, and thereby determine the current option value.

In risk-neutral valuation, the option's value is the discounted expected payoff under the assumption that all investors are indifferent to risk.

2.3 Introduction to Hardware Acceleration

Hardware acceleration has gained momentum due to the increasing limitations and technological constraints limiting the performance and energy scaling of general-purpose units. The emergence of the so-called "dark silicon" issue and diminishing returns of an ever-increasing core count have led hardware manufacturers to switch their focus to accelerators [23]. These are special-purpose hardware structures separated from the CPU, exhibiting a high degree of variability.

The trend towards hardware acceleration is a response to the same limitations occurring in general-purpose units, affecting also some accelerators with a relatively wide application spectrum. These accelerators not only target special-purpose tasks offloaded by the cores, but they feature also some specific hardware blocks, even more specialized, that push the boundaries of performance and energy efficiency [23].

The challenges connected to these accelerators concern a cooperation between these different modules, as this heavily affects performance and efficiency of applications that take advantage of more than one module. The limits of the multicore approach alone are not sufficient to satisfy the worldwide growing demand for computing power and performance. A similar scenario emerged in High Performance Computing (HPC), in embedded and, in general, in energy-constrained contexts, due to the difficulty of continuously performance per Watt metrics of general-purpose processors [23].

A possible way to overcome these limits is to relax the general-purpose trait, so as to leverage the possibility of customizing the hardware. This has led to the development of a variety of hardware accelerators, each with its own unique characteristics and advantages. In this section, we will explore the taxonomy, trends, challenges, and perspectives of hardware accelerators, providing a comprehensive overview of this rapidly evolving field.

2.3.1 Different Types of Hardware Accelerators

Hardware accelerators are specialized computing devices designed to perform specific tasks more efficiently than general-purpose Central Processing Units (CPUs). They are particularly useful in applications requiring intensive computation, high throughput, and low latency.

2.3.1.1 Taxonomy of Hardware Accelerators

General aspects describe an accelerator with a very high level of abstraction, allowing quick contextualization. This includes the accelerator's launch year, the type of accelerator (such as ASIC or FPGA), and the application domain it is designed for.

Host coupling refers to the manner in which the accelerator is connected to the host system. This includes the physical coupling (on-chip or off-chip), the logical coupling (coprocessor or memory-mapped), and the communication interface (PCIe or AXI).

Architecture covers the internal structure of the accelerator. This includes the type of architecture (spatial or systolic), the level of reconfigurability (none or fine-grain), and the memory hierarchy (scratchpad or cache).

Software aspects pertain to the software support available for the accelerator.

This includes the programming model (imperative or dataflow), the ecosystem (open-source or vendor-provided), and the granularity of tasks that can be offloaded to the accelerator (wired task, function-level, or application-level).

2.3.1.2 Specific Hardware Accelerators

ASICs (Application-Specific Integrated Circuits) are chips designed for specific applications, offering the most efficient solutions but lacking reconfigurability and programmability, and incurring high Non-Recurring Engineering (NRE) costs [24].

FPGAs (Field-Programmable Gate Arrays) are reconfigurable accelerators composed of numerous interconnected logic blocks. These blocks and interconnects provide fine-grain reconfigurability, supporting a wide range of applications and making them suitable for general-purpose computing, although reconfiguration time is typically high [25].

Spatial accelerators feature an array of simple Arithmetic Logic Unit (ALU)-like Processing Elements (PEs) that communicate directly through Networks-on-Chip (NoCs), buses, or ad-hoc inter-PE connections. These PEs can form chains and exchange data in a dataflow manner [26].

CGRAs (Coarse-Grained Reconfigurable Arrays) are a specialized form of spatial accelerators, organized as arrays of reconfigurable PEs connected by a reconfigurable interconnect. They provide coarse-grain reconfigurability, offering domain-specific flexibility [27].

Systolic accelerators perform main computations with a 1D or 2D array of arithmetic units controlled in lockstep. Partial results flow through the array in a wave-like fashion until the final level produces the complete output [28].

Vector accelerators use vector processors, each containing one or more vector cores with vector registers capable of holding numerous primitive elements. Single instructions can be applied to all elements within a vector register [29].

Manycore accelerators feature hundreds or even thousands of independent physical cores, sometimes grouped into multi-processors. This paradigm extends the multi-core concept, offering greater parallelism suitable for data-parallel workloads [30].

GPUs (Graphics Processing Units) are prominent examples of manycore accelerators, sharing similarities with vector accelerators. They consist of multi-

ple multi-processors made up of simple, in-order cores that leverage massive thread-level parallelism [31].

Each of these types of accelerators have a use in finance and examples can be found for all.

2.3.1.3 Performance Improvements

In addition to the growing trend of incorporating simple accelerators into heterogeneous SoCs, there's a parallel development where some accelerators are adopting a fractal-like design pattern. This means that as accelerators gain prominence due to technological constraints on general-purpose units, some are developing specialized modules within themselves for enhanced efficiency. This hierarchical architecture indicates that the challenges faced by general-purpose units also impact certain versatile accelerators. In response, they dive deeper into internal specialization. Consequently, these accelerators not only handle specific tasks from the cores but also introduce highly specialized hardware components that optimize performance and energy efficiency. A key challenge lies in ensuring seamless collaboration between these modules to maximize application performance and efficiency [23].

2.3.2 Challenges Associated with Hardware Acceleration

While hardware acceleration offers substantial performance and efficiency benefits, it is not without its challenges. These challenges span technical, economic, and logistical domains, posing significant hurdles to the widespread adoption and optimization of hardware accelerators.

2.3.2.1 Technical Challenges

Hardware acceleration, while offering significant performance and efficiency improvements, presents several technical challenges. One of the primary obstacles is integration complexity, as incorporating accelerators into existing systems often requires elaborate design considerations to ensure seamless operation, leading to increased design and validation times [32].

Scalability is another major concern [33], particularly as systems grow and evolve. Ensuring that accelerators can scale efficiently involves addressing both hardware scalability and the capacity to handle increasing data volumes. Additionally, power consumption poses a challenge, especially in mobile or

power-constrained environments, as accelerators can introduce significant power management issues [34].

Portability of software between different accelerators is a significant hurdle due to the specialized nature of hardware components [35]. Latency issues also arise from the communication between the main processor and the accelerator, potentially offsetting the performance gains achieved by the accelerator [36].

Additionally, striking the right balance between customization and generalization remains a persistent challenge. Designing a highly specialized accelerator for a specific task versus a more generalized one capable of handling a range of tasks requires careful consideration [37].

Addressing these technical challenges requires a combination of innovative hardware design, efficient software programming, and a deep understanding of the application domain. As the field of hardware acceleration continues to evolve, ongoing research and development efforts are crucial to overcoming these obstacles and unlocking the full potential of accelerators.

2.3.2.2 Economic and Logistical Challenges

While the technical aspects of hardware acceleration are important, the economic and logistical challenges cannot be overlooked. These challenges often dictate the feasibility and adoption rate of new accelerator technologies. Key economic and logistical challenges include:

Designing, prototyping, and testing new accelerators can be capital-intensive, making securing funding and ensuring a return on investment crucial [38]. Advanced accelerators often require cutting-edge manufacturing processes, which are both expensive and limited in capacity. Additionally, ensuring a consistent and reliable supply of components for accelerators can be challenging, especially during global disruptions or geopolitical tensions[39].

Introducing a new accelerator to the market requires not only technical superiority but also effective marketing, user training, and support infrastructure [40]. Integrating accelerators with existing legacy systems poses economic challenges due to potential compatibility issues and the costs associated with system upgrades [41]. Furthermore, navigating the complex landscape of patents, licensing agreements, and intellectual property rights can introduce significant economic challenges [42]. Addressing these economic and logis-

tical challenges requires a holistic approach that combines technical prowess with strategic planning, market analysis, and robust financial management.

2.4 Field Programmable Gate Arrays (FPGAs)

The journey of FPGAs in computational finance highlights the dynamic nature of both finance and technology. As the financial world evolved with the advent of more complicated financial products and a heightened emphasis on real-time data processing, there was a pressing need for computational tools that could keep pace. Enter FPGAs.

The 1980s and 1990s saw a boom in the financial markets, marked by the introduction of complex derivatives and an increased focus on quantitative analysis. Traditional CPUs were the workhorses of financial computations during this era. However, with the ever-growing demand for faster computations and more complex modeling, the limitations of CPUs became evident. The inherently serial nature of CPUs made it challenging to deal with tasks that required parallel processing capabilities.

In contrast, FPGAs, which had initially found their footing in the area of defense and telecommunications, began to catch the eye of financial institutions in the late 1990s and early 2000s. Their parallelism, adaptability, and the ability to reconfigure in response to changing requirements made them particularly appealing.

2.4.1 Introduction to FPGAs

FPGAs are semiconductor devices that can be reprogrammed post-manufacturing to embody nearly any type of digital system or circuit. In situations where production volumes are low to medium, FPGAs offer a cost-effective and swift solution, especially when compared to ASICs. The latter often demands significant investment in both time and money before the initial device becomes available. In contrast, FPGAs can be configured in a matter of moments and generally range in price from a couple of hundred to several thousand Euros. One notable feature of FPGAs is their ability to undergo partial reconfiguration, allowing part of the FPGA to be altered while the remainder continues to function. This adaptability also facilitates easy product updates through the upload of a revised application bitstream.

However, the adaptability that is a hallmark of FPGAs also contributes to their limitations. Their versatile nature can render them larger in size, somewhat slower, and more energy-intensive than ASICs. This disparity is mainly attributed to the programmable routing interconnect within FPGAs, which occupies nearly 90% of their overall area [43]. Still, the rapid deployment and cost benefits of FPGAs make them a notable choice for the realization of digital systems.

Typical components within an FPGA include:

- Logic blocks that execute logic operations.
- Programmable pathways connecting these logic operations.
- I/O components, linked to logic blocks via the routing interconnect, responsible for external connections.

An illustrative representation of a generic FPGA might feature configurable logic blocks (CLBs) organized in a two-dimensional matrix, connected by adjustable routing mechanisms. The I/O components would be positioned on the outer edges of this matrix, also linked to the adjustable routing. The term "programmable" or "reconfigurable" in FPGAs highlights their capacity to embody different functions after their manufacturing phase. A generic FPGA architecture is shown in Figure 2.1.

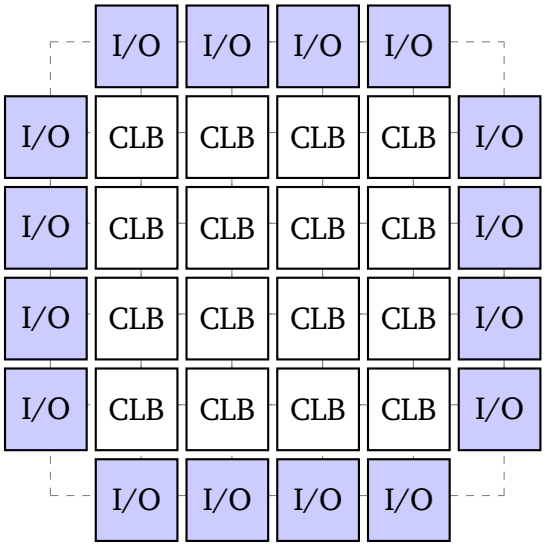


Figure 2.1: A generic FPGA architecture consisting of a matrix of configurable logic blocks (CLBs) connected by adjustable routing mechanisms, with I/O components on the outer edges.

2.4.1.1 Configurable Logic Blocks

The configurable logic blocks (CLBs) are the core of an FPGA and are responsible for implementing the actual logic functions [44]. A CLB typically contains a small lookup table (LUT), which can be configured to implement any Boolean logic function. Modern FPGAs often use multiple CLBs clustered together, along with Flip-Flops (FFs) for storage, to create more complex and flexible logic functions. Each CLB is configurable, allowing designers to specify the functionality required for different parts of the digital circuit.

2.4.1.2 Programmable Routing

The programmable routing network interconnects the logic blocks and I/O blocks, allowing the designer to route signals between them as needed [45]. The routing consists of a combination of wires and programmable switches that form the desired connections. This routing flexibility is key to FPGAs, but it also contributes significantly to the size, speed, and power consumption of the device. Most of the area in an FPGA is taken up by the routing resources, which can make up about 90% of the total chip area.

2.4.1.3 Input/Output (I/O) Blocks

The I/O blocks are responsible for interfacing the internal logic of the FPGA with external devices and systems [46]. These blocks are located at the periphery of the FPGA grid and are connected to the logic blocks through the routing interconnect. The I/O blocks are programmable, allowing for different types of signaling standards, voltage levels, and communication protocols, providing further flexibility in how the FPGA interacts with other components.

2.4.1.4 Reconfigurability

The reconfigurability of an FPGA allows the device to be programmed and reprogrammed multiple times, enabling changes to the logic functions and routing even after the FPGA has been deployed [46]. This is typically achieved through underlying programming technologies such as SRAM-based, Flash-based, or anti-fuse technology, each with its trade-offs in terms of area, speed, re-programmability, and volatility.

2.4.1.5 Programming Technologies

FPGA programmability is typically achieved using one of the following technologies [46]:

- **SRAM-based Programming:** Uses static memory cells (SRAM) to configure the routing and logic blocks. SRAM-based FPGAs are reprogrammable and leverage standard CMOS technology, making them highly flexible but consuming more area and power.
- **Flash-based Programming:** Flash memory is non-volatile, meaning the configuration is retained without power. Flash-based FPGAs are more area-efficient than SRAM-based FPGAs but have a limited number of reprogrammable cycles.
- **Anti-fuse-based Programming:** Anti-fuse technology provides the smallest area overhead and is non-volatile. However, anti-fuse FPGAs can only be programmed once, as the anti-fuse mechanism is not reversible.

2.5 Parallel Computing Platforms

In this section, we'll explore the role of parallel computing platforms in enhancing financial modeling's efficiency and precision. We'll start by understanding the significance of parallel computing in the finance sector. Following this, we'll examine OpenCL, discussing its origins, architectural nuances, and its particular relevance to our research. Lastly, we'll discuss OneAPI, shedding light on its distinct advantages and its applicability in option pricing.

2.5.1 Introduction to the importance of parallel computing in financial modeling

Financial modeling stands at the crossroads of complexity and precision. Models often involve many variables, parameters, and scenarios, each of which can profoundly influence financial outcomes. As these models grow in complication, the computational demands for their timely and accurate evaluation escalate exponentially. This is where parallel computing enters the picture.

Parallel computing refers to the practice of executing multiple tasks or pro-

cesses simultaneously, leveraging the nature of modern processors. Instead of processing sequences one after the other, as in traditional serial computing, parallel computing breaks down tasks into smaller sub-tasks that are computed concurrently. This capability is a game-changer for the world of finance for several reasons. Parallel computing plays a crucial role in financial markets by facilitating faster data processing, enabling analysts and traders to make informed decisions swiftly and avoiding delays that could lead to missed opportunities or significant losses. Additionally, financial modeling often employs Monte Carlo simulations, which require running thousands or even millions of scenarios to determine probable outcomes. Parallel computing processes these scenarios concurrently, drastically reducing the time required for such comprehensive simulations.

With the advent of big data in finance, where vast volumes of data are analyzed for insights, parallel processing ensures efficient handling of this data, extracting relevant information without unnecessary delays. In risk management, parallel computing allows for the simultaneous evaluation of multiple variables and their interplay, providing a more holistic view of potential risks and improving the accuracy of risk assessments.

In essence, parallel computing has revolutionized financial modeling, offering the dual benefits of speed and comprehensive analysis. As financial markets continue to evolve in complexity and speed, the reliance on parallel processing methodologies will only grow, underscoring its critical role in modern financial decision-making.

2.5.2 OpenCL: Origins, Architecture, and Its Significance in the Context of this Research

OpenCL (Open Computing Language) is a framework developed to enable parallel programming across heterogeneous platforms. Introduced by Apple Inc. in collaboration with other parties and managed by the Khronos Group, OpenCL provides a unified interface for programming diverse hardware platforms such as CPUs, GPUs, DSPs, and more.

2.5.2.1 Origins

Initially conceived by Apple in 2008, OpenCL's development aimed to harness the power of GPUs beyond just graphical renderings. Recognizing GPUs' com-

putational potential, Apple, in collaboration with other tech giants, aimed to devise a standard that could be universally adopted, leading to the inception of OpenCL. It quickly garnered attention and underwent rapid enhancements, leading to its current widespread adoption.

2.5.2.2 Architecture

OpenCL's architecture is built around a few core concepts:

- **Platform**: Represents the OpenCL installation on a host.
- **Device**: A computational resource where kernels (functions) execute. This could be a GPU, CPU, or other types of processors.
- **Context**: The environment within which the devices and memory objects are managed.
- **Command Queue**: A sequence of commands that will be executed on a specific device.
- **Memory Objects**: Represent memory spaces like buffers or images.
- **Kernel**: The core computational functions written in the OpenCL C language.

Here is a basic OpenCL code sample to illustrate its structure:

```
__kernel void simple_add(__global const int *A,
                         __global const int *B,
                         __global int *C) {
    int id = get_global_id(0);
    C[id] = A[id] + B[id];
}
```

Listing 2.1: Simple OpenCL kernel for addition

2.5.2.3 Significance in the Context of Financial Research

For financial computations, especially in the domain of option pricing, the need for speed, precision, and scalability is important. OpenCL's ability to harness the combined computational power of CPUs and GPUs means that complex financial models, which would traditionally take substantial time to

compute on a single type of processor, can now be evaluated more rapidly. This acceleration can be particularly crucial for real-time market analytics and simulations. Moreover, its cross-platform nature ensures that the developed solutions are versatile and can be deployed on various hardware setups, aiding in scalability and flexibility of the research solutions.

In the subsequent sections, we'll look into specific use cases and implementations that leverage OpenCL in the ecosystem of option pricing and financial modeling, showcasing its capabilities and advantages in practical scenarios.

2.5.3 OneAPI: Introduction, its advantages, and relevance for option pricing applications

OneAPI is a unified programming model designed to simplify the development of applications that run across a variety of hardware architectures, including CPUs, GPUs, FPGAs, and more. It aims to provide a single, cohesive interface for harnessing the capabilities of various hardware platforms, eliminating the need for multiple, vendor-specific languages and tools.

2.5.3.1 Advantages of OneAPI

OneAPI offers several advantages, including a unified codebase that allows developers to maintain a single codebase, eliminating the need for separate codes tailored for different architectures. This promotes flexibility and portability, facilitating seamless integration across various hardware platforms. Additionally, OneAPI is optimized for hardware, enabling it to tap into the potential of modern architectures and deliver high-performance outputs. By reducing development time and resources, OneAPI also leads to more cost-efficient project lifecycles.

2.5.3.2 Relevance for Option Pricing Applications

Financial computations, especially those related to option pricing, require intense calculations, simulations, and evaluations. Given the stochastic nature of financial markets and the complexities in the models used, there's an ever-pressing need for faster and more efficient computation. OneAPI, with its ability to tap into a range of hardware resources, provides an invaluable tool for accelerating these calculations.

For instance, consider a simple Black-Scholes PDE option pricing model. Leveraging OneAPI, one can offload parts of the calculation to different hardware resources to speed up the computation.

```
#include <oneapi/dpl/algorithm>
#include <oneapi/dpl/execution>
// ... Other necessary includes ...

double BlackScholesOnePDEAPI(double S,
                             double K,
                             double T,
                             double r,
                             double v) {
    // Implementation leveraging OneAPI's
    // parallelism features...
    auto R = exp(-r * T);
    // ... Rest of the Black-Scholes PDE formula ...

    return price;
}
```

Listing 2.2: OneAPI Code Sample for Option Pricing

In this snippet, the OneAPI's Data Parallel C++ (DPC++) capabilities can be employed to parallelize specific parts of the Black-Scholes PDE algorithm (specifically the tridiagonal system produced in each time step of the method [47]), potentially offering significant speedups.

In conclusion, OneAPI provides a robust platform for addressing the multilayered and computation-intensive needs of option pricing models. Its unified nature, combined with the capability to optimize across diverse hardware platforms, makes it a fitting choice for modern financial computations.

2.6 Conclusion

In Chapter 2, we embarked on a comprehensive journey through the foundational underpinnings of option pricing, starting from its historical evolution to

the modern techniques and technologies that power its computations. A deep dive into the topic of hardware acceleration illuminated the important role of Field Programmable Gate Arrays (FPGAs) in computational finance. These devices, coupled with the prowess of parallel computing platforms like OpenCL and OneAPI, present a potent blend of versatility, speed, and efficiency. Such tools and techniques are especially critical when grappling with complex financial models that demand both precision and expediency.

The exploration of OneAPI and OpenCL, in particular, underlines a broader trend in the financial world – the continuous strive for optimization. As markets evolve and financial instruments grow in complexity, the computational methods and hardware supporting them must also advance. OneAPI, with its unified approach to harnessing the capabilities of various hardware platforms, exemplifies this evolution. Similarly, the increasing reliance on FPGAs, given their flexibility and performance benefits, underscores the industry's shift towards specialized computational solutions.

As we transition to subsequent chapters, it's crucial to consider the connection between the theoretical constructs of option pricing and the practical tools that bring these theories to life. The next chapter, "State-of-the-art", will look deeper into current advancements and methodologies, building upon the foundational knowledge laid out in this chapter.

Chapter 3

State-of-the-art in FPGA Acceleration of Option Pricing

3.1 Introduction

Having established a foundational understanding of Field Programmable Gate Arrays (FPGAs) and option pricing in the preceding chapters, this chapter aims to systematically review the intersection of these two domains. By synthesizing the existing literature, we seek to uncover the depth and breadth of research on the application of FPGAs in option pricing. This systematic review will not only provide a comprehensive overview of current methodologies and findings but also identify gaps and opportunities for further exploration, setting the stage for our deep dive into Pascal's Simplex-Based Multinomial Option Pricing.

3.2 Methodology

To conduct the study, we followed the guidelines on literature reviews provided in [48, 49, 50]. We used the search terms + FPGA + Option pricing + Hardware acceleration + Financial simulations + Monte Carlo simulations + Black–Scholes model + Heston model + High-frequency trading + Financial engineering + FPGA-based option priced algorithms + FPGA optimization + FPGA frameworks + FPGA synthesis + FPGA parallelism + FPGA energy

efficiency + FPGA design + Derivatives pricing to search in the following digital libraries: IEEE Xplore; ACM Digital Library; SpringerLink; ScienceDirect; Google Scholar; and ProQuest. We gathered 131 papers.

In order to decide which of them to further analyze, we performed three exclusion stages. Firstly, we filtered the papers based on titles, then on abstracts. In reading the abstracts we excluded papers regarding non-engineering aspects (e.g., papers addressing purely economic aspects of option pricing). Finally, we were left with 99 papers distributed across the years illustrated in Figure 3.1. If we consider the papers published in recent years, i.e., 2021 onwards, we see modern systems such as as the Intel Stratix V FPGA (Intel Terasic Stratix-V GX FPGA, Intel Corporation, Santa Clara, California) [51], Intel Stratix 10 GX (Intel Corporation, Santa Clara, California) [52], NVIDIA RTX A5000 (Nvidia Corporation, Santa Clara, California) [53], Intel Xeon Platinum 8260L (Intel Corporation, Santa Clara, California) (Cascade Lake) [54] are used as offloading and acceleration hardware. Furthermore, modern frameworks such as SYCL (Khronos Group SYCL 2020 Specification, Oregon, United States)) [54], OpenCL (Khronos Group OpenCL 1.0 Specification, Oregon, United States) [55], Intel OneAPI (version 21.2, Intel Corporation, Santa Clara, California) [52] and CUDA (Nvidia CUDA capability 1.0, Nvidia Corporation, Santa Clara, California) [56] are used in many papers to enable rapid development of option pricing solutions.

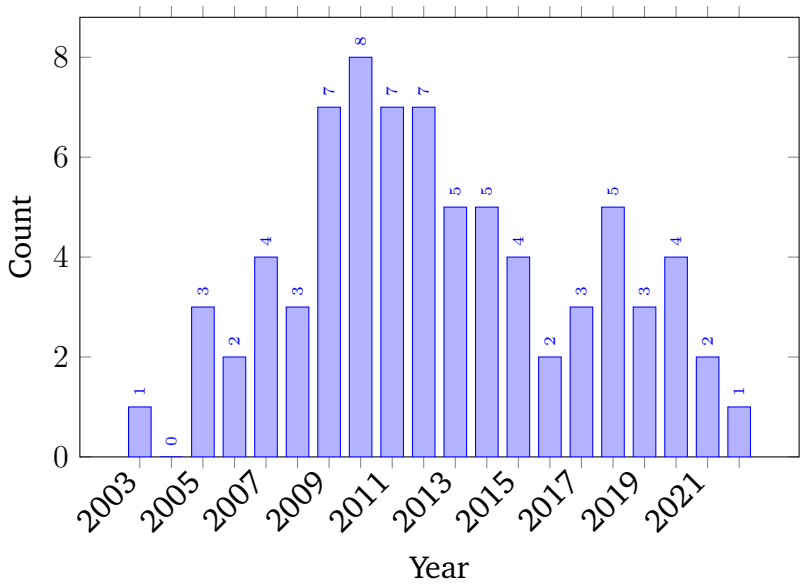


Figure 3.1: Bar chart of yearly counts of publications.

3.2.1 Data Extraction

From reading the 99 papers, we were able create 7 categories.

- Black–Scholes Model
- Binomial and Trinomial Tree Methods
- Monte Carlo Simulations
- Finite Difference
- Heston Model
- Quadrature Methods
- Miscellaneous

3.3 Comparison of FPGA, CPU, and GPU

It is useful to provide a discussion highlighting the differences between the main hardware accelerator types. The three main types of hardware for implementing algorithms are CPUs, GPUs, and FPGAs, each with distinct advantages and disadvantages [57, 58]. CPUs, the most common processors, are designed for general-purpose tasks and excel at executing sequential instructions quickly. However, they are less suited for parallel processing tasks like those in machine learning and image processing.

GPUs, with their numerous parallel computing cores, are ideal for tasks such as gaming, 3D rendering, and machine learning, thanks to their high memory bandwidth. Despite their performance, they lack support for various software libraries and have limited instruction sets, making them less suitable for general-purpose computing.

FPGAs can be programmed with custom logic circuits to perform specific functions efficiently and typically consume less power than CPUs and GPUs, making them perfect for embedded applications. However, they require specialized skills and tools for programming, which makes them challenging for general-purpose computing. Table 3.1 provides a detailed comparison of FPGA, CPU, and GPU across various noteworthy aspects such as power consumption, computing power, flexibility, latency, programming complexity, development time, cost, and parallelism [51, 59]. In summary, CPUs are great for sequential

tasks, GPUs excel in parallel processing, and FPGAs offer customizable, power-efficient solutions for specific applications.

Table 3.1: Comparison between FPGA, CPU, and GPU across various aspects.

Aspect	**FPGA**	**CPU**	**GPU**
Consumption	Low to Moderate	Moderate to High	High
Computing Power	High (specific tasks)	Moderate (general)	Very High (parallel tasks)
Flexibility	High (custom hardware)	Very High (general)	Moderate (parallel optimized)
Latency	Very Low	Moderate	Low to Moderate
Programming	Complex (hardware knowledge)	Easy (high-level languages)	Moderate (requires parallelism)
Development Time	Long (custom design)	Short (established tools)	Moderate (specialized tools)
Cost	High (initial, low volume)	Low to Moderate	Moderate to High
Parallelism	High (custom pipelines)	Low to Moderate	Very High
Use Case	Specialized, real-time	General-purpose	Parallel processing, AI/ML

3.4 Review of FPGA Implementations for Option Pricing

3.4.1 Black–Scholes Model

The Black–Scholes Model, introduced by Fischer Black, Myron Scholes [17], and later expanded by Robert Merton [60], is a fundamental mathematical model for pricing European-style options. It assumes that the price of the underlying asset follows a geometric Brownian motion with constant drift and volatility, no transaction costs or taxes, and continuous trading. The model provides a formula for calculating the price of European call and put options

based on variables such as the current price of the underlying asset, strike price, risk-free interest rate, time to expiration, and asset volatility. Despite its simplifying assumptions, the Black–Scholes Model is widely used for its robustness and forms the basis for modern financial theory and risk management strategies. It has significantly impacted the growth of options markets and financial engineering.

3.4.1.1 Black–Scholes Model on FPGA

The Black–Scholes algorithm has been deployed onto a Xilinx Virtex-II Pro architecture, partitioning the model across a processor and an FPGA, which outperformed the Mathematica software by completing computations in approximately 1% of the time [61] giving a 3400 times speedup over the ARM processor implementation and 312 times speedup over the Mathematica implementation. In [62], the authors introduced an FPGA-based pricing model for complex options, such as 'Asian' options, using a parameterized Monte Carlo version of the Black–Scholes model. Their FPGA solution demonstrated a speedup of 270 times for a single FPGA, compared to a Xeon blade CPU and 5400 times speedup for 16 FPGAs compared to a Xeon blade CPU.

Another study presented a hardware architecture for generating random vectors applied to Delta-Gamma Value-at-Risk simulations and Black–Scholes option pricing models. This FPGA solution's scalability was emphasized, showing a generation rate 200 times that of a single Opteron 2.2 GHz [63]. Indeed, in the previous example it is noted that a single FPGA can replace the equivalent of 33 quad-core computers (132 CPU cores) in a cluster, thereby significantly reducing the heat, power, space, and capital outlay required for a 32U rack to just one computer with an FPGA. The authors of [64] explored computational systems for enhancing investment decision speed and accuracy, introducing a novel FPGA cluster approach called SMILE. The SMILE architecture, with 32 nodes, is approximately five times faster than a traditional computing cluster with the same number of nodes. To achieve similar response times to the 32-node SMILE cluster, the traditional computing cluster (ALTAMIRA) needs to scale up to 256 processors.

An OpenCL-based kernel was proposed for European Stock Option pricing using the Monte Carlo Black–Scholes algorithm on an FPGA device. This FPGA-based approach significantly outperformed CPU and GPU architectures, achieving computational times in milliseconds [65] (exact times were not re-

ported). Ref. [66] presented a high-performance Black–Scholes system for pricing European call options on an Altera Stratix-V FPGA, emphasizing the FPGA's advantages of dedicated hardware and efficient complex computations. Exact speedup comparisons were not possible for this paper.

The industrial readiness of High-Level Synthesis (HLS) tools for FPGAs, particularly for pricing Black–Scholes and Heston model-based options, has been illustrated by showing how HLS tools outperformed traditional CPU, GPU, and Xeon Phi implementations, endorsing their viability in business scenarios [67]. The authors present up to a 221-times speedup compared to a sequential CPU implementation. Another project implemented a structured product pricer on an FPGA, achieving speed-ups of 550 to 1450 times compared to a one-core software solution [68].

The authors of [69] introduced a design flow for efficient hardware accelerators for option pricing on an FPGA platform, outperforming most manually-designed engines and software implementations by a factor of 2 times. Another methodology for option pricing tasks on data center FPGAs, emphasizing power efficiency improvements (30% more floating point operations per Joule of energy) in computational finance applications was, presented in [70]. Finally, a thesis [71] focused on FPGA-based accelerators for computational finance models, with FPGAs achieving 4 times to 5 times more operations per Watt than a GPU.

An implementation of the Black-Scholes formula on a Pynq-Z1 FPGA is presented in [72], but the results indicated inefficiencies (slowdown) in the FPGA solution compared to a Python execution on the ARM. An emphasis on the high potential of specialized architectures for accelerating financial product pricing on commercially available FPGAs is presented in [73], which provided a speedup of between 550 times to 1450 times. The research in [74] introduced an efficient hardware structure for the Black-Scholes option pricing model, outpacing CPU and GPU implementations in terms of throughput. This research shows a 365 times speedup over similar CPU implementations and 2.6 times speedup over GPU implementations with comparable chip manufacturing processes. They also report a throughput-power efficiency factor of 3293 times compared to a CPU and 59.4 times compared to a GPU.

The paper [75] introduced a parallelized calculation framework for computing implied volatility based on the Black-Scholes Model, with the FPGA implementation showcasing at least 4 to 5 times speedup over traditional methods.

Lastly, [56] discussed the implementation of the Black-Scholes-Merton options pricing formula on various hardware platforms (presenting an estimated 298 times speedup for Black-Scholes), highlighting that both GPU and FPGA performances were primarily limited by the interconnect.

3.4.1.2 Key Observations

Studies such as those by [61, 62, 65] have showcased the remarkable speed and efficiency gains achieved by FPGA-based systems, often outperforming their counterparts by orders of magnitude.

Another observation is the increasing prominence of High-Level Synthesis (HLS) tools in the area of FPGA-based financial computations. Tools like HLS allow for a more seamless transition from high-level programming languages to hardware descriptions, simplifying the design process and making FPGA platforms more accessible to developers without deep hardware expertise. [67] and others have demonstrated the industrial readiness of these tools, emphasizing their potential to outperform traditional computational platforms. However, it is also worth noting that not all FPGA implementations guarantee superior performance. As highlighted by the study from [72], inefficiencies, especially those related to communication times with the FPGA, can sometimes lead to sub-optimal results, underscoring the importance of meticulous design and optimization.

Lastly, the choice of the FPGA platform and the specific optimization techniques employed play an important role in determining performance outcomes. As seen in the works of [71, 74], the selection of the FPGA platform, whether datacenter-class or embedded-class, and the integration of specific algorithms and structures can significantly influence the computational speed and efficiency.

As evidenced by studies like [71, 74, 75], there is a growing emphasis on harnessing the full potential of FPGA platforms, whether they are datacenter-class or embedded-class. The integration of High-Level Synthesis (HLS) tools and platforms such as OpenCL, as highlighted by [67, 70], underscores the industry's move towards making FPGA development more accessible and efficient. Furthermore, the consistent pursuit of power efficiency, as seen in [70], indicates a broader trend towards sustainable and energy-efficient computational finance solutions. Table 3.2 shows a consolidated summary of notable findings.

Table 3.2: Summary of notable findings from various studies on FPGA implementations of the Black-Scholes model.

Study	**Notable Findings**
Tandon (2003) [61]	Implemented Black–Scholes on Xilinx Virtex-II Pro, achieving a 3400 times speedup over ARM processor and 312 times over Mathematica.
Baxter et al. (2007) [62]	FPGA-based pricing model for 'Asian' options, showing a 270 times speedup for a single FPGA and 5400 times for 16 FPGAs compared to a Xeon blade CPU.
Thomas et al. (2007) [63]	Hardware architecture for generating random vectors for Delta-Gamma Value-at-Risk and Black–Scholes, achieving 200 times the generation rate of a single Opteron 2.2 GHz CPU. Equivalent to 33 quad-core CPUs.
Bruce (2008) [56]	Implemented Black–Scholes–Merton options pricing on various hardware platforms, estimating a 298 times speedup for Black–Scholes. Highlighted interconnect limitations for both GPU and FPGA.
Castillo et al. (2009) [64]	SMILE architecture with 32 nodes, 5 times faster than a traditional computing cluster with the same number of nodes. Traditional cluster needs 256 processors to match SMILE's response times.
Patel et al. (2016) [65]	OpenCL-based kernel for European Stock Option pricing on FPGA, significantly outperforming CPU and GPU architectures, achieving computational times in milliseconds.

Cont.

Study	**Notable Findings**
Choo et al. (2013) [66]	High-performance Black–Scholes system on Altera Stratix-V FPGA for pricing European call options.
Inggs (2015) [67]	Demonstrated HLS tools' industrial readiness for Black–Scholes and Heston models, achieving up to 221 times speedup over sequential CPU implementation.
Guerrero et al. (2015) [68]	Structured product pricer on FPGA, achieving speed-ups of 550 to 1450 times compared to a one-core software solution.
Pham et al. (2016) [69]	Design flow for efficient hardware accelerators for option pricing on FPGA, outperforming most manually-designed engines and software implementations by 2×.
Inggs (2016) [70]	Emphasized power efficiency improvements (30% more floating point operations per Joule of energy) in computational finance on FPGA.
Ma (2019) [71]	FPGA-based accelerators for computational finance, achieving 4 times to 5 times more operations per Watt than GPU.
Rodrigues et al. (2019) [73]	Specialized architectures for financial product pricing on FPGA, providing a speedup of 550 times to 1450 times.
de Castro (2019) [72]	results indicated inefficiencies (slowdown) in the FPGA solution compared to a Python execution on the ARM.

Cont.

Study	Notable Findings
Li et al. (2021) [74]	Efficient hardware structure for Black–Scholes on FPGA, showing a 365 times speedup over CPU implementations and 2.6 times over GPU. Throughput-power efficiency speedup of 3293 times compared to CPU and 59.4 times compared to GPU.
Wang et al. (2022) [75]	Parallelized framework for computing implied volatility on FPGA, showing 4 times to 5 times speedup over traditional methods.

3.4.2 Binomial and Trinomial Tree Methods

The Binomial and Trinomial Tree Methods are numerical techniques used to price options by discretizing the continuous price movements of the underlying asset. Developed by John Cox, Stephen Ross, and Mark Rubinstein in 1979 [18], the binomial method models the asset price movements over discrete time intervals, allowing for two possible outcomes (up or down) at each step. The trinomial method extends this by introducing a third possible outcome (no change), improving accuracy for more complex options. These methods are particularly useful for American options, which can be exercised at any time before expiration. By constructing a recombining tree of possible prices and working backward from the option's expiration, these methods determine the option's fair value at each node.

3.4.2.1 Binomial and Trinomial Tree Methods on FPGA

The implementation of binomial option pricing models on FPGAs, particularly for American options, has been a topic of interest in recent years. One such study utilized OpenCL to achieve this, aiming to reduce energy consumption while maintaining high parallelism. The results revealed that the optimal FPGA implementation could evaluate over 2000 options/s with an average power consumption of less than 20 W [76].

A parallel pipelined architecture designed for accelerating financial computations using binomial-tree pricing models was introduced, which was specifically tailored to handle concurrent requests for option valuations. When implemented on an xc4vsx55 FPGA, this architecture was found to be over 250 times faster than a Core2 Duo processor operating at 2.2 GHz. Moreover, it outperformed an Nvidia Geforce 7900GTX processor with 24 pipelines running at 650 MHz by more than twice the speed, suggesting a promising avenue for future optimizations in performance, area, and power [77].

Another innovative approach introduced a scalable, reconfigurable hardware architecture for pricing American options using the Binomial Lattice algorithm. This method provided double precision floating point pricing and could evaluate up to N = 64,000 time steps. Notably, it achieved a 73 times speedup over an optimized CPU implementation, emphasizing its efficiency on a per-asset basis [78].

Further research explored two parallel pipelined architectures on an FPGA for accelerating option pricing models using both binomial and trinomial trees. These architectures were capable of performing over 100 times faster than a Core2 Duo processor and twice as fast as a non-CUDA Nvidia GPU. The study also reported on power consumption where their solution which operates at 160 MHz consumes 5.7 Watts, making it approximately 6-times more efficient than the Geforce 8600GTS, which requires 71 Watts, in terms of calculation speed per watt [79].

Beyond binomial and trinomial trees, there has been work on FPGA-based parallel processors optimized for solving the Black–Scholes partial differential equation, a staple in financial option pricing. One such processor achieved a 5 times speedup compared to a 2 GHz dual-core Intel CPU, showcasing its flexibility and scalability for various applications [47].

In the area of Monte Carlo option pricing, a novel FPGA-based method was proposed that utilized a discrete-space random walk over a binomial lattice. This method was hypothesized to enhance parallelization and performance without compromising accuracy. Experimental results confirmed this, showing a 50 times improvement in throughput compared to existing FPGA methods [80].

The energy efficiency of FPGAs in financial computations, especially in the binomial option pricing model for American options, was explored using Altera's

OpenCL implementation. The results from this study were promising, specifically considering energy efficiency, the FPGA implementation is more than 5 times more energy efficient than the software implementation. [76].

A systolic hardware architecture designed for pricing American options using the binomial-tree model on FPGAs demonstrated a linear increase in latency with the size of the binomial tree. This architecture outperformed previous works by achieving a 65 times improvement in option latency [81].

Another study introduced a formal mathematical framework for the binomial-tree model, which facilitated FPGA-based solutions for various binomial-tree problems. The reported performance ranges from 1.4 times throughput compared with a hand-tuned systolic design, and up to to 9.1 times and 5.6 times improvement when compared to scalar and vector architectures, respectively [82].

Comparing the performance and energy efficiency of OpenCL-accelerated computations across different platforms, one study found that while GPUs might offer superior throughput, FPGAs displayed better performance portability and energy efficiency in some cases. More specifically, the authors report a 68% device peak performance for the FPGA compared to 20% for an NVIDIA GPU. Also, they report up to a 1.4 times improvement of energy efficiency for some cases. [83].

The potential of FPGA technology in multi-asset option pricing was also explored. One study presented an architecture for a two-asset European option pricer based on Pascal's simplex, which demonstrated the efficiency of hardware acceleration for multi-asset option pricing [55] requiring an estimated 39.3 Watts. This study leverages pipelining to reduce latency of computing option pricing by a factor of 25. Another study introduced a multi-asset option pricer using Pascal's simplex and a recombining multinomial tree approach, achieving computation speeds up to 43 times faster than a software-based general-purpose processor [52].

3.4.2.2 Key Observations

The binomial and trinomial tree methods, when implemented on FPGAs, consistently outperform traditional CPU-based methods in terms of speed and efficiency [77, 78, 79]. The introduction of parallel pipelined architectures, systolic hardware designs, and formal mathematical frameworks showcases the

ongoing innovation in this field [77, 81, 82]. Furthermore, the exploration into multi-asset option pricing using FPGA technology underscores the expanding horizons of its applications, with studies achieving remarkable computation speeds and efficiencies [55, 52].

As computational demands in the financial sector grow, there is an evident gravitation towards FPGA-based solutions due to their inherent parallelism and energy efficiency [76]. The consistent exploration and development of parallel architectures, from pipelined to systolic designs, highlight the industry's commitment to optimizing performance and throughput [77, 81]. Additionally, the increasing integration of OpenCL frameworks with FPGA implementations signifies a move towards more accessible and adaptable hardware programming paradigms, allowing for broader applications and easier adaptability across various financial models [83, 52]. Table 3.3 summarizes the notable findings from this section.

Table 3.3: Summary of notable findings from various studies on FPGA implementations of binomial and trinomial tree methods.

Study	**Notable Findings**
Morales et al. (2014) [76]	FPGA implementation evaluated over 2000 options/s with <20 W power consumption.
Jin et al. (2008) [77]	250 times faster than Core2 Duo, 2 times faster than Nvidia Geforce 7900GTX.
Wynnyk and Magdon-Ismail (2009) [78]	73 times speedup over optimized CPU implementation.
Jin et al. (2009) [79]	100 times faster than Core2 Duo, 2 times faster than non-CUDA Nvidia GPU, 6 times more efficient in power consumption.
Chatziparaskevas et al. (2012) [47]	5 times speedup compared to 2 GHz dual-core Intel CPU.
Tavakkoli et al. (2014) [81]	65 times improvement in option latency
Fabry et al. (2017) [80]	50 times improvement in throughput for Monte Carlo option pricing.
Morales et al. (2014) [76]	5 times more energy efficient than software implementation.
Tavakkoli et al. (2017) [82]	1.4 times throughput vs. hand-tuned systolic design, up to 9.1 times and 5.6 times improvement vs. scalar and vector architectures.
Minhas et al. (2018) [83]	68% device peak performance for FPGA vs. 20% for NVIDIA GPU, up to 1.4 times energy efficiency improvement.

Cont.

Study	Notable Findings
Mahony et al. (2020) [55]	Two-asset European option pricer, 25 times latency reduction, 39.3 W power consumption.
Mahony et al. (2022) [52]	Multi-asset option pricer, 43 times faster than software-based general-purpose processor.

3.4.3 Monte Carlo Simulations

Monte Carlo Simulations are numerical techniques used in financial modeling to estimate the value of options and other derivatives by simulating the random paths of the underlying asset's price [19]. These simulations rely on repeated random sampling to compute their results, making them particularly useful for pricing complex options where analytical solutions are difficult or impossible to obtain. The method involves generating numerous potential future price paths for the underlying asset using stochastic processes, then calculating the payoff for each path and averaging the results to obtain an estimated option price. Monte Carlo simulations are highly flexible and can handle various payoff structures and market conditions, including American options[1] and path-dependent options. Their accuracy increases with the number of simulations, though they can be computationally intensive.

3.4.3.1 Monte Carlo Simulations on FPGA

The advancements in Field-Programmable Gate Arrays (FPGAs) have significantly revolutionized the acceleration of Monte Carlo simulations, especially in the domain of financial applications.

In the early stages of this evolution, [85] made a significant contribution by introducing a hardware accelerator for Monte Carlo simulations on FPGAs. This innovation achieved a remarkable 50-fold speedup for financial applications using the BGM interest rate model. This pioneering work paved the way for further exploration, as evidenced by [86], which demonstrated a 12-fold

[1] It should be noted that Monte Carlo for American options is not straight forward. Typically, a specialised approach is needed such as the Longstaff-Schwartz Least Squares (L3MC) [84]

speedup. This exploration was deepened by [87], which presented a modular framework and achieved speedups between 8 and 71 times for two discrete-time random walks. [88] introduced the 'HyperStreams' abstraction, achieving a 146 times acceleration for European option pricing. The Monte-Carlo simulation engine's implementation on the FPGA-based Maxwell supercomputer further emphasized this potential. This engine, as discussed in [89, 90], outperformed software solutions by factors ranging from 340 to 750 times. Additionally, [91] documented a 20 times speedup by implementing the LSMC method for American option pricing on an FPGA with considerable energy savings (>20:1). The versatility of FPGAs in financial computations was further showcased by [92], which introduced an FPGA accelerator for QMC derivative pricing, surpassing the performance of a 3 GHz processor by over 50 times.

Describing an FPGA hardware architecture, [93] focuses on expediting the Least-Squares Monte Carlo (LSMC) method for American option pricing. The FPGA-based solution achieved remarkable speed-ups: 25 times in path generation and 18 times in regression phases compared to its software counterpart. This resulted in an overall speed-up of 20 times and an energy efficiency of 54 times. Although FPGA design takes longer than software development, leveraging existing IP cores and the evolution of reconfigurable hardware present promising avenues for diverse algorithms. The authors also planned to examine heterogeneous computing platforms and fine-tune arithmetic types and word lengths for enhanced precision and efficiency.

Several studies have highlighted the adaptability and efficiency of FPGAs. For example, [94] introduced Contessa, a high-level language tailored for Monte Carlo simulations in financial computing. This language facilitated a direct route to FPGA-accelerated performance, showcasing significant speedup (maximum 62 times) and reduced power consumption (a tenth of the power consumption, reported by the authors, when compared to a CPU implementation. Precise figures were not available). [95] presented a dynamic scheduling Monte Carlo simulation framework designed for multi-accelerator heterogeneous clusters, achieving a 44-fold speedup. They also report that their cluster, with 8 Virtex-5 xc5vlx330t FPGAs and 8 Tesla C1060 GPUs, provides 19.6 times improved energy efficiency compared to a cluster with 16 AMD Phenom 9650 quad-core 2.4 GHz CPUs for the asset simulation application. The exploration of fixed-point arithmetic for Monte Carlo-based European option pricing on FPGAs by [96] revealed that the error introduced was negligible

(comparing 26-bit fixed point implementation to 30-bit implementation), thus enhancing throughput without compromising accuracy.

The energy efficiency and performance superiority of FPGAs were further highlighted by various studies. [97] presented the first FPGA-based accelerator for option pricing using the Heston model, emphasizing the energy efficiency and performance superiority of FPGAs for advanced Monte Carlo simulations and saving 89% of the energy and providing around twice the speed. [98] looked into the potential of FPGAs as hardware accelerators for European option pricing using Monte Carlo simulations, showcasing substantial speed-up and energy savings. For a Monte Carlo simulation with n = 100,000, the FPGA-based ASP-4P configuration consumes only 171 μJ, compared to 631,878 μJ for a CPU-based system, demonstrating over 3686-times more energy efficiency.

Recent advancements have further solidified the importance of FPGAs in this domain. Ref. [99] introduced CloudiFi, a cloud-native framework that significantly improved end-to-end response time and scalability for Monte-Carlo European Option Pricing workloads. Their early results indicate up to 485 times gains in microservice response time. [100] showcased a design that was 690-times faster than a software implementation when concurrently mapping and customizing computationally-intensive tasks onto an FPGA-based computing cluster.

Other significant contributions include [101]'s high-performance design for interest rate derivative pricing (58 times speed-up compared to software implementation on CPU and more than two orders of magnitude more energy efficient), and [102]'s massively parallelized Quasi-Monte Carlo simulation engine (2 times to 3 times speed-up). They also report that their power consumption measurements show FPGAs to be 336 times more energy efficient than CPUs, and 16 time more energy efficient than GPUs. [103] provided a comparative study between FPGA and GPU systems, highlighting FPGA's superior performance in specific applications. They reported, for power consumption, 4 times greater efficiency compared to an Intel Xeon 5138 dual-core processor. [104] proposed a mixed precision methodology resulting in a 170 times faster than quad-core implementations on software and also demonstrated energy efficiency improvements by being up to 170 times more energy-efficient compared to quad-core CPU implementations and up to 5.5 times more energy-efficient compared to NVIDIA Tesla C2070 GPU implementations

for various Monte Carlo simulation applications. Additionally, [105] introduced a multi-level Monte Carlo FPGA architecture where they showed a reduction of computational complexity, compared to single-level methods, of up to 50% and, also, that their design can simulate up to 100 million time steps in an asset path simulation with less than 3.6 W. [106, 107] both emphasized the capabilities of reconfigurable hardware in financial computations, showing various speed-ups ranging from 28 times to 149 times with an 18.6 energy efficiency improvement over a comparable CPU in [106]. [108, 109] showcased significant speed advantages in their respective studies compared to software (800 times and 4200 times, respectively) and the power consumption of the FPGA design is 2.8 times lower than the software (a 2.5 GHz Xeon E5420) and 6.9 times lower than the GPU design (1.3 GHz Tesla C1060 GPU). [110] introduced a novel methodology for MLMC simulations showing a 3–9 times speed-up on the same platform, and [111] highlighted the benefits of a high-level synthesis approach showing a speed-up of 3.89 times for a Black–Scholes implementation. [112] introduced a hardware framework for dynamic loading showing a speed-up of between 30 times and 120 times with up to 30-times greater energy efficiency compared to software-only execution on CPUs. [113] investigated FPGA-based accelerators using high-level synthesis and showed a 2 times speedup compared to a K4200 GPU for a Black–Scholes implementation and a 2 times speedup for an Asian option calculation compared to a GTX960 GPU. They also demonstrated significant power efficiency, consuming only 3 watts compared to approximately 100 watts for a GPU. [114] presented a high-level FPGA design toolflow, and [115] proposed *HyPER*, a modular hybrid system which is 3.4 times faster and 36 times more power efficient than a CPU implementation for various option pricing calculations. Finally, in [116], the Monte Carlo simulation for option pricing using a Xilinx Virtex-5 achieved a 46.2 times speed-up and a 14.4 times improvement in energy efficiency compared to a CPU, processing 256 million elements in 3.73 seconds while consuming 70.31 Joules.

3.4.3.2 Key Observations

The evolution of Field-Programmable Gate Arrays (FPGAs) has been instrumental in advancing Monte Carlo simulations, particularly within the financial sector. Early research, such as that of [85], laid the foundation by introducing hardware accelerators that achieved significant speedups. This trajectory was furthered by studies like [86, 87], which explored the potential of FPGAs in

various financial applications.

Option pricing emerged as a prominent area of focus, with works like [88, 89] showcasing the capabilities of FPGAs in achieving remarkable accelerations. The versatility of FPGAs was further highlighted by innovations such as the Contessa language [94] and the dynamic scheduling framework presented in [95].

Recent advancements, such as the CloudiFi framework [99], emphasize the growing importance of FPGAs in modern cloud-based computational finance. Indeed, the exploration of fixed-point arithmetic for Monte Carlo simulations of [96] and the energy efficiency studies of [97] underscore the adaptability and efficiency of FPGAs.

In conclusion, the collective contributions from various researchers have solidified the role of FPGAs in accelerating Monte Carlo simulations, with each study adding a unique perspective to this evolving domain.

Following the key observations, several distinct trends emerge in the utilization of FPGAs for Monte Carlo simulations and financial computations. The initial stages of research, as seen in studies like [85], were primarily centered around exploiting the computational power of FPGAs. This phase was marked by direct hardware accelerations, aiming for raw speedups.

As the research evolved, there was a clear gravitation towards domain-specific applications, with option pricing emerging as a focal point. This specialization, highlighted by works such as [88, 90], went beyond mere computational advantages, introducing innovative methodologies and abstractions tailored for financial computations.

Another discernible trend has been the emphasis on sustainability and adaptability. With the global push towards green computing, studies like [97, 98] have underscored the energy efficiency of FPGAs, portraying them as both powerful and environmentally responsible computational tools.

In more recent years, the trend has shifted towards integration, scalability, and user accessibility. Frameworks like CloudiFi, introduced by [99], exemplify the industry's move towards cloud-native FPGA solutions. Concurrently, the exploration of high-level languages and toolflows, as seen in contributions from [94, 114], suggests an effort to democratize FPGA-based solutions, making them more approachable for a broader audience. Table 3.4 provides a

summary of the findings from this section.

Table 3.4: Summary of notable findings from various studies on FPGA implementations of Monte Carlo simulations.

Study	**Notable Findings**
Zhang et al. (2005) [85]	Hardware accelerator achieving 50 times speedup for financial applications.
Anlauf (2005) [86]	12 times speedup for Monte Carlo simulations on FPGAs.
Bower et al. (2006) [87]	Modular framework with speedups between 8 times to 71 times.
Morris et al. (2007) [88]	'HyperStreams' abstraction, 146 times acceleration for European option pricing.
Tian et al. (2008) [89]	Monte Carlo engine, 340 times to 750 times speedup.
Tian and Benkrid (2009) [91]	LSMC method for American options, 20 times speedup and >20:1 energy savings.
Woods et al. (2008) [92]	QMC derivative pricing, 50 times performance over 3 GHz processor.
Tian et al. (2012) [93]	LSMC method, 25 times path generation, 18 times regression, 20 times overall, 54 times energy efficiency.
Thomas (2010) [94]	Contessa language, up to 62 times speedup, significant power savings.
Anson et al. (2010) [95]	Dynamic scheduling, 44 times speedup, 19.6 times energy efficiency.

Cont.

Study	**Notable Findings**
Tian et al. (2010) [96]	Fixed-point arithmetic, enhanced throughput with negligible error.
De Schryver et al. (2011) [97]	Heston model accelerator, 89% energy savings, 2 times speedup.
Hegner et al. (2012) [98]	ASP-4P configuration, 3686 times energy efficiency.
Diamantopoulos et al. (2021) [99]	CloudiFi framework, up to 485 times improvement.
Liu et al. (2010) [100]	690 times faster for intensive tasks on FPGA-based cluster.
Tian et al. (2010) [101]	Interest rate derivative pricing, 58 times speedup, more energy efficient.
Betkaoui et al. (2010) [103]	Comparative study, FPGA 4 times greater power efficiency than Intel Xeon.
Chow et al. (2012) [104]	Mixed precision, 170 times speedup, significant energy efficiency improvements.
De Schryver et al. (2013) [105]	Multi-level Monte Carlo, 50% complexity reduction, <3.6 W power.
Jin (2013) [106]	Reconfigurable hardware, 28 times to 149 times speedup, 18.6 times energy efficiency.
De Schryver et al. (2014) [115]	800 times speed advantage, 2.8 times lower power consumption.
Jong (2014) [109]	4200 times speed up over CPU
Omland et al. (2015) [110]	MLMC simulations, 3–9 times speed-up.

Cont.

Study	Notable Findings
Choi et al. (2016) [111]	High-level synthesis, 3.89 times speedup for Black–Scholes.
Lomuscio et al. (2016) [112]	Dynamic loading, 30 times to 120 times speedup, up to 30 times energy efficiency.
Muslim et al. (2017) [113]	High-level synthesis, 2 times speedup, significant power efficiency.
Setetemela et al. (2018) [114]	High-level FPGA design toolflow.
Brugger et al. (2014) [115]	HyPER, 3.4 times faster, 36 times more power efficient.
Toft et al. (2014) [116]	Monte Carlo simulation on Xilinx Virtex-5, 46.2 times speedup, 14.4 times energy efficiency.

3.4.4 Finite Difference

Finite Difference Methods are numerical techniques used to solve differential equations by approximating them with difference equations. In financial modeling, they are commonly applied to price options, particularly for solving the Black–Scholes partial differential equation [117]. These methods involve discretizing the continuous time and asset price into a grid and calculating option prices at each grid point by iterating backward from the option's expiration to the present. The explicit finite difference method updates option prices based on known values from previous time steps, while the implicit method solves a system of linear equations at each time step, offering greater stability for larger time steps.

3.4.5 Finite Difference on FPGA

A framework for the application of financial explicit finite difference methods on reconfigurable hardware, with a focus on efficient option pricing, is pro-

posed in [118]. The study highlights a 24-fold speed increase using Virtex-6 FPGA technology. However, according to the authors, the potential of this approach on other hardware technologies remains an open question.

Introducing a parallel pipelined architecture, the work in [119] aims to accelerate financial computations using the finite difference method on both reconfigurable hardware and GPUs. This architecture is capable of handling simultaneous option pricing requests with impressive throughput. The design, when implemented on an XC6VLX760 FPGA, achieved over 12 times the speed of a Pentium 4 processor and was 9.4-times more energy-efficient than a GPU. Future directions for this research encompass optimizations on modern hardware cores, synergies between FPGA and GPU, and the exploration of automated domain-specific strategies.

In the study reported in [120], a novel strategy is put forth to enhance explicit finite difference option pricing by dynamically reconfiguring constants. This approach supports both complete and partial runtime reconfigurations. An analysis on a Virtex-6 XC6VLX760 device showed that partial reconfiguration can yield a speed-up of 4.7 times compared to a static design, while full reconfiguration was found to be less advantageous due to extended reconfiguration durations.

Lastly, the technical report by [121] examines power consumption on various computing platforms, notably desktop computers and FPGA boards, utilizing a Hall effect-based current sensor. The application chosen for this power assessment is a Monte Carlo simulation tailored for European option pricing. The report further mentions that the combined FPGA and CPU system consumes less than 1% of the energy compared to the energy required by the CPU alone.

Key Observations

The explicit finite difference methods, when implemented on such hardware, have shown significant speed-ups, often outperforming traditional CPU-based solutions. Notably, the dynamic reconfiguration of constants, as discussed in some of the studies, offers a flexible approach to optimization, allowing for both full and partial runtime adjustments. This adaptability, however, comes with its own set of challenges, especially when considering the extended durations associated with full reconfigurations. Another observation is the emphasis on energy efficiency. As computational demands grow, the energy consumption of these processes becomes a critical concern. FPGA-based solutions,

in many instances, have demonstrated superior energy efficiency compared to GPUs and traditional processors.

The trajectory of research in this domain indicates a growing interest in automating the deployment of financial algorithms on reconfigurable hardware. This trend underscores the importance of ease of implementation and adaptability in rapidly evolving financial markets. Furthermore, there is a noticeable shift towards exploring synergies between different hardware platforms, such as FPGAs and GPUs. Such collaborations could potentially harness the strengths of both platforms, leading to even more optimized solutions. The focus on domain-specific strategies, and the fine-tuning of arithmetic types and word lengths, also suggest a move towards more tailored and precise computational solutions. Lastly, as energy consumption becomes a focal point in computational research, we anticipate further studies benchmarking and optimizing the energy profiles of reconfigurable hardware in financial computations. Table 3.5 shows the summary of these studies.

Table 3.5: Summary of notable findings from various studies on FPGA implementations of finite difference methods.

Study	**Notable Findings**
Jin et al. (2011) [118]	24 times speed increase using Virtex-6 FPGA.
Jin et al. (2009) [119]	12 times speed of Pentium 4, 9.4 times more energy-efficient than GPU on xc4vlx160 FPGA.
Becker et al. (2011) [120]	Dynamic reconfiguration on Virtex-6 XC6VLX760, 4.7 times speed-up for partial reconfiguration.
Albicocco et al. (2012) [121]	Combined FPGA and CPU system consuming less than 1/100th of the energy of CPU alone.

3.4.6 Heston Model

The Heston Model, developed by Steven Heston in 1993 [122], is a widely used stochastic volatility model for pricing options. Unlike the Black–Scholes model, which assumes constant volatility, the Heston Model allows for stochastic (randomly varying) volatility, making it more realistic for financial markets where volatility tends to fluctuate over time. The model is particularly noted for its ability to capture the "volatility smile", a pattern where implied volatility varies with the strike price and maturity of the option.

3.4.6.1 Heston Model on FPGA

Using the Heston model as a representative problem-model pairing [123], the efficient exploration of the vast design space is illustrated. The paper presents a detailed methodology for the design and evaluation of optimal hardware accelerators. Through case studies, it introduces a new hardware random number generator for various distributions and a specialized hardware accelerator for computing European barrier option prices. Furthermore, a unique benchmark is proposed for a balanced assessment and comparison of different accelerator implementations in financial mathematics. However, at the time of publishing, the authors were still working on an FPGA implementation.

In a comparative study between constructing hardware in a Scala Embedded Language (Chisel) and VHDL, the Heston Model was implemented [124]. The results indicated a 30% reduction in code size with Chisel and similar hardware utilization as VHDL. However, certain challenges, like the absence of floating point support and complications with Xilinx IP cores, suggest that the current version of Chisel 2.0 may not be ideal for arithmetic-heavy models like Heston. Nevertheless, Chisel's potential for future enhancements is evident.

Another study introduced a heterogeneous computing platform integrating GPU and FPGA for power-efficient high-performance computing [125]. This research analyzed various applications, including the Heston Model for option pricing. The findings underscored the importance of aligning computation tasks with the right computing architectures to achieve optimal performance and power efficiency. The reported efficiency of the FPGA compared to the GPU was 1.84×10^9 Ops/Joule.

In a recent paper, the Heston stochastic volatility model, in conjunction with the Longstaff and Schwartz path reduction, was applied for market risk analysis on a Xilinx Alveo U280 FPGA [126]. By implementing efficiency-centric techniques, the algorithm was optimized. The outcomes displayed a notable performance boost on the FPGA compared to two 24-core Intel Xeon Platinum CPUs, with performance improvements ranging from 8 to 185 times. The difference between the initial and the optimized algorithm was 320 times, underscoring the exceptional computational performance and energy efficiency. They also report a power efficiency of 2.6379×10^9 Ops/Joule.

3.4.6.2 Key Observations

The consistent mention of the Heston model across studies indicates its significance in financial mathematics and hardware acceleration research. Additionally, the emphasis on power efficiency and performance optimization showcases the industry's drive towards more sustainable and efficient computational solutions.

There is a clear trend towards the exploration of heterogeneous computing platforms, combining the strengths of different architectures like GPU and FPGA. The push for efficiency-driven techniques and the continuous evolution of embedded languages like Chisel also highlight the industry's commitment to innovation and optimization. Table 3.6 provides a summary of this section.

Table 3.6: Summary of notable findings from various studies on FPGA implementations of the Heston Model.

Study	**Notable Findings**
De Schryver et al. (2011) [123]	Detailed methodology for designing and evaluating optimal hardware accelerators for the Heston model; introduced new hardware random number generator and accelerator for European barrier option prices.
Stumm (2013) [124]	Comparative study between Chisel and VHDL for the Heston Model; 30% reduction in code size with Chisel, but challenges with floating point support and Xilinx IP cores.
Wu et al. (2014) [125]	Heterogeneous computing platform integrating GPU and FPGA; analyzed Heston Model for option pricing; FPGA 1.84 $\times$ 10^9 Ops/Joule efficiency compared to GPU.
Klaisoongnoen et al. (2022) [126]	Applied Heston model on Xilinx Alveo U280 FPGA; 8 to 185 times performance improvement over two 24-core Intel Xeon Platinum CPUs; 2.6379 $\times$ 10^9 Ops/Joule power efficiency.

3.4.7 Quadrature Methods

Quadrature Methods are numerical techniques used to evaluate integrals, which can be applied to option pricing in financial models [127]. These methods provide a way to compute the expected payoff of an option by integrating over all possible future paths of the underlying asset's price. They are particularly useful for pricing options with complex features, such as path dependency and early exercise options, where analytical solutions are difficult to obtain.

3.4.7.1 Quadrature Methods on FPGA

An FPGA-accelerated architecture is introduced for pricing complex options using quadrature methods. Anson's study [128] reports a 32.8 times speedup and an 8.3 times increase in power efficiency over a Tesla C1060 GPU. Future work will focus on leveraging the latest FPGAs, automating efficient FPGA and GPU implementations, and optimizing hardware designs using quadrature methods for applications in electromagnetic problem-solving and photon distribution calculations.

Presenting a framework for automatically generating hardware accelerators for finance applications, Jin [129] specifically addresses European option pricing problems via the quadrature method using a high-level software language (C#). The results indicate that the automatically generated accelerators on an xc4vlx160 FPGA can run over 18-times faster and be up to 143-times more power efficient than a Pentium 4 processor in single precision arithmetic. In double precision arithmetic, these accelerators are 7-times faster and 77-times more power efficient.

A novel parallel architecture for accelerating quadrature methods for pricing multi-dimensional options, such as discrete barrier, Bermudan, and American options, is introduced in another study. Tested on various platforms, including reconfigurable logic and a compute unified device architecture (CUDA)-based graphics processing unit (GPU), the findings [130] show that a 100 MHz Virtex-4 xc4vlx160 FPGA design outperforms an optimized multi-threaded software implementation running on a Xeon W3504 dual-core CPU by being 4.6 times faster and 25.9 times more energy-efficient. Additionally, it is 2.6 times faster and 25.4 times more energy-efficient than a GPU with similar silicon process technology.

Considering the integration grid density and the mantissa size of floating-point operators, research presented in [131] introduces a precision optimization methodology for quadrature computation on reconfigurable hardware. The proposed approach achieves up to 6 times faster performance compared to FPGA designs with double precision arithmetic. Moreover, it surpasses an i7-870 quad-core CPU by up to 15.1 times in speed and 234.9 times in energy efficiency, and a Tesla C2070 GPU by 1.2 times in speed and 42.2 times in energy efficiency.

3.4.7.2 Key Observations

The consistent mention of quadrature methods as a preferred technique for option pricing highlights its significance in the domain. Power efficiency, alongside speedup, emerges as a critical metric, with multiple studies showcasing the superiority of FPGA implementations over traditional CPUs and GPUs in both these aspects.

There is a clear inclination towards automating the generation of hardware accelerators which leverages high-level software languages for this purpose. This trend indicates a move towards making FPGA implementations more accessible and less reliant on low-level hardware design expertise. Another discernible trend is the exploration of precision optimization methodologies. Such methodologies aim to strike a balance between computational accuracy and performance, optimizing for specific application needs. The consistent comparison against GPUs and CPUs also suggests a competitive environment where FPGAs are continually being benchmarked against other prevalent architectures to demonstrate their advantages. Table 3.7 summarizes these studies.

Table 3.7: Summary of notable findings from various studies on FPGA implementations of Quadrature Methods.

Study	**Notable Findings**
Anson (2009) [128]	FPGA architecture for complex options pricing; 32.8 times speedup and 8.3 times power efficiency over Tesla C1060 GPU.
Jin (2009) [129]	Automated hardware accelerators for European options via quadrature method; 18 times faster and 143 times more power efficient (single precision); 7 times faster and 77 times more power efficient (double precision) than Pentium 4.
Anson (2011) [130]	Parallel architecture for multi-dimensional options; Virtex-4 xc4vlx160 FPGA 4.6 times faster and 25.9 times more energy-efficient than Xeon W3504 dual-core CPU; 2.6 times faster and 25.4 times more energy-efficient than a comparable GPU.
Tse (2012) [131]	Precision optimization for quadrature computation; up to 6 times faster than FPGA designs with double precision; 15.1 times faster and 234.9 times more energy-efficient than i7-870 CPU; 1.2 times faster and 42.2 times more energy-efficient than Tesla C2070 GPU.

3.4.8 Miscellaneous Models

The hardware implementation for Collateralized Debt Obligations (CDOs) pricing using the One-Factor Gaussian Copula (OFGC) model has been explored and, when deployed on a Xilinx XC5VSX50T, was 63 times faster than a software implementation running on a 3.4 GHz Intel Xeon processor in [132]. Further studies have looked into hardware implementations for both One-Factor and Multi-Factor Gaussian Copula models, achieving increases between

64 and 71 times compared to corresponding software running on a 3.4 GHz Intel Xeon processor [133, 134].

Next, we consider papers that look into risk analysis, VaR estimation, and related topics. Efficient FPGA-based methods have been proposed for generating vectors used in modeling correlations between stochastic time-series [135] and were 26 times faster than a quad Opteron 2.6 GHz SMP. Additionally, optimized hardware designs of a Multivariate Gaussian random number generator (MVGRNG) for FPGA platforms have been presented, focusing on financial applications like Value-at-Risk (VaR) estimation [136], where the authors reported an improvement in performance of up to 96% for VaR calculation while up to 81% improvement was observed for option pricing. A suite of hardware accelerators for financial applications, specifically for risk valuation models, has also been demonstrated in [137]. In the previous example, the authors show the Black–Scholes and Black-76 accelerators can, respectively, process data up to 348 and 297 times faster than the software implementation. They also show a Binomial accelerator, which could not be fully pipelined due to limited resources, but could process data up to 38 times faster than the software implementation.

The performance and energy consumption of GPUs and FPGAs through the implementation of the Black–Scholes and Heston option pricing models have been explored [138] where they show that energy per computation by using an FPGA can be from 5.9% to 9.8% (depending on the algorithm) that of using a GPU and performance can be from 1.71 times to 2.56 times as fast as the GPU. Novel methods for pricing multi-dimensional American options on heterogeneous CPU/FPGA systems have been introduced [139, 140] showing a 2 times speedup compared to state-of-the-art and a significant power saving (Zynq FPGA consumes only 7 kJ, while the low power Atom needs 614 kJ). Furthermore, the implementation of an Asian option pricing application on a high-performance computing (HPC) system has been explored [141] showing between 4 times and 9.2 times speedup compared to a multi-threaded software implementation and power efficiency improvements of up to 45 times compared to traditional CPU implementations. Tools like OXiGen have been designed for translating high-level code functions into optimized FPGA-based dataflow kernels, with applications in option pricing [142, 143] where it demonstrated a 88.1 times speed-up compared to a single-threaded software implementation. Research has also been conducted on accelerating the computationally-demanding Gaussian Copula Model (GCM) in financial analy-

sis [144, 145] where the authors were able to achieve a 4 times speedup compared to a software implementation on CPU.

FPGA-accelerated approaches have been introduced for processing market data feeds directly from the network [146] where they showed a 12 times improvement over the available (at the time) real-world rate. Other research has highlighted the ability of FPGA IP libraries to significantly reduce (by a factor of 2) latency in electronic trading [147]. They also mention that the Celoxica AMDC accelerator card, which uses a Xilinx Virtex 5 LX110T FPGA device, was measured to consume less than 15 watts of power from its host server. Additionally, FPGA hardware designs have been introduced to accelerate, via a 4 times latency reduction in comparison to the conventional software based approach, High Frequency Trading (HFT) applications [148]. An automation framework for Iterative Stencil Loops (ISLs) has also been presented, with potential applications in financial data processing [149]. A novel high-performance computing approach has been introduced for predicting financial market trends, particularly options [150].

Additionally, a high-performance FPGA-based tridiagonal solver, optimized for hardware acceleration, has been introduced for its application in derivatives pricing problems [151], where they showed a 36 times speed-up and 16 times speed-up for the fixed-point and floating-point designs respectively, compared to a CPU implementation and power efficiency of up to 16 times greater than that of a GPU. Special purpose processing elements have been described for optimizing investment strategies in financial securities [152] where the authors demonstrate a speedup of more than 17,000 compared to a high-performance PC. They also report the power consumption of the FPGA (ranging from 12 W to 45 W) is significantly smaller compared to the CPU and GPU used in the study. The CPU, a quad-core Core i7 870, has a thermal design power (TDP) of 95 W, while the GPU, a Tesla C2075, has a TDP of 225 W. Novel optimization methodologies have been presented for reducing hardware resource consumption in stencil-based numerical procedures on reconfigurable hardware [153] where they show a 2 times speedup as well. Automation optimization algorithms have been introduced for reconfigurable designs targeting FPGAs [154]. The authors show that an optimized design is over 100 times faster and up to 200 times more power efficient than the basic configuration. OpenCL compilation frameworks have been designed for generating high-performance hardware on FPGAs [155, 156]. The authors showed a 1.2 times speedup compared to a handcrafted implementation. The performance-

programmability gap in FPGA programming has been addressed by employing OpenCL in other papers also, e.g., [157] where a 1.22 times performance improvement was demonstrated.

Key Observations

The hardware acceleration capabilities of FPGAs have been leveraged in various financial applications, ranging from option pricing to high-frequency trading. Notably, the adaptability of FPGAs allows for fine-tuning and optimization, which is particularly beneficial for financial models that require high precision and low latency. Furthermore, the introduction of tools and frameworks, such as OpenCL compilation frameworks and ARDEGO, have simplified the process of programming and optimizing FPGAs, making them more accessible for financial computations.

The increasing complexity of financial models and the demand for real-time processing have driven the shift towards FPGA-based systems. As observed, there is a growing trend in utilizing FPGA for high-frequency trading, where latency is a critical factor. Additionally, the development of FPGA IP libraries and specific processing elements indicates a move towards more specialized and tailored hardware solutions for financial applications. The exploration of hybrid-core computing and the emphasis on energy efficiency also suggest a broader trend in the financial industry, where both performance and sustainability are becoming critical. Lastly, the continuous efforts in automating and optimizing FPGA implementations, as seen with tools like OXiGen and Senju, highlight the industry's commitment to harnessing the full potential of FPGA technology in the coming years. Table 3.8 shows a summary of these miscellaneous methods.

Table 3.8: Summary of notable findings from various studies on FPGA implementations of miscellaneous models.

Study	**Notable Findings**
Kaganov et al. (2008) [132]	CDO pricing using OFGC model; 63 times faster than software on Intel Xeon.
Kaganov et al. (2011) [134]	One-Factor and Multi-Factor Gaussian Copula models; 64–71 times faster than software on Intel Xeon.
Thomas et al. (2008) [135]	Generating vectors for modeling correlations; 26 times faster than quad Opteron 2.6 GHz SMP.
Saiprasert et al. (2010) [136]	MVGRNG for VaR estimation; up to 96% improvement in performance.
Stamoulias et al. (2017) [137]	Hardware accelerators for risk valuation; Black–Scholes and Black-76 up to 348 times and 297 times faster, Binomial up to 38 times faster.
Ma et al. (2016) [138]	Black–Scholes and Heston models; FPGA 5.9%-9.8% energy per computation, 1.71–2.56 times faster than GPU.
Varela et al. (2015) [139]	Multi-dimensional American options on CPU/FPGA; 2 times speedup, Zynq FPGA consumes only 7 kJ vs. 614 kJ for Atom.
Nestorov et al. (2017) [141]	Asian option pricing on HPC; 4–9.2 times speedup, 45 times power efficiency improvement.
Peverelli et al. (2018) [142]	OXiGen tool for FPGA-based kernels; 88.1 times speedup over single-threaded software.
Ibraev (2020) [144]	Gaussian Copula Model acceleration; 4 times speedup compared to CPU.
Morris et al. (2009) [146]	Market data feed processing; 12 times improvement over real-world rate.

Cont.

Study	**Notable Findings**
Lockwood et al. (2012) [147]	FPGA IP libraries for electronic trading; 2 times latency reduction, Celoxica AMDC accelerator card < 15 W power.
Leber et al. (2011) [148]	High Frequency Trading applications; 4 times latency reduction vs. software.
Del Sozzo et al. (2023) [149]	Senju automation framework for ISLs; potential applications in financial data processing.
Lázaro García (2013) [150]	HPC approach for predicting market trends; significant speedup for options.
Palmer (2014) [151]	FPGA-based tridiagonal solver; 36 times speedup (fixed-point), 16 times speedup (floating-point), 16 times power efficiency over GPU.
Starke et al. (2012) [152]	Optimizing investment strategies; speedup $>$ 17,000 times compared to high-performance PC, FPGA 12–45 W vs. CPU 95 W and GPU 225 W.
Jin et al. (2012) [153]	Optimization for stencil-based numerical procedures; 2 times speedup.
Kurek (2014) [154]	Automation optimization algorithms
Czajkowski et al. (2012) [155]	OpenCL compilation frameworks; 1.2 times speedup compared to handcrafted implementation.
Krommydas et al. (2016) [157]	OpenCL for FPGA programming; 1.22 times performance improvement.

3.5 Challenges and Future Directions

In this section, we examine the challenges surrounding the use of FPGAs and future directions for development and research in this space.

3.5.1 Challenges

The development and optimization of FPGA-based systems highlight several challenges in the field. While these systems have consistently shown superiority in many applications, not all implementations guarantee optimal performance. Efficient design is particularly critical, especially in terms of communication times with the FPGA, which remains a complex challenge. Furthermore, as computational demands increase, managing energy consumption becomes increasingly important. Ensuring that FPGAs maintain energy efficiency in all scenarios is a significant ongoing struggle for both researchers and developers. Additionally, despite advances in High-Level Synthesis (HLS) tools, making FPGA platforms universally accessible and user-friendly is still a hurdle, particularly for those without extensive hardware expertise. Moreover, although there is a trend towards heterogeneous computing, achieving integration of FPGAs with other platforms, like GPUs, without compromising performance continues to be problematic. Lastly, with the growing complexity of financial models, the development of domain-specific strategies and the fine-tuning of arithmetic types and word lengths tailored to specific financial computations present persistent challenges.

3.5.2 Future Directions

Looking forward, the continuous evolution of FPGA-based architectures is anticipated. Future research could examine innovative areas such as parallel pipelined architectures, systolic hardware designs, and the application of formal mathematical frameworks. The research area for FPGA applications in finance is also expanding, with potential explorations into multi-asset option pricing and high-frequency trading on the horizon. Further emphasis on user accessibility is critical; future directions might include the development of more intuitive frameworks and high-level languages to make FPGA-based solutions more approachable for a broader audience. Additionally, exploring deeper synergies between different hardware platforms could harness

the strengths of FPGAs, GPUs, and other architectures more effectively. Automating the generation of hardware accelerators is another promising area, potentially making FPGA implementations more accessible and reducing the reliance on low-level hardware design expertise. As financial models become increasingly complex, there is a growing need for specialized FPGA IP libraries and specific processing elements tailored for financial applications. Finally, the exploration of hybrid-core computing and a continued emphasis on energy efficiency could play pivotal roles in shaping the future of FPGA applications in finance.

3.6 Conclusions

The integration of Field Programmable Gate Arrays (FPGAs) into financial computation has proven to be highly effective, particularly in the field of option pricing, where their ability to accelerate complex models has been extensively documented. Our comprehensive review of 99 studies highlights significant speed and efficiency gains achievable with FPGAs, which often exceed those of traditional computing solutions by several orders of magnitude. For example, FPGAs can deliver performance gains ranging from 270 to 5400 times faster than conventional CPU implementations depending on the specific option pricing model employed.

Despite these advances, several challenges hinder the universal adoption of FPGAs. Key among these are the difficulties of FPGA programming and the need for optimized design frameworks that enhance both power efficiency and ease of integration with existing computing infrastructures. Also, adoption trends for FPGAs, GPUs, and CPUs reveal a sharp contrast: while GPUs have become the standard for high-performance computing (HPC) due to their massively parallel architectures and have seen significant adoption in both general-purpose and AI-specific tasks, FPGAs have not kept pace with this growth. Despite their potential for specific applications with irregular data types and high energy efficiency, FPGAs face challenges such as low memory bandwidth, complex programming requirements, and high development times, which limit their adoption compared to GPUs and CPUs [158]. Future research should thus focus on simplifying the programming environment and enhancing the design tools that facilitate the broader application of FPGAs in financial systems. The importance of these improvements is underscored by

the fact that FPGAs have demonstrated energy efficiency improvements of up to 30% more floating-point operations per Joule compared to traditional systems.

At the end of this review we have shown that, with the exception of our own work of course, research into applying Pascal's Simplex to Option Pricing on FPGA has not been carried out. In the next chapter we will present how we can transform the existing theoretical solution to make it applicable in practice.

Chapter 4

Implementation of a Multinomial Tree Method for Option Pricing

4.1 Introduction

The exploration of financial models and computational methods, as described in the preceding chapters, lays the foundation for the practical implementation of these theoretical frameworks in a real-world context. The multinomial tree method, known for its application in option pricing, provides a robust model that can be harnessed to generate tangible and actionable insights in the financial domain. This chapter looks towards providing a concrete implementation of the multinomial tree method, particularly focusing on its deployment on Field-Programmable Gate Arrays (FPGAs). Thus, this chapter serves as the bridge between the theoretical model presented in published research and the realization of this model in hardware.

The dynamics of financial markets necessitate the development and application of sophisticated models to navigate, predict, and strategize financial decisions. The multinomial tree method, as discussed in[159], presents a compelling methodology for option pricing, offering a discrete-time model that can approximate a wide array of exotic and path-dependent options. While the theoretical underpinnings of this method have been explored, a gap persists in translating this theory into a viable, efficient, and scalable computational model. FPGAs, with their parallel processing capabilities and configura-

tional flexibility, emerge as a potent platform to bridge this gap, enabling the rapid and accurate computation of option prices.

This chapter aims to navigate through the journey from theoretical understanding to practical realization, investigating the process of implementing the multinomial tree method on an FPGA. The primary objectives encompass:

- Summarize the Sierag-Hanzon [159] approach such that the reader can understand the relationship between the theory and the implementation proposed in this chapter.
- Providing a detailed breakdown of the algorithmic translation of the multinomial tree method into a computational model.
- Discussing the design, structure, and functionality of the the model, ensuring it is tailored to harness the capabilities of FPGAs.
- Highlighting the strategic decisions and optimizations employed to enhance computational efficiency and accuracy in the implementation.

Section 4.2 presents the theoretical framework, providing a refresher and establishing a foundation for the implementation strategy. Section 4.3 navigates through the implementation strategy, code design, and functionality, culminating in a discussion in Section 4.4 that presents the implementation. Section 4.5 explores its implications, and discusses the challenges and limitations encountered. While testing, validation, and performance evaluation are crucial facets of this implementation, they will be explored in-depth in the following chapters, ensuring a focused and thorough exploration of each phase of this research.

4.2 Theoretical Framework

The multinomial tree method, a crucial element in financial mathematics, functions as an effective model for option pricing, offering a structured and adaptable approach to addressing the complexities of financial derivatives. This section aims to revisit and consolidate the theoretical foundations of the multinomial tree method, establishing a solid basis for its subsequent implementation.

4.2.1 The Sierag-Hanzon Model

Sierag and Hanzon introduce an innovative model for pricing multi-asset derivatives by building on the widely used binomial tree in finance and extending it to multinomial trees based on Pascal's simplex. The traditional binomial tree is a fundamental tool in financial modeling (as discussed in previous chapters), allowing for the simulation of an asset's price as it moves up or down over time with each step. This tree structure is typically "multiplicative," meaning price changes are proportional—each step reflects a percentage change rather than an absolute increase or decrease. For example, in a binomial tree for stock prices, the stock price might move up by a factor of 1.1 or down by a factor of 0.9, reflecting a 10% increase or a 10% decrease, respectively.

One of the advantages of the binomial tree model is that it can be extended to a "multinomial tree," where multiple assets or more complex price paths are simulated. For instance, in a multinomial tree, each step can represent changes in the prices of several assets simultaneously. In Sierag and Hanzon's model, they use Pascal's simplex to structure these multinomial trees, which enables them to incorporate more than one asset in a single tree structure, making the model adaptable to complex multi-asset financial products like options on portfolios of stocks or multi-currency options.

A unique aspect of Sierag and Hanzon's model is the ability to switch between the traditional multiplicative structure and an "additive" version, where each step represents an absolute amount added or subtracted. The transformation between these two versions is achieved through a logarithmic function. For instance, by taking the logarithm of each value in a multiplicative tree, the resulting tree becomes additive, where each step adds or subtracts a fixed amount. This transformation is crucial because, as the tree depth increases (i.e., the number of steps grows large), additive multinomial trees naturally begin to resemble a Gaussian, or bell curve, distribution. This is a useful property, as Gaussian distributions are often used to approximate real-world relative price changes and the corresponding lognormal distributions are well-suited for simulating asset prices in a model.

To ensure that the prices in the multinomial tree behave in a standard, predictable way, Sierag and Hanzon's model uses a mathematical technique called the Cholesky factorization (which we discuss in subsection 4.2.3). This

method transforms the distributions in the additive tree so that they have a mean of zero and unit variance, creating a standardized class of multinomial trees. For example, if we're simulating two assets with very different volatilities, we can use the Cholesky factor to adjust them so that each has the same basic "scale."

Once this standardized tree structure is established, Sierag and Hanzon convert it back to a multiplicative form by applying an exponential function, which is necessary to reflect how prices typically change in financial markets (i.e., in terms of percentage rather than fixed amounts). This conversion allows their model to maintain the multiplicative nature of real asset prices, which fluctuate by proportional amounts.

While this approach offers a powerful way to model complex, multi-asset derivatives, it also requires extensive computation, especially as the number of assets increases. The authors suggest using Field-Programmable Gate Arrays (FPGAs)—a type of specialized hardware optimized for fast, parallel calculations—to handle the computational demands efficiently. This can make the model more practical for real-world applications where computational resources are limited.

Since multinomial trees based on Pascal's simplex and the associated use of Cholesky transformations are relatively new concepts in financial modeling, it's beneficial for readers to have a clear understanding of these structures. Pascal's simplex is a generalization of Pascal's triangle to higher dimensions, which allows for the creation of multinomial trees that can handle multiple assets within the same framework. This approach is at the core of Sierag and Hanzon's model, providing a tool for pricing derivatives in complex, multi-asset environments.

4.2.2 Key Concepts and Definitions

In the context of option pricing, the multinomial tree method employs a lattice framework, wherein the price of the underlying asset is modeled to move among multiple possible nodes or prices at each discrete time step. Each node represents a potential vector of prices that the assets can attain, and the transitions between nodes are governed by probabilistic measures, which are derived from the assumptions about the underlying asset's price dynamics. The method provides a versatile structure that can be adapted to various types of

options and assets, accommodating different payoff structures and expiration times.

Pascal's Triangle Pascal's Triangle is a mathematical construct that presents a triangular array of binomial coefficients. It is constructed by starting with a single '1' at the top. Each subsequent row is generated by adding the two numbers directly above in the preceding row, placing the sum in the position below. Mathematically, the entries of Pascal's Triangle, denoted $C(n, k)$, can be computed using the binomial coefficient formula:

$$C(n, k) = \frac{n!}{k!(n-k)!} \tag{4.1}$$

where n is the row number (starting from 0), k is the position in the row (starting from 0), and '!' denotes the factorial function. The triangle visually represents the coefficients in the binomial expansion of $(x + y)^n$, providing a powerful tool in combinatorics, algebra, and calculus.

$$\begin{array}{ccccccccc}
 & & & & \binom{0}{0} & & & & \\
 & & & \binom{1}{0} & & \binom{1}{1} & & & \\
 & & \binom{2}{0} & & \binom{2}{1} & & \binom{2}{2} & & \\
 & \binom{3}{0} & & \binom{3}{1} & & \binom{3}{2} & & \binom{3}{3} & \\
\binom{4}{0} & & \binom{4}{1} & & \binom{4}{2} & & \binom{4}{3} & & \binom{4}{4}
\end{array}$$

Figure 4.1: A portion of Pascal's Triangle.

Two-Dimensional Lattice and Relevance of Pascal's Triangle A two-dimensional lattice is a grid formed by points, often used to model various problems in financial mathematics, such as option pricing. Each point, or node, in the lattice represents a potential state or value of a financial instrument at a given point in time. Consider a binomial lattice, where each node connects to two subsequent nodes, representing an upward or downward movement in the price of an asset. The number of paths to reach a node at (n, k) in the lattice can be determined using Pascal's Triangle. The entry in the n-th row and k-th column of Pascal's Triangle gives the number of distinct paths to the node (n, k) in the lattice, assuming we start at $(0, 0)$. Mathemati-

cally, the number of paths, $P(n,k)$, is given by the binomial coefficient:

$$P(n,k) = \binom{n}{k} = \frac{n!}{k!(n-k)!} = C(n,k) \tag{4.2}$$

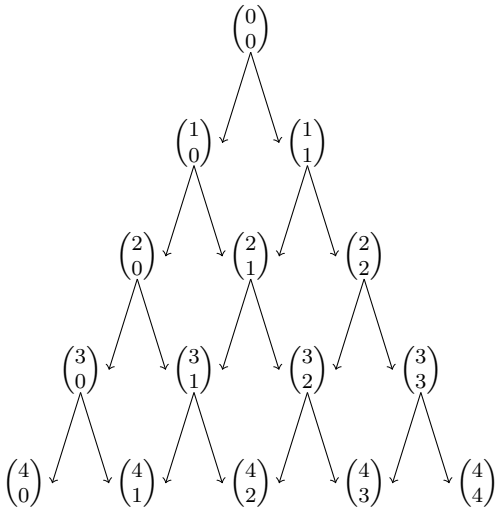


Figure 4.2: A binomial lattice with entries from Pascal's Triangle.

4.2.3 Models and Equations

The multinomial tree method is underpinned by specific models and equations that dictate the asset price movements, transition probabilities, and option valuations. The asset price at each node is determined by multiplying the preceding node's price by a factor, which represents an upward, downward, or stationary movement. The transition probabilities are derived from risk-neutral valuation and are utilized to compute the expected payoff at each node, which is then discounted back to the present value using a risk-free rate. The option price is ultimately determined by traversing through the tree, calculating, and discounting expected payoffs from the expiration back to the present time.

Cholesky Decomposition of the Covariance Matrix Σ Cholesky decomposition is a mathematical technique used to decompose a positive-definite matrix into the product of a lower triangular matrix and its transpose. A positive-definite matrix is a symmetric matrix A for which $\mathbf{x}^T A\mathbf{x} > 0$ for any non-zero vector $\mathbf{x}$, ensuring all its eigenvalues are positive and it can be decomposed via the Cholesky decomposition. A lower triangular matrix is a square matrix in which all the elements above the main diagonal are zero.

When applied to the covariance matrix Σ, it provides a way to represent the covariance structure of asset returns in a simpler form.

Example: Consider a simple covariance matrix Σ for two assets:

$$\Sigma = \begin{pmatrix} 4 & 2 \\ 2 & 3 \end{pmatrix}$$

To find the Cholesky decomposition of Σ, we need to find a lower triangular matrix L such that $\Sigma = LL^T$. The matrix L will have the form:

$$L = \begin{pmatrix} l_{11} & 0 \\ l_{21} & l_{22} \end{pmatrix}$$

We can determine the elements of L as follows:

1. Calculate l_{11}:
$$l_{11} = \sqrt{\Sigma_{11}} = \sqrt{4} = 2$$

2. Calculate l_{21}:
$$l_{21} = \frac{\Sigma_{21}}{l_{11}} = \frac{2}{2} = 1$$

3. Calculate l_{22}:
$$l_{22} = \sqrt{\Sigma_{22} - l_{21}^2} = \sqrt{3 - 1^2} = \sqrt{2}$$

Thus, the lower triangular matrix L is:

$$L = \begin{pmatrix} 2 & 0 \\ 1 & \sqrt{2} \end{pmatrix}$$

We can verify this decomposition by multiplying L by its transpose L^T:

$$LL^T = \begin{pmatrix} 2 & 0 \\ 1 & \sqrt{2} \end{pmatrix} \begin{pmatrix} 2 & 1 \\ 0 & \sqrt{2} \end{pmatrix} = \begin{pmatrix} 4 & 2 \\ 2 & 3 \end{pmatrix}$$

This confirms that the Cholesky decomposition is correct. Mathematically, for

a covariance matrix Σ, the Cholesky decomposition is given by:

$$\Sigma = LL^T$$

where L is a lower triangular matrix.

In the context of option pricing, the covariance matrix Σ captures the relationships between the returns of different assets. Understanding these relationships is crucial for modeling the joint behavior of asset prices, especially when pricing options on multiple assets or considering portfolio effects.

The Cholesky decomposition simplifies this covariance structure, allowing for more efficient simulations and calculations. By using the decomposed form, we can also generate correlated asset paths for Monte Carlo simulations, a common method in option pricing. This ensures that the simulated asset paths reflect the observed relationships between assets, leading to more accurate option valuations.

In summary, Cholesky decomposition provides a computationally efficient way to incorporate the covariance structure of assets into option pricing models.

Rotation Matrices The purpose of the rotation matrices in this context is to facilitate the projection of a random walk in a higher-dimensional space onto a lower-dimensional space that represents the price dynamics of multiple financial assets. Specifically, the rotation matrices serve to:

1. **Align the Space for Projection**: The rotation matrices align the coordinate system such that the projection corresponds to the hyperplane orthogonal to a specific vector (ι) representing the probabilities of movement along different axes in the original space.

2. **Simplify Multivariate Mapping**: After rotation, the system's dimensionality is reduced by projecting onto a hyperplane, which simplifies the representation of movements in $(k+1)$-dimensional space to k-dimensional space.

3. **Preserve Orthogonality and Probabilities**: The rotated and projected vectors maintain orthogonality and preserve the probabilities necessary for constructing recombining multinomial trees. These trees are utilized to model the price dynamics of multiple assets under specific stochastic processes.

This approach ensures that the recombining multinomial tree structure accurately represents the underlying geometric Brownian motion of the assets

while enabling efficient computation for derivative pricing.

Suppose that $\alpha \in \mathbb{R}$. We define the rotation function $R_\alpha : \mathbb{R}^2 \to \mathbb{R}^2$ as follows: Any vector in the plane can be written in polar coordinates as $r(\cos(\theta), \sin(\theta))$ where $r \geq 0$ and $\theta \in \mathbb{R}$. For any such vector, we define

$$R_\alpha \left(r(\cos(\theta), \sin(\theta))\right) = r(\cos(\theta + \alpha), \sin(\theta + \alpha)).$$

Notice that the function R_α doesn't change the norms of vectors (the number r), it just affects their direction, which is measured by the unit circle coordinate. We call the function R_α the *rotation of the plane by angle* α.

If $\alpha > 0$, then R_α rotates the plane counterclockwise by an angle of α. If $\alpha < 0$, then R_α is a clockwise rotation by an angle of $|\alpha|$. The rotation does not affect the origin in the plane. That is, $R_\alpha(0, 0) = (0, 0)$ always, no matter which number α is.

Examples

- $R_{\frac{\pi}{2}}$ is the function that rotates the plane by an angle of $\frac{\pi}{2}$, or 90°. Because $\frac{\pi}{2} > 0$, it is a counterclockwise rotation. Thus, $R_{\frac{\pi}{2}}(1, 1)$ is the point $(-1, 1)$ in the plane that we obtain by rotating $(1, 1)$ counterclockwise by an angle of $\frac{\pi}{2}$.
- Because $-\frac{\pi}{2} < 0$, $R_{-\frac{\pi}{2}}$ is a clockwise rotation. $R_{-\frac{\pi}{2}}(1, 1)$ is the point $(1, -1)$ in the plane obtained by rotating $(1, 1)$ clockwise by an angle of $\frac{\pi}{2}$.
- The function $R_0 : \mathbb{R}^2 \to \mathbb{R}^2$ rotates the plane by an angle of 0. That is, it doesn't rotate the plane at all. It's just the identity function for the plane.

Composing Rotations Theorem: $R_\alpha \circ R_\beta = R_{\alpha+\beta}$

Proof: If first we rotate the plane by an angle of β, and then we rotate the plane by an angle of α, we have rotated the plane by an angle of $\alpha + \beta$. That's what this theorem says.

$$R_\alpha \circ R_\beta = R_{\alpha+\beta}$$

Corollary: $R_\alpha^{-1} = R_{-\alpha}$

Proof: As discussed earlier, the rotation R_0 is a rotation by an angle of 0, which means R_0 doesn't rotate anything at all. It's the identity function on the

plane. That is, $R_0 = id$.

Using the theorem we see that

$$R_\alpha \circ R_{-\alpha} = R_{\alpha-\alpha} = R_0 = id \quad \text{and} \quad R_{-\alpha} \circ R_\alpha = R_{-\alpha+\alpha} = R_0 = id$$

$$R_\alpha \circ R_{-\alpha} = id \quad \text{and} \quad R_{-\alpha} \circ R_\alpha = id$$

We have seen that the functions R_α and $R_{-\alpha}$ satisfy this relationship, so they are inverse functions. That is, $R_\alpha^{-1} = R_{-\alpha}$.

Rotations as Matrices We know what the rotation function $R_\alpha : \mathbb{R}^2 \to \mathbb{R}^2$ does to vectors written in polar coordinates. The formula is

$$R_\alpha \left(r(\cos(\theta), \sin(\theta))\right) = r(\cos(\theta + \alpha), \sin(\theta + \alpha)).$$

What's less clear is what the formula for R_α should be for vectors written in Cartesian coordinates. For example, what's $R_\alpha(3, 7)$? We'll answer this question below.

Theorem: $R_\alpha : \mathbb{R}^2 \to \mathbb{R}^2$ is linear and represented by the matrix:

$$R_\alpha = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{pmatrix}.$$

This follows by a straight forward calculation as

$$\begin{pmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{pmatrix} \begin{pmatrix} r\cos(\theta) \\ r\sin(\theta) \end{pmatrix} = \begin{pmatrix} r\cos(\alpha + \theta) \\ r\sin(\alpha + \theta) \end{pmatrix}.$$

One-Dimensional Representation of the $(k-1)$-Dimensional Random Walk Imagine a particle moving in a $(k-1)$-dimensional space, where at each step it can move along any of the $(k-1)$ independent directions. This movement can be visualized as a path on a $(k-1)$-dimensional grid, where each grid point represents a possible position of the particle. In financial mathematics, this kind of movement, known as a random walk, is often used to model the changing prices of multiple financial instruments over time. However, analyzing every possible path in $(k-1)$ dimensions can be computationally intensive and mathematically complex. To simplify this, the $(k-1)$-dimensional path can

be projected onto a single line, converting the problem into a one-dimensional random walk, which is easier to analyze and compute.

This projection is achieved using matrices [159]. The Q matrix is used to rotate and reflect the system such that one of the unit vectors coincides with a specific vector ι. This vector ι is chosen in such a way that the projection from $(k-1)$ dimensions to one produces a zero-mean random variable. The M vectors are then used to project the rotated system onto a line, essentially reducing the problem from a $(k-1)$-dimensional grid to a single line.

In the context of financial mathematics, particularly in modeling the prices of multiple financial instruments over time, a multidimensional random walk is often utilized. Imagine a particle moving in a $(k-1)$-dimensional space, where each point represents a possible state or value of a combination of financial instruments at a specific time. The Q matrix and M vectors are mathematical tools used to simplify the analysis of this movement. The Q matrix is an orthogonal matrix that rotates the system so that a specific vector, ι, is mapped to a unit vector, e_1. Mathematically, the elements of Q are defined as:

$$Q_{ij} = \begin{cases} \sqrt{p_i} & \text{if } j = 1, \\ \dfrac{\sqrt{p_j}}{\sqrt{1-p_1}} & \text{if } j = i+1, \\ 0 & \text{if } j > i+1, \\ -\dfrac{\sqrt{p_{j-1}p_i}}{\sqrt{(1-p_1)p_j}} & \text{if } j < i+1, \end{cases} \quad \text{for} \quad i,j = 1,2,\ldots,k+1.$$

The M vectors are derived from the Q matrix and are utilized to project the rotated system onto a line, converting the problem from a $(k-1)$-dimensional grid to a single line, which is computationally simpler to analyze. The M vectors are defined as the columns of the matrix

$$M = \tilde{M} \cdot \text{diag}\left(\frac{1}{\sqrt{p_1}}, \frac{1}{\sqrt{p_2}}, \ldots, \frac{1}{\sqrt{p_{k+1}}}\right),$$

where $\tilde{M}$ is derived from the Q matrix. This methodology allows for a simplified analysis while maintaining the essential characteristics of the original $(k-1)$-dimensional random walk.

Simplified Explanation of the Q Matrix Imagine you have a collection of arrows, which in mathematical terms we call *vectors*. These vectors point in various directions and each of them represents possible changes in the price of financial products, like stocks or options. The **Q matrix** is a special tool that helps us to rotate all these vectors together in a coordinated way, while ensuring that certain mathematical rules (related to probability) are respected.

In the context of the document, we can express this idea as follows:

- The vectors represent potential changes in financial prices.
- The Q matrix helps to rotate these vectors, making calculations related to pricing more manageable.
- The rotations performed by the Q matrix still adhere to the basic rules of probability.

In mathematical terms, the Q matrix is utilized to simplify and manage the complex calculations related to financial pricing by rotating vectors in a specific, rule-abiding manner.

Description and Computation of the Q Matrix The Q matrix is an orthogonal positive lower Hessenberg matrix used to transform the direction vectors. Its transpose $Q^\top$ is an orthogonal positive upper Hessenberg matrix. The first row of $Q^\top$ aligns with the requirement that $Q^\top$ is a rotation matrix that rotates the vector ι to the unit vector.

$$Q = \text{Gram-Schmidt}(\iota, \text{standard basis vectors})$$

Where:

- ι is a vector $\left(\sqrt{p_1}, \sqrt{p_2}, \ldots, \sqrt{p_{k+1}}\right)^\top$.
- The standard basis vectors in $\mathbb{R}^{k+1}$ are denoted as $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_{k+1}$.

Description:

Computation:

The Q matrix can be computed using the following steps:

Q	is an orthogonal positive lower Hessenberg matrix used to transform the direction vectors.
$Q^\top$	is an orthogonal positive upper Hessenberg matrix, which is the transpose of Q.
ι	is a vector composed of the square roots of the probabilities associated with each axis in the $(k+1)$-dimensional lattice.

1. Start with the vector ι and the standard basis vectors $\mathbf{v}_i$.
2. Apply the Gram-Schmidt process to orthogonalize these vectors:
   a. Initialize $u_1 = \iota$.
   b. For each subsequent vector $\mathbf{v}_i,$ subtract the projection of $\mathbf{v}_i$ onto each of the previously calculated orthogonal vectors u_j :
   $$u_i = \mathbf{v}_i - \sum_{j=1}^{i-1} \frac{\mathbf{v}_i \cdot u_j}{u_j \cdot u_j} u_j.$$
   c. Normalize the orthogonal vectors to produce orthonormal vectors:
   $$q_i = \frac{u_i}{\|u_i\|}.$$
3. Form the orthogonal matrix Q using the orthonormal vectors q_i as columns.

Example of Computing the Q Matrix Let's consider an example where we have equal probabilities $p_1 = p_2 = p_3 = \frac{1}{3}$, so that:

$$\iota = \begin{pmatrix} \sqrt{\frac{1}{3}} \\ \sqrt{\frac{1}{3}} \\ \sqrt{\frac{1}{3}} \end{pmatrix}$$

The standard basis vectors in $\mathbb{R}^3$ are:

$$\mathbf{v}_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \quad \mathbf{v}_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \quad \mathbf{v}_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}.$$

1. **Initialization**:

$$u_1 = \iota = \begin{pmatrix} \frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{3}} \end{pmatrix}$$

2. **Orthogonalization**:

For u_2:

$$\begin{aligned} u_2 &= \mathbf{v}_1 - \frac{\mathbf{v}_1 \cdot u_1}{u_1 \cdot u_1} u_1 \\ &= \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} - \begin{pmatrix} \frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{3}} \\ \frac{1}{1} \end{pmatrix} u_1 \\ &= \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} - \frac{1}{\sqrt{3}} \begin{pmatrix} \frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{3}} \end{pmatrix} \\ &= \begin{pmatrix} 1 - \frac{1}{3} \\ -\frac{1}{3} \\ -\frac{1}{3} \end{pmatrix} \\ &= \begin{pmatrix} \frac{2}{3} \\ -\frac{1}{3} \\ -\frac{1}{3} \end{pmatrix} \end{aligned}$$

Normalize u_2 to obtain q_2:

$$\|u_2\| = \sqrt{\left(\frac{2}{3}\right)^2 + 2\left(-\frac{1}{3}\right)^2} = \sqrt{\frac{4}{9} + \frac{2}{9}} = \sqrt{\frac{6}{9}} = \sqrt{\frac{2}{3}}$$

$$q_2 = \frac{u_2}{\|u_2\|} = \sqrt{\frac{3}{2}} \begin{pmatrix} \frac{2}{3} \\ -\frac{1}{3} \\ -\frac{1}{3} \end{pmatrix} = \begin{pmatrix} \sqrt{\frac{2}{3}} \\ -\frac{1}{\sqrt{6}} \\ -\frac{1}{\sqrt{6}} \end{pmatrix}$$

For u_3:

$$
\begin{aligned}
u_3 &= \mathbf{v}_2 - \sum_{j=1}^{2} \frac{\mathbf{v}_2 \cdot u_j}{u_j \cdot u_j} u_j \\
&= \mathbf{v}_2 - \left(\frac{\frac{1}{\sqrt{3}}}{\frac{1}{1}} u_1 + \frac{-\frac{1}{3}}{\frac{2}{3}} u_2 \right) \\
&= \mathbf{v}_2 - \left(\frac{1}{\sqrt{3}} u_1 - \frac{1}{2} u_2 \right)
\end{aligned}
$$

Compute the components:

$$
u_3 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} - \left[\frac{1}{\sqrt{3}} \begin{pmatrix} \frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{3}} \end{pmatrix} - \frac{1}{2} \begin{pmatrix} \frac{2}{3} \\ -\frac{1}{3} \\ -\frac{1}{3} \end{pmatrix} \right] = \begin{pmatrix} 0 \\ \frac{1}{2} \\ -\frac{1}{2} \end{pmatrix}
$$

Normalize u_3 to obtain q_3:

$$
\|u_3\| = \sqrt{\left(\frac{1}{2}\right)^2 + \left(-\frac{1}{2}\right)^2} = \sqrt{\frac{1}{4} + \frac{1}{4}} = \sqrt{\frac{1}{2}}
$$

$$
q_3 = \frac{u_3}{\|u_3\|} = \sqrt{2} \begin{pmatrix} 0 \\ \frac{1}{2} \\ -\frac{1}{2} \end{pmatrix} = \begin{pmatrix} 0 \\ \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{pmatrix}
$$

3. **Form the orthogonal matrix Q**:

$$
Q = \begin{pmatrix} q_1 & q_2 & q_3 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{3}} & \sqrt{\frac{2}{3}} & 0 \\ \frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{6}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{6}} & -\frac{1}{\sqrt{2}} \end{pmatrix}
$$

This example demonstrates the process of computing the Q matrix using the Gram-Schmidt orthogonalization process applied to the vector ι (composed of the square roots of the probabilities) and the standard basis vectors $\mathbf{v}_i$ in $\mathbb{R}^3$.

Description and Computation of the M Matrix from Q The M matrix is derived from the Q matrix to project the **standard basis vectors** in $(k+1)$-dimensional space onto a k-dimensional hyperplane.

$$M = Q \cdot \text{diag}\left(\frac{1}{\sqrt{p_1}}, \frac{1}{\sqrt{p_2}}, \ldots, \frac{1}{\sqrt{p_{k+1}}}\right)$$

Where:

- Q is the orthogonal matrix derived using the Gram-Schmidt process.
- $\text{diag}\left(\frac{1}{\sqrt{p_1}}, \frac{1}{\sqrt{p_2}}, \ldots, \frac{1}{\sqrt{p_{k+1}}}\right)$ is a diagonal matrix with elements $\frac{1}{\sqrt{p_i}}$.

Description:

Q	is an orthogonal matrix used to transform the standard basis vectors.
$\text{diag}\left(\frac{1}{\sqrt{p_1}}, \ldots, \frac{1}{\sqrt{p_{k+1}}}\right)$	is a diagonal matrix with elements $\frac{1}{\sqrt{p_i}}$.
M	is the resulting matrix derived from the multiplication of Q and the diagonal matrix.

Computation:

$$\begin{aligned} Q &= \text{Gram-Schmidt}(\iota, \text{ standard basis vectors}) \\ M &= Q \cdot \text{diag}\left(\frac{1}{\sqrt{p_1}}, \frac{1}{\sqrt{p_2}}, \ldots, \frac{1}{\sqrt{p_{k+1}}}\right) \end{aligned}$$

Description and Computation of the Direction Vectors d The direction vectors, denoted by d, represent the possible directions of the asset price movements. They are computed using a combination of the Cholesky decomposition of the covariance matrix, the M matrix derived from probabilities, and a modified drift term.

$$d_{i,j} = \exp\left(\sqrt{\delta t}\left((LM) \cdot i\right)_j + \hat{\mu}_j \delta t\right), \quad i = 1, 2, \ldots, k+1,\ j = 1, \ldots, k$$

Where:

- δt is a time increment.
- L is the lower triangular matrix obtained from the Cholesky decomposition of the covariance matrix Σ.

- M is the matrix derived from the probabilities p.
- $\hat{\mu}$ is a modified drift term [1].
- i indexes the direction vectors.

Description:

$d_{i,j}$	represents the j-th component of the i-th direction vector for the asset price movement.
L	is the lower triangular matrix obtained from the Cholesky decomposition of Σ.
M	is derived from the probabilities p and represents the impact of probabilities on the direction.
$\hat{\mu}$	is a modified drift term representing the expected return of the asset.

Computation:

$$
\begin{aligned}
L &= \text{Cholesky Decomposition}(\Sigma) \\
M &= \text{Matrix derived from probabilities } p \\
LM &= L \times M \\
d_{i,j} &= \exp\left(\sqrt{\delta t}\left((LM)\cdot i\right)_j + \hat{\mu}_j \delta t\right)
\end{aligned}
$$

4.2.4 Multinomial Tree Methodology

The methodological approach of the multinomial tree method encompasses determining transition probabilities, constructing the tree, computing payoffs at the terminal nodes (at option expiration), and iteratively calculating and discounting expected payoffs back through the tree to the present time. The flexibility of the multinomial tree allows for the incorporation of various conditions and constraints, such as early exercise features in American options or specific barriers in exotic options. The methodology can be generalized and adapted to cater to various types of options and underlying assets, providing a

[1] In [159] the drift rate is described as an integrable function representing the expected rate of change of the process in the Ito process formulation. It is modified for risk-neutral adjustments (e.g., replacing the real-world drift with the risk-free rate in financial modeling).

comprehensive framework that can be tailored to specific financial derivatives and market conditions.

Constructing a recombining binomial tree Imagine a tree structure where each node represents a possible price of an asset at a particular time step. Starting from the initial node, the asset price can either move up or down, branching out to form new nodes at each subsequent time step. This tree structure is constructed in such a way that it recombines, meaning that nodes at the same level (time step) and position (price) merge into a single node, thereby reducing computational complexity. Mathematically, the asset price at a node is modeled as:

$$Z_n = Z_0 \cdot \prod_{i=1}^{n} d_{Y_i},$$

where Z_n is the asset price at time step n, Z_0 is the initial asset price, d_{Y_i} represents a direction vector, and Y_i is a random variable determining the direction of the price movement (up or down). The direction vectors, d, are defined as:

$$d_{i,j} = \exp\left(\sqrt{\delta t}\left((LM)\cdot i\right)_j + \hat{\mu}_j \delta t\right)$$

where δt is the time step size, L and M are matrices derived from the asset's price dynamics, and $\hat{\mu}$ represents adjusted drift terms. This tree structure allows for the simulation of various price paths, which can be utilized to compute option prices by averaging the payoffs and discounting them back to the present value. The recombining nature of the binomial tree ensures computational efficiency by minimizing the number of nodes, and thus calculations, required to model the price evolution up to expiration.

Extending binomial trees to multidimensional lattices Imagine a tree where each node represents a possible price of an asset at a particular time step, and from each node, the price can either move up or down, forming a recombining tree structure. Extending this concept to multiple assets, we explore a $k+1$-dimensional lattice, which can be visualized as a multidimensional grid where each node represents a possible state of $k+1$ assets at a specific time step. The concept of Pascal's Simplex is introduced as a generalization of Pascal's Triangle to higher dimensions. In a similar manner to how Pascal's Triangle can be used to determine the coefficients of the binomial expansion, Pascal's Simplex provides the coefficients for the expansion of a multinomial, a polynomial with k terms. Mathematically, the Multinomial

Theorem states that:

$$(x_1 + x_2 + \ldots + x_k)^n = \sum_{n_1+n_2+\ldots+n_k=n} \binom{n}{n_1, n_2, \ldots, n_k} x_1^{n_1} x_2^{n_2} \ldots x_k^{n_k},$$

where the multinomial coefficients are given by

$$\binom{n}{n_1, n_2, \ldots, n_k} = \frac{n!}{n_1! n_2! \ldots n_k!}.$$

This $k+1$-dimensional lattice, constructed based on Pascal's Simplex, allows for the modeling of the joint evolution of $k+1$ assets in a recombining lattice, providing a computationally efficient framework for pricing options on multiple assets.

Computational Projection via M-vectors In financial modeling, particularly in the context of multi-asset option pricing, the concept of a random walk in a $k+1$-dimensional space is utilized, where each step represents a possible state of $k+1$ assets at a specific time step. To facilitate computational and analytical tractability, this $k+1$-dimensional random walk is projected onto a hyperplane of k-dimensions, which can be visualized as a k-dimensional grid. The M-vectors are derived to achieve this projection and are defined as the columns of the matrix $M = \tilde{M}.\text{diag}\left(\frac{1}{\sqrt{p_1}}, \frac{1}{\sqrt{p_2}}, \ldots, \frac{1}{\sqrt{p_{k+1}}}\right)$, where $\tilde{M}$ is derived from the Q matrix. The Q matrix is an orthogonal matrix that rotates the system in such a way that a specific vector, ι, is mapped to a unit vector, e_1. The elements of $\tilde{M}$ are defined as:

Definition 3.1 The elements of $\tilde{M}$ are given by:

$$\tilde{M}_{ij} = \begin{cases} \sqrt{\frac{\sum_{m=j+1}^{k+1} p_m}{\sum_{m=j}^{k+1} p_m}} & \text{if } i = j, \\ 0 & \text{if } i > j, \\ -\sqrt{\frac{p_i p_j}{\left(\sum_{m=i+1}^{k+1} p_m\right)\left(\sum_{m=i}^{k+1} p_m\right)}} & \text{if } i < j. \end{cases}$$

The M-vectors, derived from $\tilde{M}$, facilitate the projection of the $k+1$-dimensional random walk onto a k-dimensional hyperplane, enabling analysis of the joint evolution of $k+1$ assets in a recombining lattice.

Building a Recombining Multinomial Tree Imagine a tree structure where each node represents a possible state of multiple assets at a particular time step. From each node, the assets can move in multiple directions, each corresponding to a different combination of movements of the individual assets. This tree structure is constructed in such a way that it recombines, meaning that nodes at the same level (time step) and position (state) merge into a single node, thereby reducing computational complexity.

Mathematically, the asset prices at a node are modeled as:

$$Z_n = Z_0 \circ \prod_{i=1}^{n} d_{Y_i},$$

where:

- Z_n and Z_0 are vectors representing the asset prices at time step n and the initial asset prices, respectively.
- $\circ$ denotes element-wise multiplication of the vector components.
- d_{Y_i} represents a direction vector, and Y_i is a random variable determining the direction of the price movement.

Recall that the direction vectors were described in subsection 4.2.4.

4.3 Implementation Strategy

In this section we move further towards our FPGA option pricer. More specifically, we reconsider the binomial option pricing model for European options. We then examine how to extend this to two asset option pricing and then multi-asset option pricing. Once we understand this, we look in more detail as to how the recursive nature of the multi-nomial option pricing can be implemented which necessities a certain type of ordering scheme to ensure we can implement the parallelism we desire. Finally we look at optimization and computation of certain parameters needed for our computation.

4.3.1 Binomial Option Pricing Model for European Options

The binomial option pricing model is a simple yet powerful approach to option pricing, which divides time into discrete intervals and assumes that the price

of the underlying asset can only move up or down by a specific factor per period. This model, pseudocode presented in Algorithm 1, is based on constructing a recombining binomial tree, representing possible paths the underlying asset's price can take over the option's life.

Algorithm 1 Pseudocode for Binomial Model

```
Initialize binomial tree parameters: up factor (u), down factor (d), and risk-free rate (r)
Construct binomial tree for underlying asset's price movements
for each final node at expiration do
    Calculate option value using payoff function
end for
for each preceding node do
    Calculate option value by discounting expected value from following nodes
end for
```

4.3.2 Extension to Multinomial Option Pricing

The multinomial option pricing model extends the binomial framework to accommodate multiple assets and their interactions, utilizing Pascal's simplex for higher-dimensional representation (pseudocode presented in Algorithm 2). This model allows for a more comprehensive analysis of options on multiple underlying assets, capturing the complex price dynamics and correlations.

Algorithm 2 Pseudocode for Multinomial Model

```
Define dimensions based on the number of assets
Initialize multinomial tree parameters, including correlations among assets
Construct the multinomial tree using Pascal's simplex, accounting for correlations
for each final node at expiration do
    Calculate the option value using the payoff function
end for
for each preceding node do
    Compute the option value by discounting the expected value from following nodes
end for
```

The transition from binomial to multinomial option pricing models involves moving from a one-dimensional framework to a multi-dimensional one, capturing the complexities of multiple underlying assets. This implementation

strategy outlines the construction of both models, emphasizing the scalability and adaptability of the multinomial model for complex option pricing tasks. The paper by Hanzon and Sierag states **"By allocating the data in memory in an efficient way the computation time can be reduced."** which this research implements using First-In First-Out (FIFO) queues.

4.3.3 Simplified multiasset option pricing

In this subsection we briefly introduce a number of important features of our implementation, specifically in the management of values.

Algorithm 3 Option Pricing Process with FIFOs

```
1: Setup:
2: Initialize components and data structures
3: Read configuration and set parameters
4: Preparation:
5: Compute vectors and matrices for option pricing
6: Initialize FIFOs for data management
7: Main Loop:
8: while there are elements in FIFOs do
9:     Retrieve data from FIFOs
10:    Calculate option values based on current data
11:    Update FIFOs with new calculations
12: end while
13: Finalization:
14: Display calculated option prices
15: Release resources and clean up
```

4.3.3.1 Use of FIFOs

Using FIFOs to store intermediate option pricing values and employing monomial ordering optimizes data flow and resource utilization in FPGA computations. Algorithm 3 shows the pseudocode for this. The first FIFO being the deepest ensures that initial, more complex computations have ample storage, while subsequent FIFOs, decreasing in depth, align with reduced computational requirements as the process advances. Monomial ordering facilitates organized data retrieval, crucial for maintaining computational consistency and efficiency in option pricing models, particularly when scaling to multiple options.

Monomial ordering is a system used to arrange monomials (single terms in a

polynomial) in a specific sequence. This ordering is crucial in computational models, including those implemented on FPGAs for tasks such as option pricing.

Examples of Monomial Ordering:

1. *Lexicographic Ordering* (lex): Monomials are ordered based on the alphabetical order of the variables. For instance, for variables x and y, $x^2 > xy > y^2$ in lex ordering because x is considered before y.

2. *Graded Lexicographic Ordering* (grlex): Monomials are first compared by their total degree, and ties are broken using lexicographic order. For example, $x^2y^2 > x^3 > xy$ in grlex because the total degree of x^2y^2 (which is 4) is greater than that of x^3 (which is 3), even though x^3 would precede x^2y^2 in lex order.

Equations:

Consider a polynomial $f(x, y) = 3x^3y + 4xy^2 + 5y^3 + 6x^2$. The monomial ordering will dictate the sequence of these terms:

- In *lex* order: $3x^3y > 6x^2 > 4xy^2 > 5y^3$
- In *grlex* order: $3x^3y > 4xy^2 > 5y^3 > 6x^2$

This ordering affects how computations are structured and optimized, particularly in data storage and retrieval processes in computational models.

Consider option pricing based on two underlying asset moves, X and Y. We use FIFOs to store intermediate values, with monomial ordering $X^2, XY, Y^2, X, Y, 1$. The depth of FIFOs, i.e, the number of values it can store, decreases as the complexity of terms decreases.

This ordering and varying of FIFO depths ensures efficient utilization of storage and computational resources, aligning with the progressive simplification of computations in option pricing models.

Consider a two-stage option pricing example with assets X and Y. Each monomial (e.g., X^2, XY, Y^2) corresponds to a node in the tree, representing the values associated with different combinations of movements in the assets.

Stage 1: Tree Structure and Node Values

In the binomial tree, each node corresponds to a specific monomial (e.g., X^2, XY, Y^2), and their associated values are stored in FIFOs. For example:

- Node X^2 (FIFO$_{X^2}$) stores values: $100, 110, 120$.

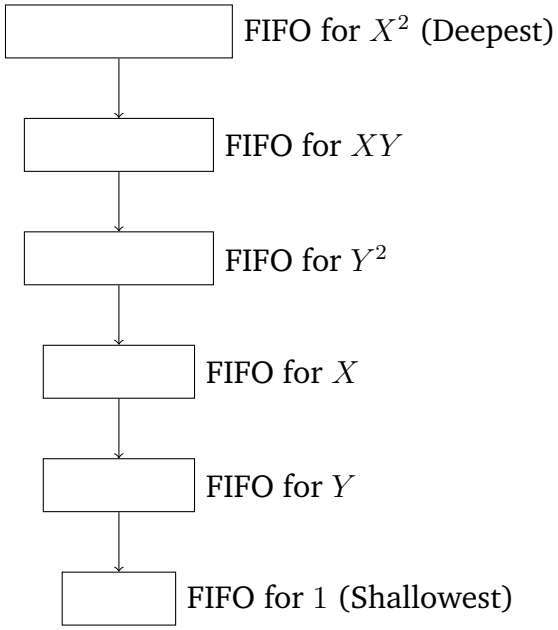


Figure 4.3: Visualization of FIFO Queues for Different Variables

- Node XY (FIFO$_{XY}$) stores values: $110, 121, 132$.
- Node Y^2 (FIFO$_{Y^2}$) stores values: $120, 132, 144$.

These values are computed based on the specific directions of asset movements and stored systematically for use in subsequent computations in later stages.

Stage 2: (Using values from Stage 1)

At this stage, the values stored in the FIFOs from Stage 1 are combined to compute new values for X and Y, corresponding to the next level in the tree. The computations are performed as follows:

$$\text{FIFO}_X : \frac{100+110}{2}, \frac{121+132}{2} \quad (\text{computed from FIFO}_{X^2} \text{ and FIFO}_{XY})$$
$$\text{FIFO}_Y : \frac{120+110}{2}, \frac{144+132}{2} \quad (\text{computed from FIFO}_{Y^2} \text{ and FIFO}_{XY})$$

The resulting FIFOs for Stage 2 are:

$$\text{FIFO}_X : 105, 126.5$$

$$\text{FIFO}_Y : 115, 138$$

These values represent the aggregated results at this stage and are used for further computations in subsequent levels of the multinomial tree.

This example demonstrates the use of FIFOs to store and retrieve intermediate values in a structured manner, facilitating efficient computations across multiple stages. The values at each node in the binomial tree are calculated based on the values at its successor nodes, which are stored in the FIFO structure.

Multinomial trees extend the binomial tree framework by accommodating multiple assets, where each node corresponds to a state of all assets. This approach becomes particularly important as the number of assets increases, as multinomial trees mitigate the rapid combinatorial growth in computational complexity by recombining nodes at each level.

4.3.4 Recursive Factor Adjustment Method

The Recursive Factor Adjustment Method is a systematic approach used to compute stock values in binomial and multinomial tree models efficiently. This method leverages recursive calculations and the adjustment of state transition factors to simplify the computation process. We present this concept for the binomial model for simplicity.

4.3.4.1 Concept Overview

The method starts with an initial stock price S_0 and recursively applies up (u) and down (d) factors to compute the values at each node of the tree. This approach avoids the need to compute each node's value individually from scratch, making the process more efficient.

4.3.4.2 Steps Involved

1. **Initialization:**
    - Start with the initial stock price S_0.
    - Define the up factor u and down factor d, where $u > 1$ and $0 < d < 1$.
2. **Recursive Computation:**
    - At each step in the tree, adjust the stock price by multiplying it with the appropriate factors.

- This adjustment is done recursively, meaning the process is repeated at each level of the tree.

3. **Combination of Factors:**
   - Combine the factors in a specific order to reflect the possible movements in stock price.
   - For example, in a binomial tree, you would combine the up and down factors to compute the stock price at each node.
4. **Efficient Calculation:**
   - By using recursive adjustments, the method avoids the need to compute each node's value individually from scratch.
   - Instead, it builds upon the previous computations, making the process more efficient.

4.3.4.3 Example Calculation

Consider a binomial tree where the stock price can move up or down at each step. The initial stock price is S_0, and the stock price at each node is calculated based on the previous step:

- For an upward movement, $S_u^N = S_u^{N-1} \cdot u$.
- For a downward movement, $S_d^N = S_d^{N-1} \cdot d$.

For example, starting with S_0:

$$\begin{aligned} S_u^1 &= S_0 \cdot u, \quad S_d^1 = S_0 \cdot d, \\ S_u^2 &= S_u^1 \cdot u = S_0 \cdot u^2, \quad S_d^2 = S_d^1 \cdot d = S_0 \cdot d^2, \\ S_{ud}^2 &= S_u^1 \cdot d = S_0 \cdot u \cdot d. \end{aligned}$$

At any step N, the stock price after i upward movements and $N - i$ downward movements is:

$$S_{i,N} = S_0 \cdot u^i \cdot d^{N-i}.$$

4.3.5 Monomial Ordering and FIFOs

This method efficiently organizes computations by combining monomial ordering with FIFO (First-In-First-Out) queues to store intermediate values. The

FIFO queues store the **asset prices** (not option values) at each node in the multinomial or binomial tree. Each monomial corresponds to a valid node in the tree, enabling structured data storage and retrieval during computations.

Link to the Tree Structure: In a multinomial or binomial tree, each node represents a valid monomial, corresponding to a specific combination of movements in the underlying assets. For example:

- At the terminal layer (payoff layer), each node corresponds to the payoff associated with a monomial representing the final state of the assets.
- Intermediate nodes correspond to combinations of upward and downward movements, represented by monomials like $X^2, XY, Y^2, \ldots$, which are **valid nodes** in the tree.

Monomial ordering ensures that these nodes are processed systematically, while FIFOs store the intermediate values needed to backstep through the tree. This backstepping allows recursive calculations from terminal nodes to the root.

Recursive Factor Adjustment Method: The Recursive Factor Adjustment Method organizes intermediate values using FIFOs at each layer of the tree. At the terminal layer, the FIFOs store asset prices corresponding to valid monomials. These values are then recursively combined at preceding layers to compute intermediate asset prices. For example:

- At each node, the FIFO contains the results of computations at successor nodes. These values are combined using ratios of u (upward movement factor) and d (downward movement factor).
- Deeper FIFOs, like for XY, store values that are reused in multiple computations, demonstrating the necessity for structured data storage.

Why FIFOs? FIFOs ensure that intermediate asset prices are easily accessible for multiple computations without recalculating them. Deeper FIFOs are critical at intermediate layers where values are reused in various combinations, as in the case of XY contributing to both X and Y calculations.

4.3.6 Optimization of Input Parameters

In the context of implementing the multinomial tree method on FPGA using OpenCL and OneAPI, significant performance improvements can be achieved by optimizing input parameters to be as static as possible. This approach involves precomputing and fixing values that do not change during execution, thereby reducing runtime computations and enhancing efficiency.

Precomputation Strategy Key parameters and constants involved in option pricing, such as risk-free rates, volatility, and initial asset prices, are evaluated and fixed at initialisation-time. By precomputing these static values, the need for dynamic computation is minimized, leading to a streamlined execution path on the FPGA.

Static Data Structures Data structures, particularly those pertaining to the multinomial tree, such as transition probabilities and asset price nodes, are initialized and populated with precomputed values before kernel execution begins. This does not imply that all nodes of the multinomial tree are precomputed; rather, key nodes are initialized to ensure an efficient starting point for recursive computations during kernel execution. This strategy ensures that memory access patterns are predictable and optimized for the FPGA's architecture, enabling faster execution.

Kernel Optimization The OpenCL kernels are tailored to leverage the static nature of the input parameters. Conditional logic and branching operations, which are expensive in FPGA pipelines, are minimized by restructuring computations to rely on precomputed values wherever possible. The use of OneAPI facilitates the fine-tuning of these kernels, ensuring they are highly optimized for the hardware and exploit the FPGA's resources efficiently.

Memory Management With static parameters, memory allocation and data transfer between the host and FPGA are optimized, ensuring that data is readily available for the compute units. This reduces the overhead associated with data movement and increases the overall throughput of the application. The memory hierarchy is carefully managed to ensure low-latency access for frequently reused values, while less critical data is staged appropriately.

Summary By adopting a strategy that maximizes the use of static input parameters, the implementation not only becomes more efficient but also better suited to the unique characteristics of FPGAs. This optimization plays a crucial role in achieving high performance and reliability in financial computing applications such as option pricing.

4.3.7 Computation of d and Q

In Section 4.2.4, we introduced the d-matrix and Q-matrix. The computation of d and Q (called `compute_d_and_q` in code) is instrumental in the multinomial tree method for option pricing, primarily due to its role in precomputing the values needed to simulate the complex dynamics of financial markets. The function computes the direction vectors and the orthogonal positive lower Hessenberg matrix, which are essential for accurately modeling the price movements of underlying assets.

Simulating Price Dynamics The direction vectors, computed by the function, represent the potential steps along paths that the price of an asset can take over time. This capability is crucial for option pricing, where the payoff depends on the path and final value of the underlying asset. By capturing the essence of market volatility and the correlation between different assets, these vectors allow for a more nuanced and realistic simulation of price movements.

Enhancing Computational Efficiency The Q-matrix, computed by the `compute_d_and_q` function, is an orthogonal positive lower Hessenberg matrix. A critical component of this matrix is the Q_{vector}, which consists of the last k rows of Q. These rows form the basis of the k-dimensional hyperplane orthogonal to the vector $\iota = (\sqrt{p_1}, \sqrt{p_2}, \ldots, \sqrt{p_{k+1}})$. The Q_{vector} is then used to construct the M-matrix, which maps the random walk's movement from $(k+1)$-dimensional space to k-dimensional space. This precomputation of d-matrix and Q_{vector} significantly reduces runtime computations, particularly in FPGA implementations, where computational efficiency and speed are critical.

Facilitating Accurate Valuations Accuracy in option pricing is of high importance, given the sensitivity of option values to the underlying asset's price movements. The direction vectors in the d-matrix, combined with the transformation provided by Q and Q_{vector}, ensure that price movements in the multinomial tree closely reflect real-world financial dynamics. This alignment en-

hances the reliability of option prices, making the function an indispensable component of the option pricing model.

Method:

1. *Initialization*: Allocates memory for various matrices and computes the vector $\iota = (\sqrt{p_1}, \sqrt{p_2}, \ldots, \sqrt{p_{k+1}})$ and the modified drift term.
2. *Cholesky Decomposition*: Decomposes the 'sigma_matrix' using Cholesky decomposition to extract covariance structures.
3. *Direction Vectors*: Constructs the d-matrix, representing possible asset price movements, as described in earlier sections.
4. *Orthogonal Positive Lower Hessenberg Matrix*: Constructs the Q-matrix using the Householder transformation (described in Section 4.3.7.2). The last k rows of Q (referred to as Q_{vector}) are extracted for further use.
5. *Second Moment Matching*: Optionally refines the d-matrix and Q_{vector} using a second moment matching technique to align with statistical properties (described in Section 4.3.7.1).

4.3.7.1 Second Moment Matching

Second moment matching is a statistical technique employed to approximate a complex probability distribution with a simpler one, such as a normal distribution, by ensuring that the first two moments—mean and variance (or covariance matrix for multivariate cases)—are equivalent. It is important to distinguish between the second moment (which includes both variance and covariance) and the variance itself, as the second moment encompasses the spread and relationships between variables in multivariate distributions.

Application in Multivariate Option Pricing In financial mathematics, the normal distribution is commonly used to model the distribution of log-asset prices under the assumptions of geometric Brownian motion. However, real-world asset returns often deviate from this idealized model, exhibiting more complex behavior. Second moment matching allows us to approximate these complex behaviors in a multinomial tree by matching the statistical properties of the distribution used in the tree to those of the continuous-time geometric Brownian motion model. In this context, the "simpler distribution" corresponds to the normal distribution implied by the additive structure of the

multinomial tree.

By matching the mean vector and covariance matrix, the multinomial tree reflects the central tendency, variability, and interdependencies between assets, ensuring that the discrete model remains consistent with the continuous dynamics of geometric Brownian motion.

Procedure The process of second moment matching involves several key steps for multivariate distributions:

1. **Identify the Complex Multivariate Distribution:** Determine the complex probability distribution that accurately describes the multivariate random variables of interest, such as the joint returns of multiple financial assets. This distribution may be derived from historical data or theoretical models.
2. **Choose a Simpler Multivariate Distribution:** A multivariate normal distribution is often chosen due to its mathematical tractability and its compatibility with financial models that assume normally distributed log-returns. The multivariate normal distribution is fully described by its mean vector and covariance matrix.
3. **Match the First Moment:** Calculate the mean vector of the complex distribution and adjust the mean vector of the simpler distribution to match, ensuring alignment in the central tendency of all variables.
4. **Match the Second Moment:** Compute the covariance matrix of the complex distribution and adjust the covariance matrix of the simpler distribution to match. This step ensures that the spread and correlations between variables are preserved in the simpler distribution.

Relevance to This Work In the context of this thesis, second moment matching is utilized to align the multinomial tree with the statistical properties of geometric Brownian motion. Specifically:

- The complex distribution corresponds to the distribution of the underlying assets' joint returns.
- The simpler distribution corresponds to the normal distribution implied by the multinomial tree's additive structure.

This method ensures that the tree accurately reflects the mean, variance, and

covariance of the underlying process, providing a robust basis for option pricing. By doing so, the discrete multinomial tree remains consistent with the continuous-time dynamics of financial models.

Key Warning It is critical to distinguish between the second moment (which includes variance and covariance) and variance alone. This distinction becomes especially relevant when dealing with multivariate distributions, as the second moment captures relationships between variables in addition to individual variability.

By matching the mean and covariance, the simpler multivariate normal distribution approximates the original distribution's central tendency and variability. While this method does not account for higher moments such as skewness and kurtosis, it provides sufficient accuracy for practical applications, including option pricing and risk management.

4.3.7.2 Construction of the Q Vector using Householder Transformation

Householder Transformation: The Householder transformation is a method used to transform a given matrix into a tridiagonal or Hessenberg form. It achieves this by constructing a series of orthogonal matrices, which when applied successively to the original matrix, zeroes out elements below the diagonal.

Given a vector $\mathbf{v}$, the Householder matrix $\mathbf{P}$ is defined as:

$$\mathbf{P} = \mathbf{I} - 2\mathbf{v}\mathbf{v}^T$$

where $\mathbf{I}$ is the identity matrix.

Application in the Function: In the context of the 'compute_d_and_q' function, the Householder transformation is applied to the 'd_matrix' to construct the 'Q' vector. The transformation ensures that the 'Q' vector is an orthogonal positive lower Hessenberg matrix. Specifically, the Q-vector is derived from the last k rows of the matrix after all Householder transformations.

The steps involved are:

1. For each column of the matrix A (referred to as the 'd_matrix' in the code), compute the Householder vector $\mathbf{v}$ that introduces zeros below the diagonal.

2. Construct the Householder matrix P using the formula:

$$P = I - 2\frac{\mathbf{v}\mathbf{v}^\top}{\mathbf{v}^\top\mathbf{v}},$$

   where $\mathbf{v}$ is the Householder vector.

3. Apply the Householder matrix iteratively to matrix A (from left to right) to zero out the elements below the diagonal. The Householder transformation is a well-established method for QR decomposition and Hessenberg transformations in numerical linear algebra. For a detailed explanation of the recursive application of the Householder transformation, see [160].

4. The final matrix after all transformations contains the orthogonal positive lower Hessenberg structure, and the last k rows are extracted to form the Q-vector.

This process ensures that the direction vectors encoded in the 'd_matrix' are systematically orthogonalized and structured for subsequent use in recombining multinomial trees.

The function 'linalg_HH_svx_gen' encapsulates this process. It first performs the Householder transformation on the matrix 'A' to transform it into a Hessenberg form. The matrix A contains the direction vectors ($d_{i,j}$) that encode the potential price movements in the multinomial tree. Each column of A corresponds to a step in the price path for the underlying assets. The Householder transformation is applied to A to enforce orthogonality and create the lower Hessenberg structure of the Q-matrix. Then, it performs back-substitution to solve the linear system, effectively computing the 'Q' vector. The first column of A corresponds to the probabilities (p_1, p_2, p_3), which define the weights for the transitions in the multinomial tree. Subsequent columns contain scaled direction vectors that ensure the matrix aligns with the risk-neutral framework. The linear system solved during back-substitution relates to the orthogonalization of the d-matrix, ensuring that the resulting Q-matrix (recall, we do not use the full matrix, only a vector of the matrix) satisfies the required orthogonal and Hessenberg properties. This step ensures the transformed matrix retains numerical stability for subsequent computations.

Table 4.1: OpenCL kernel inputs. These inputs are precomputed from the traditional option price parameters, i.e., ρ, σ etc.

Kernel	**Description**
N	Tree depth
max_min	Whether to call/put on max or min (boolean value)[a]
end_node_values	defined as a 2-element vector containing $\left\{(d_{0,0})^N Z_0, (d_{1,0})^N Z_1\right\}$ where the two assets[b] are multiplied by the first two elements in the direction vector to the power of N.
init_steps	defined as a 2x2 element matrix containing $\left\{\frac{d_{1,2}}{d_{1,1}}, \frac{d_{1,1}}{d_{1,0}}\right\}, \left\{\frac{d_{0,2}}{d_{0,1}}, \frac{d_{0,1}}{d_{0,0}}\right\}$
mult_coeff	3-element multiplicative coefficient vector containing $(1/R) * q_2$,$(1/R) * q_1$,$(1/R) * q_0$[c] where R is defined as $e^{r*(1/N)}$ (r is the risk neutral rate)
share_coeff	2-element share coefficient vector containing $\left\{\frac{1}{d_{1,1}}, \frac{1}{d_{0,1}}\right\}$
strike_vector	2-element vector containing the strike prices K.

[a] These inputs refer to pricing options in a multinomial tree framework, specifically for assets where the option payoff depends on the maximum or minimum value of the underlying assets. These serve as benchmarks in this research due to their well-studied theoretical models, allowing for robust validation of computational approaches.

[b] This table illustrates inputs for the 2-asset case, modeled using a Pascal tree. The 2-asset framework simplifies the extension to higher-dimensional trees while maintaining computational feasibility.

[c] Here, q_0, q_1, q_2 represent risk-neutral probabilities associated with transitions in the multinomial tree. These probabilities satisfy the condition $q_0 + q_1 + q_2 = 1$, ensuring that the expected value of the discounted payoff remains neutral under the risk-neutral measure.

4.3.8 Data Structures and Storage

We create data structures to encapsulate and abstract the required variable that we pass from host to device. Table 4.1 presents these data structures and inputs to our OpenCL implementation and we describe them in detail in this subsection.

4.3.8.1 Init_steps parameter

In the context of an arbitrage-free financial market, the concept of direction vectors, denoted as $d_{i,j}$, is critical for computing replicating portfolios and, consequently, the value of financial derivatives. These direction vectors are defined in a multinomial model, particularly when:

$$\mu_j = r, \quad j = 0, 1, \ldots, k-1,$$

where μ_j is the drift term (set to the risk-free rate r) and are associated with a probability vector:

$$p = (p_0, p_1, \ldots, p_k).$$

The real numbers λ_j, for $j = 0, 1, \ldots, k-1$, are defined as:

$$\sum_{i=0}^{k} p_i d_{i,j} e^{-\lambda_j \delta t} = e^{-r\delta t}, \quad j = 0, 1, \ldots, k-1.$$

ensuring the model maintains no arbitrage. This constraint allows us to compute replicating portfolios by solving a linear system of equations at each node.

The bias-corrected direction vectors, $\tilde{d}_{i,j}$, for $i = 0, 1, \ldots, k$, are then defined as:

$$\tilde{d}_i(j) = d_i(j) e^{\lambda_j r \delta t}, \quad j = 0, 1, \ldots, k-1.$$

This bias correction ensures consistency with the underlying financial model by accounting for adjustments in the drift term. For a detailed explanation of this bias correction, refer to [159].

In the implementation, the 'init_step' vector is formulated as:

$$\text{init_step}[i][j] = \frac{d[i][j]}{d[i-1][j]},$$

where i and j iterate over the dimensions and steps, respectively, in the multinomial model, starting at 0. The 'init_step' vector captures the relative change

in the direction vectors between two consecutive steps, providing a mechanism to update stock prices in the tree. These direction vectors do not change over time but rather represent fixed ratios derived from the multinomial tree structure.

This vector is utilized in the OpenCL kernel to adjust stock prices as the tree evolves. By maintaining consistency with the financial model, the 'init_step' matrix ensures accurate representation of stock price movements, adhering to the specified direction vectors. This construction enables the tree structure to evolve in a manner consistent with the underlying theoretical assumptions.

Example: To clarify the role of the 'init_step' matrix, consider a 2-asset multinomial tree. The direction vectors could represent movements as follows:

$$d_{0,0} = \frac{x}{y}, \quad d_{0,1} = \frac{x^2}{y}, \quad d_{1,0} = \frac{x}{y^2}, \quad d_{1,1} = \frac{x^2}{y^2}.$$

In this case:

$$\text{init_step}[0][0] = \frac{d_{0,0}}{d_{0,0}}, \quad \text{init_step}[0][1] = \frac{d_{0,1}}{d_{0,0}}, \quad \text{init_step}[1][0] = \frac{d_{1,0}}{d_{0,0}}.$$

In the computational kernel 'stockPrice', the matrix 'init_step' is instrumental in determining the evolution of stock prices through the multinomial tree. Each element init_step$[i][j]$ signifies a multiplier for the stock price of the i-th asset, simulating a movement in the j-th direction in the asset price space. Throughout the tree traversal, the stock prices are iteratively updated by multiplying them with the corresponding values from the 'init_step' matrix, thereby generating new potential end prices for the assets.

This multiplication is executed in a manner that either simulates a "sideways" movement, updating the asset prices without advancing to a subsequent time step, or propels the asset prices to the next time step in the model. Notably, the algorithm is designed to efficiently compute only the terminal nodes of the tree, representing the possible final asset prices at the option's expiration, without explicitly calculating the entire tree. These terminal nodes are then employed in subsequent calculations to determine the option's payoff and, consequently, its present value in the financial market model.

Clarifications on the Model's Scope While this explanation is valid for European options, the statement regarding "terminal nodes" does not apply to American options or knock-out options, where intermediate nodes are required for early exercise and barrier conditions. For these cases, additional computations are necessary at each intermediate node to evaluate potential option payoffs.

Key Clarifications: 1. The direction vectors $d_{i,j}$ represent relative movements in asset prices and are fixed throughout the computation. The 'init_step' vector captures their relative changes step-by-step, as required for multinomial tree traversal. 2. The replicating portfolios are computed using the $d_{i,j}$ vectors and their associated probabilities, ensuring alignment with arbitrage-free pricing principles.

4.3.8.2 Mult_Coeff Parameter

The `mult_coeff` vector is computed by iterating through each dimension, indexed by j, from `num_dim = 0`. For each j, `mult_coeff[j]` is calculated as:

$$\texttt{mult_coeff}[j] = \frac{1}{\texttt{conf_set->EXP_RATE}} \times \texttt{conf_set->eu_prob_vect}[j],$$

where: - `conf_set->EXP_RATE` is the exponential growth rate R, defined as $R = e^{r \times \frac{T}{N}}$, where r is the risk-neutral rate, T is the runtime (or time-to-maturity of the option), and N is the tree depth. - `conf_set->eu_prob_vect` represents the risk-neutral probabilities (q_2, q_1, q_0) associated with each dimension of the multinomial tree.

Role of Mult_Coeff This vector plays a pivotal role in ensuring that the model adheres to the principles of risk-neutral valuation by appropriately weighting the possible movements in the multinomial tree. By combining the growth rate and risk-neutral probabilities, `mult_coeff` ensures accurate computation of the option value at each node.

Expected Value vs Payout The term "expected payout" is replaced with "expected value" to better reflect the computation at each node. Specifically, the

model calculates the expected value of the financial derivative at terminal nodes or intermediate nodes, depending on the type of option being priced.

Use in Parallel Computation In the function `computeValue`, the vector `mult_coeff` is used in a parallel computation framework. This is typical in FPGA implementations, where data is processed through multiple parallel channels (e.g., `euPricerInChannel_0, euPricerInChannel_1, etc.`). Each channel handles a subset of computations, such as determining the expected value of a node by summing the weighted products of subsequent states and the corresponding multipliers in `mult_coeff`.

For example:

$$\texttt{sumP} = \sum(\texttt{payout}_{\text{next state}} \times \texttt{mult_coeff}).$$

Here, `sumP` represents the aggregated expected value for the current node, which is subsequently written to an output channel for further processing.

FIFO and Memory Management The number of values available at each computation step is carefully managed using FIFO (First-In-First-Out) buffers. Due to the multinomial tree structure, only the values from adjacent nodes in the previous layer are required for the current node's computation. This ensures minimal memory usage and efficient data retrieval. Additionally, care must be taken to ensure the correct number of values is available at each computation step, particularly for intermediate nodes.

Differentiating the Channels Each channel corresponds to a specific computation path in the parallelized implementation. These channels handle distinct subsets of the multinomial tree, ensuring scalability and efficient use of FPGA resources. A detailed explanation of the role of these channels, the data they process, and their synchronization mechanism would warrant a dedicated subsection to provide clarity.

4.3.8.3 Computation of the `mult_coeff` Using `eu_prob_vect`

After normalizing the `q_vector` and storing the results in `conf_set[index]->eu_prob_vect`, the code proceeds to compute the multiplier coefficients for an object `opp`.

Given:

- $q[i]$: Elements of the q-vector.
- `EXP_RATE`: Constant value from `conf_set->EXP_RATE`, defined as $e^{r\Delta t}$, where r is the risk-free rate and Δt is the time step.
- `eu_prob_vect`[j]: Normalized values of the q-vector, computed as:

$$\text{eu_prob_vect}[j] = \frac{q[j]}{\sum_{k=0}^{\text{num_dim}} q[k]}.$$

The `mult_coeff` for each index j is computed as:

$$\text{mult_coeff}[j] = \frac{\text{eu_prob_vect}[j]}{\text{EXP_RATE}}.$$

Computing the End of the Tree For a multinomial tree, the stock price at the terminal nodes after n periods is calculated based on the direction vectors and the multinomial tree structure. In this context:

- S: Initial stock price.
- $d_{i,j}$: Direction vector that governs the movement in dimension j at step i.
- n: Number of time steps (tree depth).

The stock price at a terminal node is given by:

$$S_{i,j} = S_0 \prod_{k=1}^{n} d_{k,j},$$

where $d_{k,j}$ represents the factor by which the stock price moves in the j-th direction at step k. This formula generalizes the binomial tree approach, which uses the simpler u and d factors.

Working Backwards Up the Tree When working backwards through the multinomial tree for a European option, the expected value at each node is computed by discounting the future values from its successor nodes to the present. The weighting is determined by the q-vector probabilities, which are associated with the direction vectors.

For a multinomial tree, the expected value at a node is calculated as:

$$\text{Value at node} = \frac{1}{e^{r\Delta t}} \sum_{j=1}^{k+1} (\text{Value at successor node} \times q[j]) \,,$$

where:

- $q[j]$: Risk-neutral probability associated with the j-th direction.
- r: Risk-free rate.
- Δt: Time step.

For clarity, this differs from the binomial case, where the option price at a previous node is computed using:

$$C = \frac{qC_u + (1-q)C_d}{e^{r\Delta t}},$$

with $q = \frac{e^{r\Delta t} - d}{u - d}$. The multinomial framework extends this by considering multiple dimensions and their respective probabilities.

Risk-Neutral Probabilities in Multinomial Trees The q-vector probabilities in a multinomial tree are generalized to represent the likelihood of movement in each direction. These probabilities are normalized such that:

$$\sum_{j=1}^{k+1} q[j] = 1.$$

This ensures consistency with the risk-neutral valuation framework while accommodating the increased complexity of multinomial branching.

4.4 Code Implementation

In this section we look at the implementation of the option pricer. More specifically, we look at the code structure, the functionality that code provides and how optimizations for the FPGA are delivered.

4.4.1 Code Structure and Design

Host side Pseudocode

The functions listed below are designed to run on the host PC that the FPGA is attached to.

1. Function: `initFPGA` **Purpose**: Initializes the FPGA environment for option pricing.

Step 1: Find and set the OpenCL platform for Intel.

Step 2: Get devices and select the first available device.

Step 3: Create an OpenCL context and command queue.

Step 4: Load and build the binary file for the FPGA.

2. Function: `doKernel` **Purpose**: Executes the main kernel for option pricing on the FPGA.

Step 1: Allocate memory buffers on the device for inputs and outputs.

Step 2: Set kernel arguments (e.g., initial steps, strike price, option type).

Step 3: Transfer data from host to device buffers.

Step 4: Execute the kernel using `clEnqueueTask`.

Step 5: Read back the computed option value to the host.

3. Function: `init_fifos` **Purpose**: Initializes FIFOs for shares, values, and partitions.

Step 1: Allocate and initialize share values and partitions.
Explanation: Each partition corresponds to a node in the multinomial tree at the initial time step. These partitions are critical for organizing the tree traversal and ensuring efficient computation of the option pricing.

Step 2: Compute initial share values based on the model parameters.
Explanation: The share values are derived from the initial conditions of the financial model and represent the starting point for computations in the tree structure.

Step 3: Push initial values and partitions into their respective FIFOs.
Explanation: This step ensures that each node in the tree has the required data available for further computations as the tree evolves.

4. Function: `compute_d_and_q` **Purpose**: Computes direction vectors (`d_matrix`) and the Q matrix (`q_vector`).

Step 1: Apply Cholesky decomposition to the Σ matrix.
Explanation: The Cholesky decomposition is performed on the covariance matrix (`sigma_matrix`), and its lower triangular part is extracted. This decomposition is essential for calculating the direction vectors.

Step 2: Calculate direction vectors using the decomposed matrix and model parameters.
Explanation: Direction vectors (`d_matrix`) are computed using the decomposed matrix and drift terms. These vectors represent possible paths in the multinomial tree structure.

Step 3: Construct the Q matrix using Householder transformations.
Explanation: The Q matrix is constructed by applying Householder transformations. This ensures orthogonality and stability in the computed probabilities (`q_vector`), aligning with the multinomial tree model constraints.

6. Function: `compute_payoff` **Purpose**: Calculates the payoff for a given set of shares and configurations.

Step 1: Determine the type of option (call or put).

Step 2: Calculate the payoff based on the option type and share values.

Step 3: Return the computed payoff value.

7. Function: `main` **Purpose**: Main function to execute the option pricing algorithm.

Step 1: Initialize the FPGA and load kernels.
Explanation: This step involves configuring the FPGA hardware and loading the required kernels for the computation.

Step 2: Prepare initial data structures and parameters.

Explanation: This step initializes the share prices and other parameters. Note that the share prices are built up at this stage, and a dedicated FIFO is used to manage the initial share prices for subsequent computations.

Step 3: Call the `doKernel` function to process FIFOs and compute option values.
Explanation: The `doKernel` function is invoked to manage the main processing loop. It handles reading from FIFOs, performing calculations to update asset prices, and computing option values based on the model.

Step 4: Display the final computed option price.
Explanation: Once the `doKernel` function completes, the computed option price is output to the user.

Step 5: Clean up resources and exit the program.
Explanation: This step releases all allocated resources, such as memory and device buffers, before safely exiting the program.

4.5 Conclusion

The primary objective of Chapter 4 was to bridge the gap between the theoretical underpinnings of the multinomial tree method and its practical implementation in the context of option pricing. By translating mathematical models into an operational framework, this chapter lays the groundwork for understanding how complex financial derivatives can be valued using advanced hardware solutions.

To achieve this, the chapter began with a review of the multinomial tree methodology, emphasizing its adaptability for modeling multi-asset options. Key mathematical constructs, including the computation of direction vectors and the Q matrix, were introduced to provide the necessary foundation for the implementation. These elements, critical to the multinomial framework, were carefully designed to align with the constraints and capabilities of FPGA platforms.

The chapter then detailed the implementation strategy, discussing how essential parameters, such as transition probabilities and asset price nodes, were computed and structured for FPGA deployment. Specific emphasis was placed on the role of static precomputation and the use of FIFO queues to manage

intermediate results efficiently. This strategy not only streamlined the computational process but also demonstrated how hardware constraints could be addressed through algorithmic optimization.

Through the combination of theoretical insights and practical considerations, the chapter demonstrated how the multinomial tree method could be adapted to FPGA architectures to achieve both accuracy and efficiency. By introducing and elaborating on the core variables and parameters required for the option pricing model, the discussion serves as a pivotal link between the abstract mathematical framework and its real-world application.

In summary, Chapter 4 serves as a bridge, translating theoretical constructs into concrete inputs to the option pricers presented in the next chapters. Chapters 5 through 7 use the inputs constructed in this chapter to accelerate the option pricing algorithm.

Chapter 5

Two-asset European option pricing

This chapter presents the first architecture and implementation of a two-asset option pricer based on Pascal's simplex, which takes advantage of the parallelism and pipelining offered by FPGA technology. The theory that this architecture is constructed from is based on a recombining multinomial tree approach which, in turn, is a generalization of the binomial tree model discussed in the previous chapters. Furthermore, we show that while a significant difficulty exists in efficiently maintaining the intermediate values required for the computation, a solution exists in the form of FIFOs.

Our implementation, on an Intel Stratix 10 GX FPGA, is based on the OpenCL framework and can compute 6250 two asset option prices per second for a time step depth of 100 and the pipelining of the option value computation shows a 25 times improvement when a 50-step pipeline is created.

5.1 Introduction

Hardware accelerators based on Field-Programmable Gate Arrays (FPGA) have found many applications in finance[161][162]. The ability to create custom circuits for frequently performed complex operations[163], as well as the ability to parallelize financial applications on FPGAs[164], make them attractive for acceleration purposes. Examples are discussed in [79],[119], and [76].

Financial options are one tool that can be utilized by businesses in increasing the potential for profit[165]. One of the most popular models used for com-

puting the value of an option is the Black-Scholes model[17]. This approach allows for generating a formula which provides an estimate of the price of an option. An alternative model is the *Binomial option pricing model*, created by Cox, Ross and Rubinstein[18]. The two approaches (Black-Scholes and Binomial option pricing model) provide for calculating the value of an option on a single asset. In this chapter, we are concerned with calculating the value of an option on two assets[76].

The remainder of the chapter is as follows: Section 5.2 discusses related research in this area. In Section 5.3 we provide an overview of financial options, the Binomial options pricing model, and then we provide a summary of the theory underlying the computation of the price of a derivative on two assets using recombining multinomial trees. Section 5.4 outlines the architecture of the reconfigurable option pricing hardware accelerator as well as detailing the OpenCL kernels required to instantiate the architecture. Section 5.5 provides details on the test environment as well as results from the implementation. Finally, in Section 5.6, we review our conclusions.

5.2 Related Work

Option pricing acceleration has been explored in the past[166]. However, no implementation to date has implemented two-asset option pricing using Pascal's Simplex. In this section, we review the existing research in this area to demonstrate the novelty of our approach.

In [167] an OpenCL Option Pricer on an Altera Stratix IV is presented which can achieve 2000 options/s for a depth of 1024. To reuse this implementation for two assets requires the computation of all combinations of possible outcomes which potentially results in a computation rate of 30 options/s (see Section 5.3.3 for comparison method).

An alternative implementation for computing multi-asset option prices on FPGA is presented in [168] where a Monte Carlo approach is used. While this approach achieved a 123 times speedup on a single FPGA, the accuracy and convergence issues that exist in the Monte Carlo method, when compared to the Binomial Model [169], were not addressed. Other Monte Carlo implementations on FPGA are also presented in [163] and [89] without accuracy comparisons. However, in [170] it is noted that the Monte-Carlo solver is at least 100 times less accurate than other pricing methodologies. Indeed, they

remark that their results show that the FPGA based Monte Carlo solver should only be used when there are no other solvers available.

Quadrature Methods for Option Valuation are considered in [171] and [172]. This approach compares well against other methods in terms of accuracy. However, as presented in [170], the quadrature method does not compare favourably with the tree-based approach as the time variable increases.

Other combinations of hardware acceleration of option pricing also exist, e.g. CPU-GPU hybrid [173]. However, none of these alternative approaches allows for multi-asset option pricing.

In the prior research surveyed, CPU implementations generally compare poorly in terms of performance[102] and, as a result, we do not consider a CPU implementation in our work. The same survey noted research that concluded that FPGAs provide better performance portability in terms of achieved percentage of device's peak performance compared to NVIDIA GPUs [83], and FPGA based option pricing outperform GPU based option pricing [174] in terms of latency.

Our two-asset option pricer based on Pascals Simplex is shown to be novel based on the current state of the art. Furthermore, based on our survey, the FPGA based Simplex based multi-asset option pricer also has clear differences compared to alternative approaches, such as using a single asset option pricer, Monte Carlo based multi-asset option pricer, or Quadrature based multi-asset option pricers.

5.3 Financial Options

The most common definition of an option is an agreement between two parties, the option seller and the option buyer, whereby the option buyer is granted a right (but not an obligation) to carry out some operation (or exercise the option) under specific conditions. For European options, this right can only be exercised at a predetermined future date, known as the expiration date[175]. In contrast, American options allow the holder to exercise the option at any time up to and including the expiration date.

Options with more general payoffs can also be considered, and these are often approximated by an appropriate portfolio of call and put options. The predetermined price at which the option can be exercised is referred to as the strike

price. This chapter focuses primarily on European options unless stated otherwise, as their payoff structures align closely with the multinomial tree framework discussed in this work.

There are two main types of options. A call option grants its holder the right to buy the underlying asset at the strike price, while a put option grants the holder the right to sell the underlying asset at the strike price.

The question is, what determines option values? We know the value of an option at maturity, but how do we compute its value initially? Several variables are relevant to these calculations [175], including the price of the underlying asset, the strike price (the predetermined price at which the option can be exercised), the volatility of the asset price over time, and the term of the option. These variables form the foundation of the Binomial Option Pricing Model.

5.3.1 Binomial Option Price Model

The binomial model is a discrete-time model for pricing options in which it is assumed that price changes in the underlying assets occur only after regular time intervals[176]. It involves constructing a binomial tree that represents different possible paths that the price of the underlying asset might follow. Let Z_T be the stock price at the expiration time T and K the strike price. Then the pay-off of a European call option is $c(T) = \max(Z_T - K, 0)$, which reflects the fact that an option might be exercised if $Z_T > K$ and might not be exercised if $Z_T \leq K$. The payoff of a European put option is $p(T) = \max(K - Z_T, 0)$, which reflects the fact that an option might be exercised if $Z_T < K$ and might not be exercised if $Z_T \geq K$.

Consider a non-dividend paying stock whose price is initially Z_0. Divide time into small time intervals of length δt. A time interval will be referred to as a period. Denote by Z the initial stock price at the beginning of a time interval. Assume that in each time interval the stock price moves either to $d_1 Z$ (an "up" movement) or to $d_2 Z$ (a "down" movement). The parameters d_1 and d_2 are equal to one plus the realized return during the time interval. In general, $d_1 > 1$ and $d_2 < 1$. Let q be the probability of an "up" movement and $(1 - q)$ be the probability of a "down" movement.

In 1979, Cox, Ross, and Rubinstein introduced a method to approximate the value of options on a single asset Z via a binomial tree[18]. The goal is to construct a recombining binomial tree with N levels to approximate the price

process of an asset Z and the price of an option. In each time-step, the value of the asset can go either up or down (although $0 < d_2 < d_1$ is all that is required), where the asset price at time 0 equals Z_0. Each node in the tree represents a possible outcome of the asset price and has a certain probability.

5.3.2 Recombining Multinomial Trees Based on Pascal's Simplex

This method is a generalization of the well-known binomial tree method to options on several assets. An advantage of the method introduced in [159] is that it has a minimal number of successor nodes at each node in the tree. Therefore, the proliferation of nodes in the case of several assets is less severe than in other tree-based approaches (such as using a separate binomial tree for each asset, where all combinations of possible outcomes need to be computed).

By exploiting the fact that the simplex-trees have close connections to polynomial algebra (i.e., each node can be associated with a multi-index $\nu = (\nu_1, \nu_2, \ldots, \nu_{k+1}), k = 2$ (two assets), with corresponding time step index $t = |\nu| := \sum_{j=1}^{k+1} \nu_j$, which in turn can be associated with a monomial $x^\nu = x_1^{\nu_1} x_2^{\nu_2} \ldots x_{k+1}^{\nu_{k+1}}$ of auxiliary variables $x_1, x_2, \ldots, x_{k+1}$), we use graded monomial ordering (grlex) [177] tailored to the multinomial tree model. This graded ordering is crucial to ensure a systematic approach for bookkeeping the values attained at the nodes of the tree and their relationships with neighboring nodes. It aligns with the multinomial tree structure and differs from the trinomial [1] ordering described earlier in the thesis.

Figure 5.1 presents a visualization of a projection of the random walk of the recombining trinomial tree in three dimensions onto two dimensions. This example uses three-time steps of length δt on two assets Z_1 and Z_2. The random walk starts at the black node on top, which is projected onto the black node in the two-dimensional plane. From this starting point, there are three possible moves to time δt, represented by the red nodes and lines. The transitions from δt to $2\delta t$ are depicted with blue lines, while green lines represent transitions from time $2\delta t$ to time $3\delta t$. The possible outcomes at time $T = 3\delta t$ are represented by the green nodes and the black node.

[1] Recall, there is a trinomial option pricing model for a single asset (up, down, no change) but this trinomial approach is for two assets.

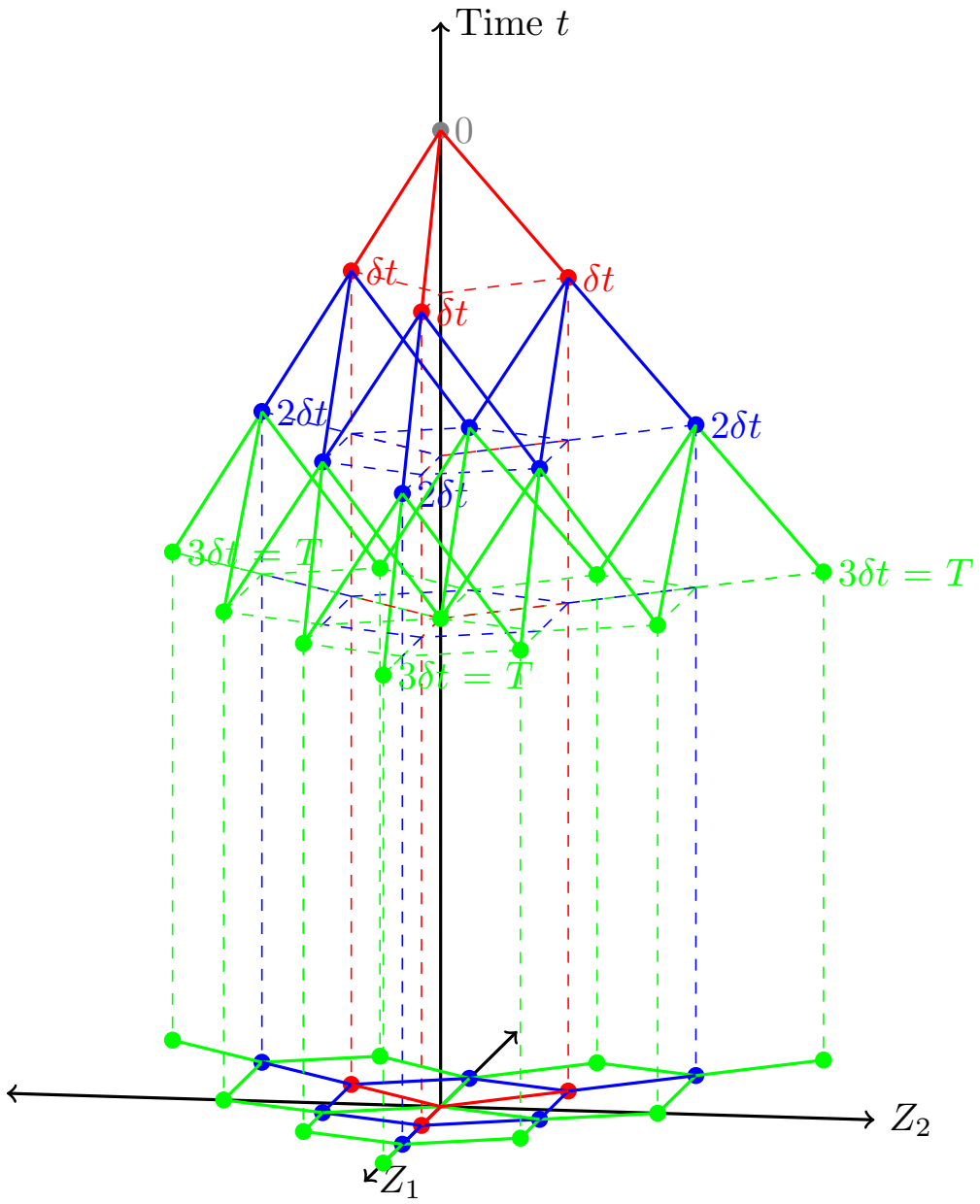


Figure 5.1: Example of 3-dimensional to 2-dimensional projection, redrawn from the example in [159].

The option price computation is performed as follows: at final time T the asset price vector $Z(T)$ has a value on each of the $\binom{T+2}{2}$ (in general $\binom{T+k}{k}$) nodes ν on the tree at the final time. Let the values of Z, F on the tree at node ν now be denoted as $Z(\nu)$ and $F(\nu)$(F denotes the pay-off function), respectively. We also define $F(v)$ as the value $F(t, Z)$ where (t, Z) are the values of the time and of the process Z at node v. For each ν with $|\nu| = T$ the option value $V(\nu) = F(Z(\nu))$ is calculated ($V(v)$ defined in equation 5.1). The value (for a European option) at node ν at time $t = |\nu| < T$, given the values at time $t+1$ at the successor nodes $\nu + e_j,\ j = 1, 2, 3$, where e_j stands for the standard basis row vector with entry 1 on position j and zeros elsewhere, is given by

$$V(\nu) = \sum_{j=1}^{3} q_j V(\nu + e_j) e^{-r\Delta t}, \tag{5.1}$$

where r stands for the interest rate per time unit. So, the task to be carried out by the acceleration device is (1) to generate the asset price vectors $Z(\nu), |\nu| = T$ in case of European options (and $Z(\nu), |\nu| = 0, 1, 2, \ldots, T$ in case of American options, which we do not treat here, however), (2) to compute the option prices $V(\nu) = F(Z(\nu))$ at the final time $|\nu| = T$, and (3) to iteratively compute the option values $V(\nu)$, for decreasing values of $|\nu|$, for all $|\nu|$ with $|\nu| \leq T$, ending up with $V(0)$, which is the desired option value approximation.

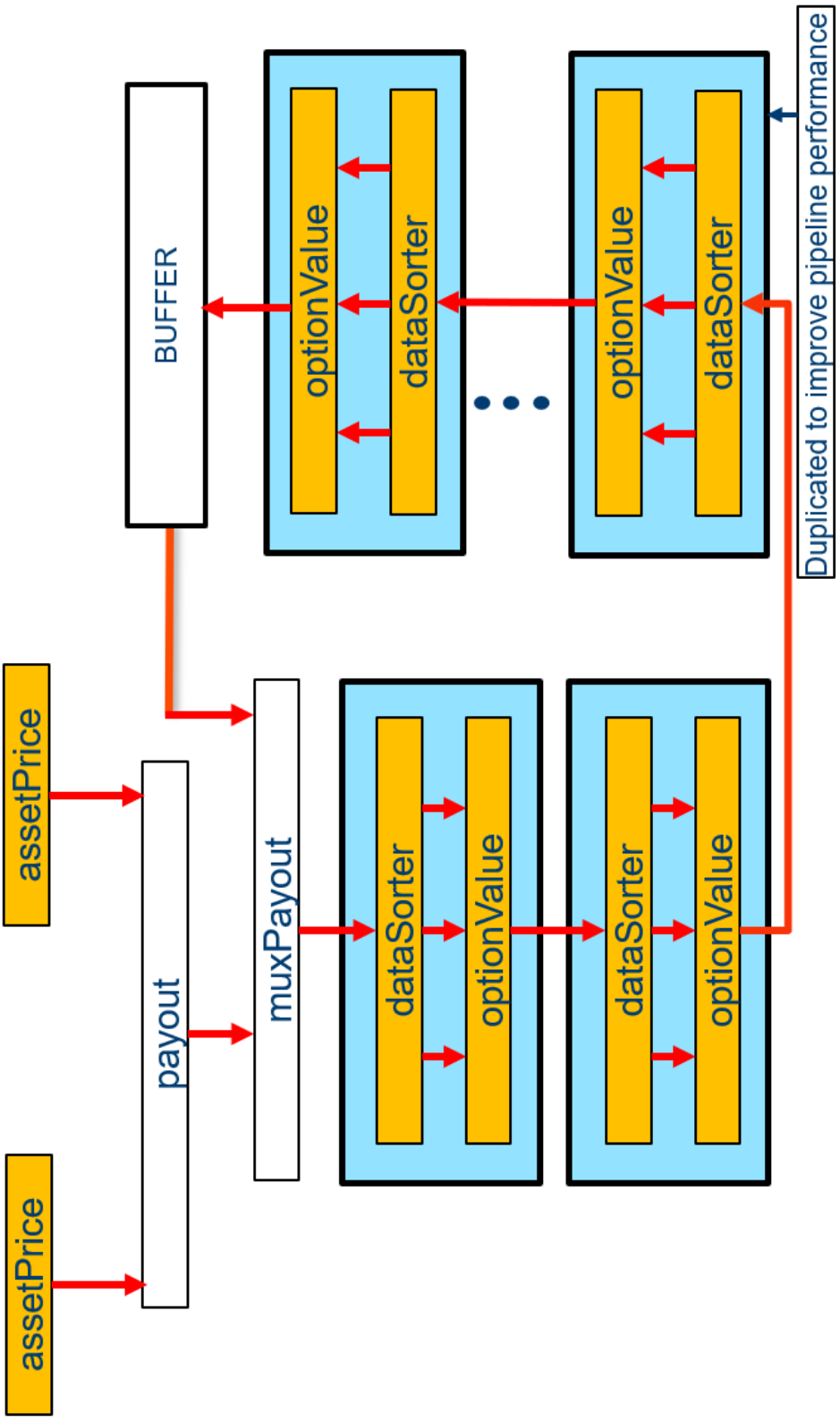


Figure 5.2: The block diagram of the proposed system architecture

5.3.3 Comparison against using Binomial Trees for Two Asset Computation

The number of nodes required to construct the binomial trees, when compared to the simplex-based approach, has a significant impact on the computational effort required to calculate the option price. In the case where binomial trees are used, a separate tree must be constructed for each of the two assets, and all combinations of nodes from each tree must be considered at every time step. This dramatically increases the computational complexity.

Assuming a tree depth of N, our trinomial tree for two assets requires $\frac{N+3}{3} \cdot \frac{1}{2}(N+1)(N+2)$ nodes. This result is derived as follows: - For a trinomial tree, the number of nodes at a given time step t is given by $\binom{t+2}{2} = \frac{(t+2)(t+1)}{2}$, which accounts for all possible outcomes of the two assets at each step. - Summing this across all time steps up to N, and considering the trinomial recombining structure, leads to the total number of nodes $\frac{N+3}{3} \cdot \frac{1}{2}(N+1)(N+2)$.

In contrast, using multiple binomial trees for a trinomial problem requires $\frac{2^{N+3}}{3} \cdot \frac{1}{2}(N+1)(N+2)$ nodes, as the number of nodes increases exponentially with 2^t for the binomial tree of each asset, combined at each step.

As an illustration, consider $N = 768$: Using multiple binomial trees, the total number of nodes required is approximately 151,881,345. Using the multinomial tree approach, the total number of nodes required is approximately 76,088,705, which is nearly half as much, demonstrating the computational efficiency of the multinomial approach.

This reduction in node count highlights the practicality of the multinomial tree for multi-asset computations, making it a more scalable and efficient method.

5.4 Architecture

The architecture (illustrated in Figure 5.2), which achieves pipelining and parallelism, is described in this section. The parallelism is possible since the European option price allows us to compute the stock value (the leaf nodes of the multinomial tree) directly without storing the intermediate values. These are the *leaf nodes* of the multinomial tree, corresponding to the values $V(\nu)$ at the final time step $|\nu| = T$. This is the task (1) described in Section 5.3.2 (i.e., generation of the asset price vectors $Z(\nu), |\nu| = T$). For simplicity, we align the

tree depth N with the expiration time T, i.e., the tree depth corresponds with the expiration time in our architecture. This alignment is achieved by rescaling the mean and variance at each step without any loss of generality.

Our implementation of the tasks (2) and (3) from Section 5.3.2 iteratively computes the option values $V(\nu)$. This computation is carried out by the pipelined stages of our architecture. Once the first three leaf nodes $V(\nu)$ are computed, the pipeline stage can begin processing. These nodes correspond to the encoded points $x^{\nu} = (x^{\nu_1,0}, x^{\nu_1,\nu_2,0}, x^{\nu_1,\nu_2,\nu_3,0})$, representing the initial values required for intermediate computations. The remaining $V(\nu)$ values are not immediately required for the initial stages of intermediate European value calculations. The pipeline can be extended up to the limits of the hardware. Buffers, organized as separate FIFOs, are required to store intermediate option values for the respective computations.

All red arrows in Figure 5.2 are "channels" used for passing data between kernels. These channels are analogous to "pipes" in the OpenCL standard. While "pipes" are a part of the OpenCL standard, "channels" are an extension of pipes specific to the Intel implementation of the OpenCL standard and, therefore, are not usable on platforms that do not support the Intel OpenCL SDK.

The required OpenCL kernels are presented in Table 5.1 and described in detail in Subsection 5.4.3. The producer and consumer kernels are not shown in Figure 5.2. However, these are simply used to interface with the host code running on the CPU.

5.4.1 Computation of the Direction Vectors and Risk-Neutral Probability Vector

Recall the parameters p, u, r and d from the Binomial Option Price Model presented in Chapter 2. Conceptually, for the multinomial case, we also have these parameters; however, they do not describe an up and down movement and the probability of those movements in the same fashion as the binomial case. Instead, they need to describe the probabilities of these movements for both assets in a multidimensional movement. Similarly, to the binomial case, however, we can compute this matrix and vector once, in advance of running the computation of the final option value. The elements of the direction matrix d are referred to as *direction vectors* d_i, where $0 \leq i \leq k$, and these vectors belong to $\mathbb{R}^k$ (as we are dealing with $k = 2$ assets here). Each vector d_i con-

Table 5.1: OpenCL kernels for two asset option pricing

Kernel	Description
producer	Takes values from host kernel
assetPrice	Asset price calculation $Z(\nu)$ (duplicated, running in parallel)
optionValue	European Value Calculation $V(\nu)$
payout	Computes payout $F(\nu)$
muxPayout	Maintains order of payout values
muxTreeDepth	Maintains tree depth value
dataSorter	Reads values from previous calculations and determines correct usage for value
returnBuffer	Used to buffer intermediate values
consumer	Consumer of final value calculation

tains elements $d_{i,j}$, where $0 \leq j \leq k$, which describe the movements and their associated probabilities for asset i in dimension j.

5.4.2 Pipelined Computation of Final Option Value

In Figure 5.2 we make use of two kernels to compute each node value on the return traversal of the tree, *dataSorter* and *optionValue*. On the return traversal, we need to compute $V(v)$ for decreasing values of $|v|$ (equation 5.1). As we can see, there is one input to the *dataSorter* kernels, which is the option value from the previous level of the tree. We need three such values to compute the value of the next level, and the *dataSorter* has knowledge of the correct order, i.e. graded monomial ordering [177], that the previous values must be arranged into. The size of the pipeline has a significant impact on performance and the *dataSorter/optionValue* block can be duplicated as many times as the hardware can support.

5.4.3 OpenCL Kernels for Two Asset European Option Pricing

Table 5.1 provides an overview of the kernels required to compute the option value. This section describes each kernel in more detail.

5.4.3.1 producer

The *producer* kernel serves two purposes: (1) it acts as an interface between the host system and the FPGA, and (2) it distributes the provided inputs to the appropriate channels. These inputs include precomputed parameters, as detailed in Table 4.1. The kernel ensures the seamless flow of data into the FPGA, enabling the subsequent computation stages.

5.4.3.2 assetPrice

This kernel computes the value of the underlying assets at each node, denoted as $Z(\nu)$, where $|\nu| = N$. The computation is based on precomputed values in the *end_node_values* vector and the *init_step* matrix. For European options, this direct computation approach is feasible as intermediate asset prices do not need to be stored. However, this approach is unsuitable for American options (see Chapter 7 for the American asset case), where the possibility of early exercise requires intermediate asset prices. For the European case, the *assetPrice* kernel efficiently calculates $Z(\nu)$ at the final time T.

5.4.3.3 optionValue

The *optionValue* kernel calculates the option value $V(\nu)$ at a given node ν by implementing Equation 5.1. This kernel uses the *mult_coeff* vector, which contains the precomputed coefficients $\frac{1}{R} * q$. It combines these coefficients with the payout values passed into the kernel to determine $V(\nu)$ efficiently. For the two-asset case, q is a vector, and this kernel processes the values for both assets simultaneously.

5.4.3.4 payout

This kernel computes the payout for a given option. The payout is defined as:

$$F(N) = \begin{cases} \max(K_N - Z_N, 0), & \text{for a call option, European style,} \\ \max(Z_N - K_N, 0), & \text{otherwise.} \end{cases} \tag{5.2}$$

Here, Z_N refers to the vector of asset prices, and K is the strike price. For clarity, Z_N represents multiple assets when dealing with multi-asset options.

5.4.3.5 muxPayout

The *muxPayout* kernel initially reads payout values computed by the *payout* kernel from the asset price computations. After consuming all asset price computations, it switches to reading values from the option value pipeline. This kernel ensures smooth data flow between stages.

5.4.3.6 muxTreeDepth

The *muxTreeDepth* kernel maintains the current tree depth during the reverse traversal of the value tree. Acting as a control kernel, it signals other kernels to complete once the traversal reaches the root node. This kernel is essential for coordinating the computation pipeline.

5.4.3.7 dataSorter

The *dataSorter* kernel arranges the outputs of each computing block into the correct order for the *optionValue* computation. It employs a graded reverse lexicographical monomial ordering scheme (Definition 3.3 in [178]). This optimized order ensures that values required for multiple node computations are rearranged and available just in time for processing. For the multi-asset case, this kernel uses three FIFOs—one for each direction vector.

5.4.3.8 returnBuffer

The *returnBuffer* kernel is needed when the depth of the tree exceeds the number of computing blocks supported by the FPGA. This kernel provides temporary storage, with the maximum size of the buffer determined at compile time based on the FPGA hardware's computing block configuration.

5.4.3.9 consumer

The *consumer* kernel transfers the final computed option value from the FPGA back to the host system. This kernel ensures that the host receives the computed results for further analysis or display.

5.5 Results

The device used for this experiment was the Intel Stratix 10 GX. The Intel FPGA SDK for OpenCL supplied (version 19.2) with the Stratix 10 supports the OpenCL 1.0 Standard. The host machine was a Dell Precision 7810. We measured the latency of all the running kernels using the OpenCL event profiling information. The pipeline was implemented using 64-bit floating-point precision (double data type). Host-to-device and device-to-host transfer times were measured using high-resolution wall-clock timing around buffer access operations. For a total data size of approximately 82.8 KB, the measured transfer time was **7 μs** per iteration, accounting for just **0.6%** of the total runtime at tree depth 100 (X1 configuration).

Table 5.2: Input to two-asset option pricer

Parameter	**Value**
Option Type (Call or Put)	Call
Number of Layers (Tree Depth)	2000
Number of Assets	2
Initial Asset Price (Asset 1)	40
Initial Asset Price (Asset 2)	40
Sigma Matrix (Volatility): $\sigma_{1,1}$	0.04
Sigma Matrix (Volatility): $\sigma_{1,2}$	0.03
Sigma Matrix (Volatility): $\sigma_{2,1}$	0.03
Sigma Matrix (Volatility): $\sigma_{2,2}$	0.09
Path Probability (Path 1)	0.29595426
Path Probability (Path 2)	0.380041989
Path Probability (Path 3)	0.324003751
Time to Maturity (T, in years)	0.583333
Risk-Free Rate (r)	0.05
Strike Price (Asset 1)	45
Strike Price (Asset 2)	45
Option Basis (Max or Min)	Max

The pipelining of the *dataSorter* and *optionValue* block is significant regarding the overall latency. We measured three different levels of pipeline

Table 5.3: Latency performance and pipelining impact

Tree Depth	**Time (usecs) X50**	**Time (usecs) X25**	**Time (usecs) X1**	**Ratio X1/X50**	**Ratio X1/X25**
100	160	190	1220	7.62	6.42
200	580	617	8347	14.39	13.52
300	1441	1639	27296	18.94	16.65
400	2862	3430	63928	22.36	18.65
500	4985	6225	124092	24.89	19.93

Table 5.4: Performance-area results

	Ours	Tavakkoli[81]	Wynnyk [78]	Jin[119]
Tree Depth	96	96	96	96
Precision	Double	Fix 16.16	Double	Fix 16.16
FPGA	Stratix 10 s10gx	Virtex-4 xc4vsx55	Stratix III ep3se260	Virtex-4 xc4vsx55
Clock (MHz)	320	224	150	82.7
LUTs RAMs DSPs	18% logic 700 (5%) 158 (2%)	85% logic 401 (78%) 11 (3%)	10% logic 84% bits 148 (19%)	22% logic 252 112 (22%)
Latency usecs	133	162.5 (estimated)	507 (estimated)	21970 (estimated)

sizes: using only one *dataSorter/optionValue* block, using twenty-five *dataSorter/optionValue* blocks and using fifty *dataSorter/optionValue* blocks. In Table 5.3, we present these latencies. An example input is shown in Table 5.2 however the average latencies are taken from executions of 2000 different inputs using pseudo-random inputs, where randomness is provided by the standard C library RNG. The ratio of depth to pipeline size illustrates the benefit of the larger pipeline when computing for deeper trees and the X50 pipeline helps reduce the latency by a factor of 25 when computing two-asset option

values for depth 500.

Also noteworthy is the system utilization which was only 18% of flip-flops, 5% of RAMs, and 2% of DSPs for the 50x deep pipeline design and requires an estimated 39.3 Watts (using maximum power characteristics).

The performance demonstrated by our approach when compared to the implementation by Tavakkoli[81] (which is implemented using systolic arrays), Wynnyk [78] (which uses a pipelined architecture), and Jin[119] (which also implements a pipelined architecture using "HyperStreams") as illustrated in Table 5.4 shows that using a single option pricer to achieve the same goal has a higher latency when we consider that using a binomial tree for each share individually requires significantly more computation effort. Note, the estimated latencies of the alternative implementations are solely based on the number of nodes required. However, they do not account for the complexity of the recombining operation, therefore it is expected the actual latencies will be higher in practice. Furthermore, our approach uses greater precision when compared to fixed precision.

5.6 Conclusion

Two-asset option pricing based on Pascal's simplex is an efficient method for computing the fair price of an option based on two underlying assets. Taking advantage of the pipelining and parallel aspects of the computation lends itself to hardware acceleration via FPGA. In this chapter we present such an acceleration architecture implemented using the OpenCL framework. Our implementation takes advantage of the channel's extension provided by the Intel FPGA SDK. This approach allowed us to achieve a performance of 200 two-option value calculations per second for a time step of 500 steps.

In the next chapter we explore how to extend the two-asset case to a multi-asset case.

Chapter 6

Multi-asset option pricing of European Options

Multi-asset option pricing is a financial tool that offers a competitive advantage in the global financial markets. To achieve the commercial advantage, performance and accuracy are two key desiderata of such tools. Techniques for financial option pricing have been actively researched for some time. Combined with recent advances in heterogeneous computing, new computing methods of lower latency option pricing have become available. Field-Programmable Gate Array technology offers a powerful framework for building architectures for highly parallel option pricing. This chapter presents a versatile, high performance implementation of an accurate multi-asset option pricer on an Intel Stratix 10 GX device (extending the two asset case from the previous chapter). The implementation is highly flexible for the number of assets supported, the maximum tree depth, and the pipelining parameters. This novel architecture was implemented on Field-Programmable Gate Array technology. The implementation presented in this chapter enables the computation of an option price on up to 7 assets, which is the limitation of our hardware, and can compute up to 43 times faster than a software implementation in a general-purpose processor.

6.1 Introduction

Financial applications have been shown to have critical latency requirements [179]. The ability to carry out these operations with low latency is crucial in being able to profit from opportunities. This need for low latency has accelerated the research into non-traditional platforms such as Graphics Processing Units (GPUs), or Field Programmable Gate Arrays (FPGAs), or custom Application-Specific Integrated Circuits (ASICs). One such application is that of options pricing. Market practitioners require continuous streams of option prices and, when required, orders are placed to exchanges. Modern trading consists of algorithms making high-frequency trading (HFT), and these algorithms depend on low latency updates to the options prices. The ability to reduce the latency of certain operations through the use of heterogeneous hardware accelerators is well documented [180].

Hardware accelerators based on FPGAs have found many applications in finance [161], [162]. The ability to create custom circuits for frequently performed complex operations [163] as well as the ability to parallelize financial applications on FPGAs [164] make them attractive for acceleration purposes. Examples involving tree-based financial applications are discussed in [79], [119], and [76].

Financial options are one tool that can be used by businesses in increasing the potential for profit [165]. One of the most popular models used for computing the value of an option is the Black-Scholes model [17]. This approach allows for generating a formula that provides an estimate of the price of an option (which is exact if all the assumptions underlying the model are satisfied). An alternative model is the *Binomial option pricing model*, created by Cox, Ross and Rubinstein [18]. In this chapter, we are concerned with the generalization, namely calculating the value of an option on multiple assets [159]. Indeed, a multi-asset option is an example of a financial innovation that offers a competitive advantage on financial markets as it offers the investor a broader range of underlying assets[181]. High-performance implementation of a multi-asset option pricer is, hence, of prime concern. In this paper, we present an original architecture for this algorithm and its efficient implementation on FPGA supporting up to 7 asset option pricing. To our knowledge, based on a survey of the literature, our implementation of this architecture on FPGA has not been proposed previously. The proposed architecture signifi-

cantly outperforms other software and hardware implementations.

The remainder of the chapter is as follows: Section 6.2 discusses related research in this area. In Section 6.3 we provide an overview of financial options, the Binomial options pricing model, and then we provide a summary of the theory underlying the computation of the price of a derivative on multiple assets using recombining multinomial trees. Section 6.4 outlines the architecture of the reconfigurable option pricing hardware accelerator as well as detailing the OpenCL kernels required to instantiate the architecture. Section 6.5 provides details on the test environment as well as results from the implementation. Finally, in Section 6.6, we review our conclusions for this chapter.

6.2 Related Work

In this section, we review the existing research in this area to demonstrate the novelty of our approach. Option pricing acceleration has been explored in the past[166]. However, no hardware implementation to date has been shown to implement multi-asset option pricing using Pascal's Simplex. It was shown in [159] that Pascal's simplex is a suitable method for computing the option price on multiple assets and a first evaluation of the computation complexity of the algorithm on CPU was also performed.

In [76], an OpenCL Option Pricer on an Altera Stratix IV is presented which can achieve 2000 options/s for a depth of 1024. Reusing this implementation for two assets requires the computation of all combinations of possible outcomes which potentially results in a computation rate of 3 options/s.

An alternative implementation for computing multi-asset option prices on FPGA is presented in [168], where a Monte Carlo approach is used. While this approach achieved a 123 times speedup on a single FPGA, the accuracy and convergence issues that exist in the Monte Carlo method, when compared to the Binomial Model [169], were not addressed. Other Monte Carlo implementations on FPGA are also presented in [163] and [89] without accuracy comparisons. Yet, in [170] it is noted that the Monte Carlo solver is at least 100 times less accurate than other pricing methodologies. Indeed, they remark that their results show that the FPGA based Monte Carlo solver should only be used when there are no other solvers available. Finally, [80] presents an implementation of a single-asset option pricer on FPGA using Monte Carlo with a 50 times speed up. They also provide a speed versus accuracy comparison.

Interestingly, they also detail a very low resource Monte Carlo path simulator kernel.

Quadrature Methods for Option Valuation are considered in [171] and [172]. This approach compares well against other methods in terms of accuracy. Yet, as presented in [170], the quadrature method does not compare favourably, in terms of latency, with the tree-based approach as the time variable increases.

More recently, machine learning and FPGAs are being combined for the purposes of option pricing. In [182], the authors present an option pricer that combines the Heston model with machine learning for a single asset to achieve a significant speedup compared to the CPU.

Other combinations of hardware acceleration of option pricing also exist, e.g., CPU-GPU hybrid [173]. However, none of these alternative approaches allows for multi-asset option pricing.

In the prior research surveyed, CPU implementations generally compare poorly in terms of performance[102] and, as a result, we do not consider a CPU implementation in our work. The same survey noted research that concluded that FPGAs provide better performance portability in terms of achieved percentage of device's peak performance compared to NVIDIA GPUs [83], and FPGA based option pricing outperforms GPU based option pricing [174] in terms of latency. As a result, the FPGAs are chosen as the underlying technology for our implementations.

Our multi-asset option pricer based on Pascal's Simplex is shown to be novel based on the current state of the art. Furthermore, based on our survey, the FPGA based Simplex based multi-asset option pricer also has clear differences compared to alternative approaches such as using a single asset option pricer, Monte Carlo based multi-asset option pricer, or Quadrature based multi-asset option pricers.

6.3 Financial Options

The most common definition of an option is an agreement between two parties, the option seller and the option buyer, whereby the option buyer is granted a right (but not an obligation) to carry out some operation (or exercise the option) at some moment in the future. There are also options with more general payoffs; typically, these can be approximated by an appropriate

portfolio of call and put options.

The predetermined price at which the option can be exercised is referred to as the *strike price*, and the future date at which the option expires is called the *expiration date* [175]. There are two main types of options:

- A **call option** grants its holder the right to buy the underlying asset at the strike price at some moment in the future.
- A **put option** grants its holder the right to sell the underlying asset at the strike price at some moment in the future.

In the context of **multi-asset options**, the concepts of call and put options are extended to apply to a portfolio of assets. Specifically:

- A **multi-asset call option** grants the holder the right to buy a set of underlying assets at their respective strike prices at a specified moment in the future.
- A **multi-asset put option** grants the holder the right to sell a set of underlying assets at their respective strike prices at a specified moment in the future.

Each asset within the portfolio has its own associated strike price and contributes to the overall payoff of the option. The payoff for a multi-asset call option is determined by summing the positive differences between each asset's price and its strike price, while the payoff for a multi-asset put option sums the positive differences between each strike price and its corresponding asset price.

The question arises: what determines option values? While the value of an option is known at its maturity, the challenge lies in computing its value initially. For the multi-asset case, several variables influence the option price:

- **The current prices of the underlying assets**: Each asset contributes to the overall payoff of the option, making its current price a key determinant.
- **The strike prices of the assets**: The strike prices define the exercise threshold for each asset in the portfolio.
- **The volatilities of the asset prices**: The fluctuations in the prices of the underlying assets affect the probability distribution of the final payoffs.

- **The correlation between the assets**: For multi-asset options, the correlation structure between assets is crucial. It affects how the prices of different assets interact and, consequently, the likelihood of certain payoff scenarios.
- **The time remaining until expiration (term of the option)**: The length of time to expiration determines how much the asset prices can potentially move, influencing the option's value.
- **The risk-free interest rate**: This rate is used to discount future payoffs back to their present value.

For multi-asset options, the computation becomes significantly more complex due to the need to account for the interaction between the assets.

6.3.1 Recombining Multinomial Trees Based on Pascal's Simplex

This method is a generalization of the well-known binomial tree method, which is used for option prices with one underlying asset, to options on several assets. An advantage of the method introduced in [159] is that it has a minimal number of successor nodes at each node in the tree. Therefore, the proliferation of nodes in the case of several assets is less severe than in other tree-based approaches (such as using a separate binomial tree for each asset where, all combinations of possible outcomes need to be computed).

To describe the behavior of asset prices, two distinct types of probabilities are involved:

- **Physical probability:** This describes the real-world behavior of the asset price and is used in simulating price paths.
- **Risk-neutral probability:** This is used for pricing derivatives by ensuring that the discounted expected payoff is equal to the current option price.

By exploiting the fact that the simplex-trees have close connections to polynomial algebra (i.e. each node can be associated with a multi-index $\nu = (\nu_1, \nu_2, \ldots, \nu_{k+1})$, with corresponding time step index $t = |\nu| := \sum_{j=1}^{k+1} \nu_j$, which, in turn, can be associated with a monomial $x^\nu = x_1^{\nu_1} x_2^{\nu_2} \ldots x_{k+1}^{\nu_{k+1}}$ of auxiliary variables $x_1, x_2, \ldots, x_{k+1}$). We use a *graded monomial ordering* [178] to sys-

tematically organize the values at the nodes of the tree and their relationships with neighboring nodes.

Figure 5.1 illustrates a visualization of an additive recombining multinomial tree projected from three dimensions to two dimensions with three-time steps of length δt on two assets Z_1 and Z_2. The random walk starts at the black node on top, which is projected onto the black node in the two-dimensional plane. From this node, there are three possible moves to time δt, represented by the red nodes and lines. The transitions are processed in order: first, the red lines corresponding to transitions to δt, followed by the blue lines representing transitions from δt to $2\delta t$, and finally the green lines representing transitions from $2\delta t$ to $3\delta t$. The possible outcomes at time $T = 3\delta t$ are represented by the green nodes and the black node.

Note: While some angles in the diagram may visually appear as $45°$, they are not $45°$ in terms of the mathematical order of processing within the multinomial tree.

The option price computation is performed as follows: at the final time T, the asset price vector $Z(T)$ has a value on each of the $\binom{T+2}{2}$ (in general $\binom{T+k}{k}$) nodes ν on the tree at the final time. Let the values of Z and F on the tree at node ν now be denoted as $Z(\nu)$ and $F(\nu)$ (F denotes the payoff function), respectively. We also define $F(v)$ as the value $F(t, Z)$ where (t, Z) are the values of the time and of the process Z at node v. For each ν with $|\nu| = T$, the option value $V(\nu) = F(Z(\nu))$ is calculated ($V(v)$ defined in Equation 6.1). The value (for a European option) at node ν at time $t = |\nu| < T$, given the values at time $t + 1$ at the successor nodes $\nu + e_j,\ j = 1, 2, 3$, where e_j stands for the standard basis row vector with entry 1 in position j and zeros elsewhere, is given by:

$$V(\nu) = \sum_{j=1}^{3} p_j V(\nu + e_j) e^{-r}, \tag{6.1}$$

where r stands for the interest rate per time unit. It is important to clarify that the angle between any two consecutive vectors m_1, m_2, and m_3 is not necessarily $120°$ in terms of the processing order, even though their spatial projection may suggest otherwise.

Thus, the task to be carried out by the acceleration device is as follows:

1. Generate the asset price vectors $Z(\nu), |\nu| = T$ for European options (and $Z(\nu), |\nu| = 0, 1, 2, \ldots, T$ for American options, though the latter is treated in the next chapter).

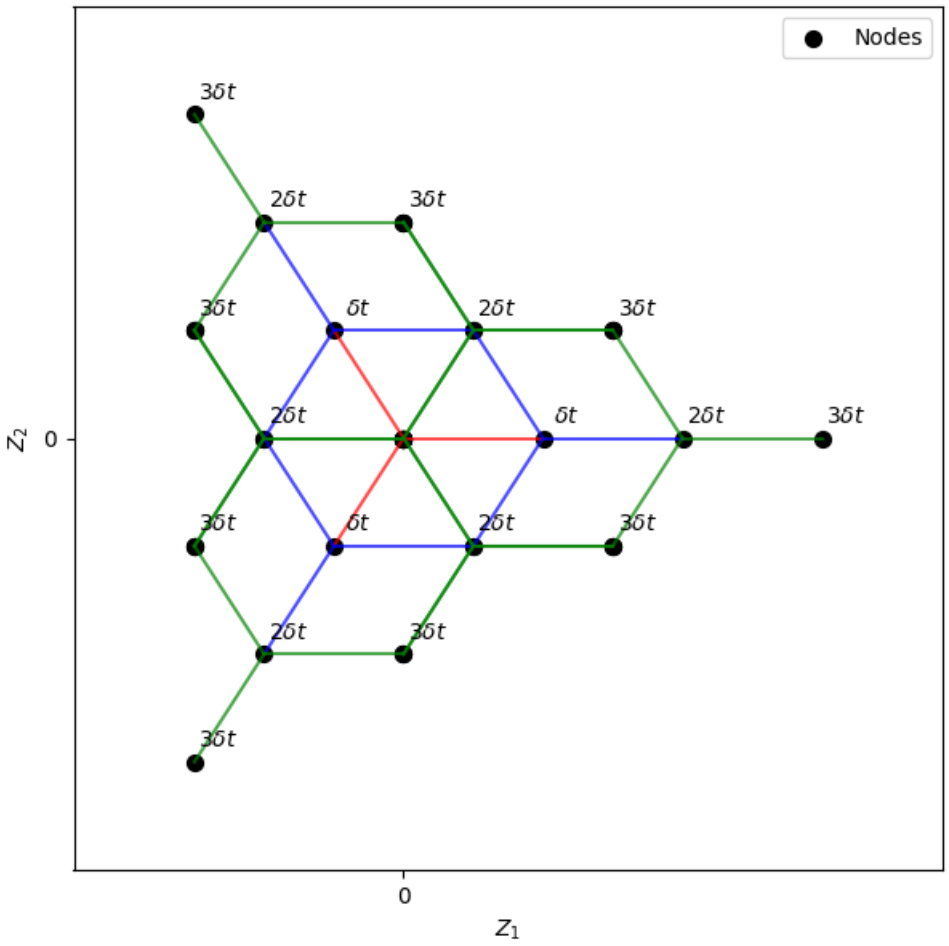


Figure 6.1: Visual example of a three-dimensional projection to two dimensions

2. Compute the option prices $V(\nu) = F(Z(\nu))$ at the final time $|\nu| = T$.
3. Iteratively compute the option values $V(\nu)$ for non-increasing values of $|\nu|$, for all $|\nu|$ with $|\nu| \leq T$, ending up with $V(0)$, which is the desired option value approximation.

6.4 Proposed FPGA architecture for financial options

The implementation of a two-asset option pricer presented in Chapter 5 can in theory be extended to compute the option pricer of more than two assets, however we noted a number of limitations. Firstly, the memory required to store the values computed during the option valuation quickly exceeded the

on-chip memory which necessitated the usage of the Double Data Rate Dynamic Random-Access Memory (DDR RAM). Secondly, to optimize the DDR RAM access for best performance, we implemented a multi-bank ping pong access method for efficient reading and writing of two DDR RAM banks.

6.4.1 High-level system design exploration

The high level design of our implementation is illustrated in Figure 6.2. Conceptually, we divided the option price calculation into two distinct phases. These are to (i) calculate the stock prices at the final time directly via a formula and, (ii) using pipelined compute blocks, calculate the option values for each level of the tree at the earlier nodes. A compute block (described in subsection 6.4.3.2) is a logical grouping of kernels together which enables the pipelining of multiple stages for increased performance. The data formats taken by the accelerator are that of double precision floating point numbers. The utilization of the hardware modules for each kernel is presented in Table 6.3. The two kernels which constitute the compute block, namely the $computeValue$ kernel and the $payout$ kernel, require the most resources out of the OpenCL kernel. This example is for a tree depth of 2048.

Figure 6.2: High-level architecture of the multi-asset option pricer. The stock value kernels are duplicated in line with the number of underlying assets and execute in parallel, allowing for the calculation of multiple stock prices simultaneously. The number of compute blocks, consisting of a data sorter kernel and an option value kernel, is determined at synthesis time and optimized based on the COMPUTE_UNITS parameter in the CMake configuration file.

Table 6.1: OpenCL kernel descriptions for kernels shown in Figure 6.2

Kernel	**Description**
Producer	Sends input data to correct kernels
Stock Values	Computes the stock value at final time
Data Pre-Sorter	Provides ordering information
Data Sorter	Arranges incoming data into correct order for next level
Payout	Compute payout based on previous levels data
channelDDR0	Sends data to DDR0
channelDDR1	Sends data to DDR1
Option Value	Compute option value for current node
Multi-tier buffer	Controls temporary storage of data

6.4.2 Multi-bank ping-pong access

The internal memory of the FPGA is extremely limited and the depths of the OpenCL FIFOs (which use the Intel channels extension to provide FIFO functionality) are quickly exhausted as the number of assets increases. To account for this limitation, we make use of the on-board DDR memory in a ping-pong fashion (illustrated in Figure 6.3). We also use additional DDR memory for temporary storage of data from lower layers of the tree, prior to them being required for the option value calculation itself. This channel size also grows exponentially as the number of assets and tree depth becomes larger. In this buffer, we store an option price (64 bits), plus other status variables for the tree (64 bits), for a total of 128 bits or 16 bytes. This means that the channel depth cannot exceed 536,870,912 entries (8GB of on-board SDRAM), or it will overflow the entire memory bank.

The multi-tier buffer architecture employed in this implementation is designed to efficiently manage memory access patterns arising during option pricing computations on FPGAs. It consists of multiple distinct memory banks organized into tiers, each serving as temporary storage for intermediate calculation results and final outputs. This hierarchical organization enhances the memory

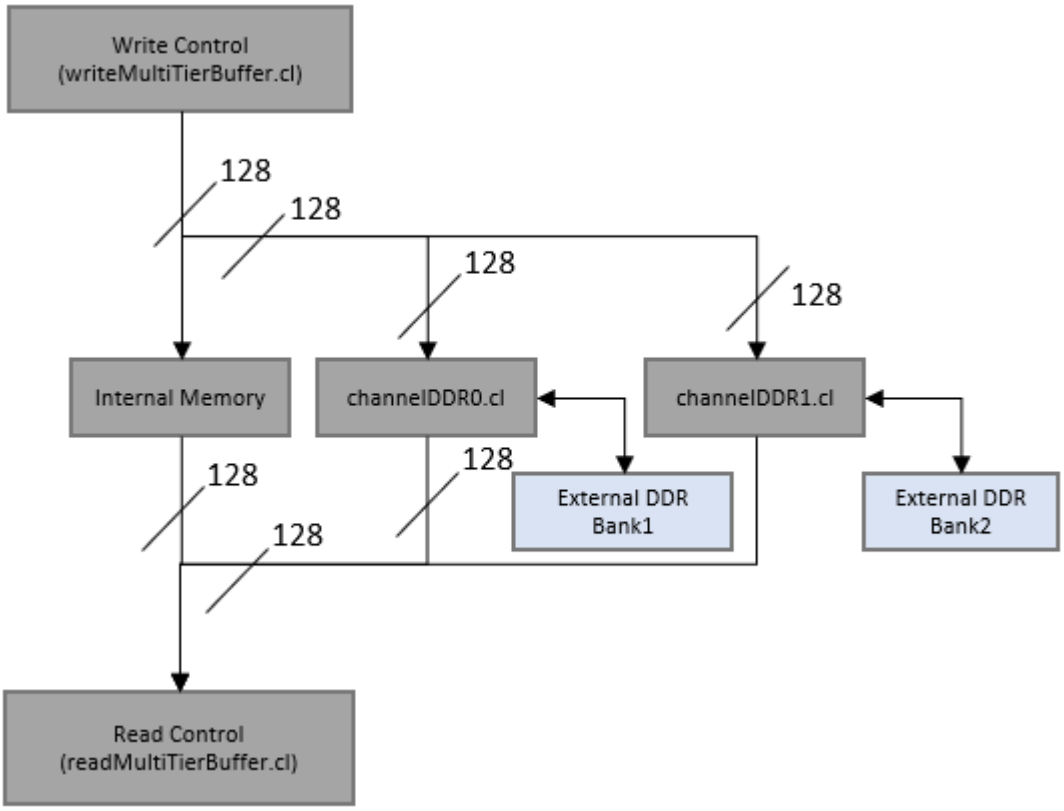


Figure 6.3: The Multi-tier Stock and Option Value Buffer controls where the values are read from or written to. When the data storage requirements drop under a certain threshold only Internal Memory is used. Each channel used within the diagram is 128 bits wide.

throughput by distributing memory access operations across different tiers.

At the heart of this multi-tier buffer architecture is the *ping-pong* access mechanism. The ping-pong strategy utilizes paired memory banks within each buffer tier, operating alternately in read and write modes. While one memory bank is actively being read by computational kernels, the paired *pong* bank is simultaneously being written with the next set of data. Upon completion of these operations, the roles of these two banks are swapped—hence the term *ping-pong*. This cyclical alternation allows continuous data flow, eliminating idle times and effectively masking memory access latency.

This buffering technique is particularly beneficial in financial computations involving multinomial trees, where each iteration of computations depends heavily on results from preceding steps. By using ping-pong buffers within the multi-tier buffer structure, data dependencies can be managed seamlessly, maximizing the computational efficiency and performance gains achievable on FPGA hardware.

Table 6.2: OpenCL kernel inputs. These inputs are precomputed from the traditional option price parameters, i.e., ρ, σ, etc. Section 6.3.1 explains $d_{i,j}$, etc., in more detail. This table generalizes to the multi-asset case, where M is the number of assets.

Input	**Description**
N	Tree depth, representing the number of time steps in the multinomial tree.
max_min	Boolean value indicating whether to compute call/put on the maximum or minimum of the asset values.
end_node_values	Defined as an M-element vector containing: $\left\{(d_{0,0})^N Z_0, (d_{1,0})^N Z_1, \ldots, (d_{M-1,0})^N Z_{M-1}\right\}$, where M is the number of assets, and each asset Z_i is scaled by the corresponding $d_{i,0}$ direction vector element raised to the power of N.
init_step	Defined as an $M \times M$ element matrix containing: $\left\{\frac{d_{i,j+1}}{d_{i,j}}\right\}$ for $i, j = 0, \ldots, M-1$, where $d_{i,j}$ represents elements of the direction matrix.
mult_coeff	$M+1$-element multiplicative coefficient vector containing $(1/R) * q_0, (1/R) * q_1, \ldots, (1/R) * q_M$, where R is defined as $e^{r*(1/N)}$ (r is the risk-neutral interest rate), and q_i are the risk-neutral probabilities.
share_coeff	M-element share coefficient vector containing: $\left\{\frac{1}{d_{i,j}}\right\}$ for $j = 1$ and $i = 0, \ldots, M-1$, where $d_{i,j}$ corresponds to the direction matrix elements.
strike_vector	M-element vector containing the strike prices K_i for each asset i.

Table 6.3 : OpenCL kernel utilization for kernels shown in Figure 6.2. The tree depth in this example is 2048 and the number of assets supported is 2.

Kernel Name	ALUTs	FFs	RAMs	DSPs	MLABs
channelDDR0	3164	7209	36	0	45
channelDDR1	3164	7209	36	0	45
computeValue	14103	19569	18	12	500
dataPreSorter	3566	5313	0	0	13
dataSorter	1204	2017	0	0	14
muxTreeDepth	1534	1493	0	0	6
payoutKernel	13391	18637	21	0	82
producer	8466	15509	162	0	62
readMultiTierBuffer	1528	2968	0	0	16
stockPrice	13875	20070	4	16	74
writeMultiTierBuffer	691	1172	0	0	8

6.4.3 System implementation details

OpenCL (Open Computing Language) is a framework for writing programs that execute across heterogeneous platforms consisting of central processing units (CPUs), graphics processing units (GPUs), digital signal processors (DSPs), field-programmable gate arrays (FPGAs) and other processors or hardware accelerators[183]. Because OpenCL is an open standard for parallel programming on heterogeneous platforms, it may be seen as more than a language, implying a specific work-flow, which provides solutions for FPGA design and implementation. Some OpenCL key concepts that organize the parallel code require introduction. The first one is the division of the code into parts: host code and device code. The host code is the sequential part of the application, comprising mostly of algorithm configuration and execution flow control. The device code is the computationally intensive part that will run on the specific parallel hardware. Parallel functions in the OpenCL context are called kernels, which are implemented by the programmer taking into account the parts of the application that require parallel execution and using the respective available API (Application Programming Interface). The OpenCL compiler converts it into the platform specific machine code according to the

programmable device, in this case the Stratix 10. This means that an OpenCL development is comprised of a sequential part, written in C/C++ and executed in the host, which prepares data to be processed in parallel, and a sequence of kernel calls, written in OpenCL, to process the data in specialized parallel hardware. An illustration of the programming model is shown in Figure 6.5.

6.4.3.1 OpenCL Kernels

Our design consists of nine OpenCL kernels. The OpenCL kernels are named and described in Table 6.1 with the inputs named and described in Table 6.2. Connecting the kernels are a significant number of channels. The categories of channels are:

- Channels to pass data from host to producer kernel
- Channels to pass data to and from Multi-tier Buffer
- Channels to pass pre computed Sorting order to payout kernel
- Channels to pass Tree Depth value between kernels
- StockPrice calculation Channels Out
- First Payout calculation Channels Out
- Multiplexer channels used for control data in duplicated kernels
- EU pricer channels

Each of these channels is implemented as an Intel channel and functions in a First-In First-Out (FIFO) fashion. The size of these channels is sensitive to the tree depth, in particular the channels in the compute blocks consume the most resources.

With regard to resource consumption of the individual kernels, the kernels controlling accesses to DDR RAM consume most of the memory related resources (although the compute blocks can consume more RAM if the user-supplied pipeline size is large enough and the number of compute blocks is duplicated enough). The Data Pre-Sorter consumes the most logic resources of the FPGA.

6.4.3.2 Compute Blocks

We can enable pipelining through the duplication of the two kernel block *Data Sorter* and *Option Value*, the block illustrated in Figure 6.4. This block is duplicated multiple times and organized in a pipeline to enhance performance. The number of channels is equal to the number of assets plus one. The channel sizes decrease progressively from right to left due to the graded lexicographic ordering. The far-right channel processes the largest number of nodes at the final time step, corresponding to the maximum tree depth. As the 'dataSorter' iteratively reduces the tree depth, the number of nodes per channel decreases, reflecting the convergence of the tree towards the root. The far-left channel, with a size of one, processes only the root node, representing the final computation result after all branches have merged. The size of the pipeline has a significant impact on performance and the dataSorter/optionValue block can be duplicated, using compiler and synthesis instructions, as many times as the hardware can support. The number of channels used by a compute block is always equal to the number of assets plus one. However, the depth of the channels, especially the first channel, grows quite significantly. Indeed, the first channel grows exponentially based on the combination of tree depth and number of assets.

6.5 Results and performance analysis

In this section we present our development environment and our results. The results encompass metrics such as the latency of the option pricer solution when asked to compute the option price for multiple assets (from two assets up to and including seven assets). We also present the impact on the utilization of the FPGA resources when we increase the size of the pipeline. Furthermore, we discuss the precision implication of our approach when compared to an analytical implementation.

6.5.1 Hardware and software specifications

The device used for this experiment was the Intel Stratix 10 SX Programmable Acceleration Card (Intel FPGA PAC) (shown in Figure 6.6). The Intel FPGA SDK for OpenCL supplied (version 21.2) with the Stratix 10 supports the OpenCL 1.0 Standard. The host code was developed on the Intel

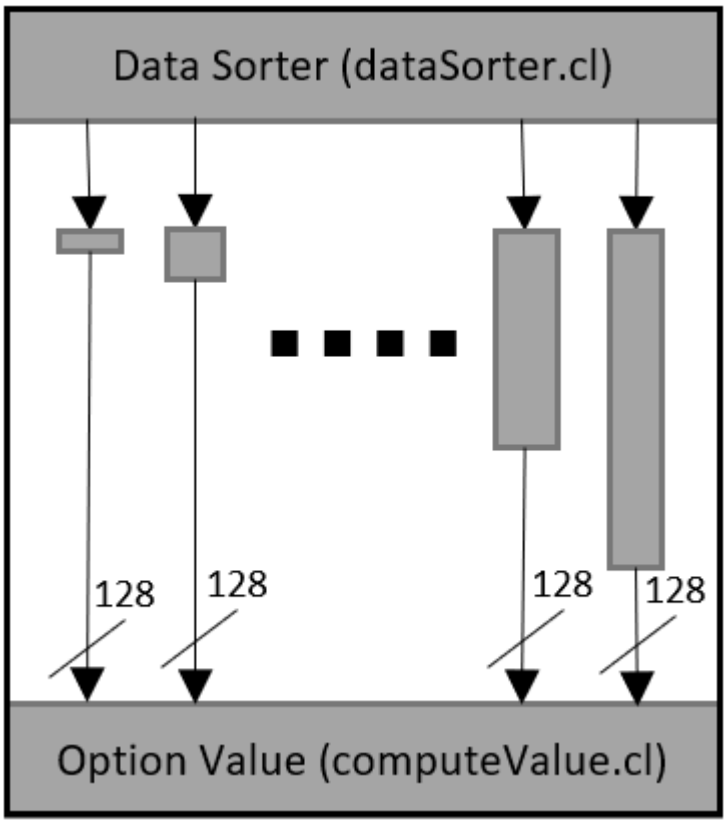


Figure 6.4: The compute block.

DevCloud[184]. We measured the latency of all the running kernels using the OpenCL event profiling information. The kernels were implemented using 64-bit floating-point precision (double data type).

The function to be accelerated is referred to as an OpenCL kernel. These kernels are written in standard C; however, they are annotated with constructs to specify parallelism and memory hierarchy. An OpenCL application consists of a host program running on the host CPU that triggers the execution of kernels in an accelerator device, managing all required data transfers in a master-slave model.

6.5.2 Metrics and key performance indicators

To assess our implementation we considered two important metrics, precision and latency. Both of these are important to users of option pricer implementations and are relevant to hardware implementations. Additionally, using FPGA resource utilization and the Intel Early Power Estimator (EPE), we estimate the power consumption of the multi-asset OpenCL implementation (supporting two assets at a tree depth of 2048) to be approximately 40.458 W.

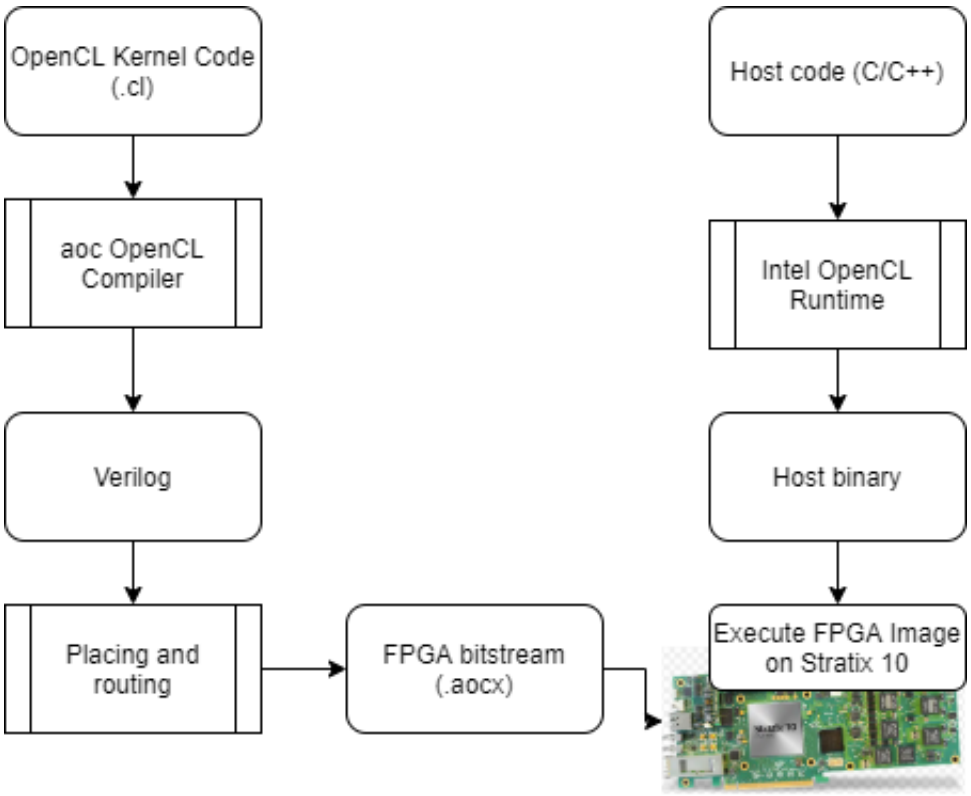


Figure 6.5: Schematic Diagram of the Intel FPGA SDK for OpenCL Programming Model



Figure 6.6: Stratix 10 Programmable Acceleration Card

6.5.2.1 Precisions comparisons

In this section we discuss the precision of the simplex approach when compared to the analytical option prices as computed by [185]. It has already been noted in the literature that, in the case of a binomial tree, there is a systematic underestimation or overestimation of an option price analytical approximation[186] and, indeed, we observe convergence differences between the simplex and the analytical option price.

We present a graphical comparison of the precision of two assets for a tree depth up to 4096 compared to [185] in Figure 6.7, with input parameters as shown in Table 6.4. In this example, we wish to calculate the price of a European call option X on the maximum of two stock values, with an exercise price of €10 where the initial stock values are S_1 =€40 and S_2 =€40. Also, we use an expiration date of 7 months in the future (parameter T). The tree depth, N, varies between 100 and 4096. We also use two volatility values (σ) both set to 0.3 and a ρ value of 0.1. The σ values and ρ describe the volatility matrix used to drive the share prices. σ_1 is the volatility for $S1$ and σ_2 is the volatility for $S2$. ρ represents the correlation of these volatilities. Our example also uses a risk free rate of 5%. The precision of four assets for a tree depth of 180 is presented in Figure 6.8 as a reference. In the case of two assets, it could be remarked that relatively small tree depths appear to get as close to convergence as larger tree depths.

Table 6.4 : Parameters to the option pricer which results in the call option prices presented in Figure 6.7

Parameter	**Symbol**	**Values**
Expiry Time	T	$7/12$
# of assets	n	2
$S1$ asset price	$S1$	40
$S2$ asset price	$S2$	40
Exercise Prices	X	10
Risk free rate	r	0.05
Sigma	σ	0.3
Rho	ρ	0.1
Tree Depth	N	Varies

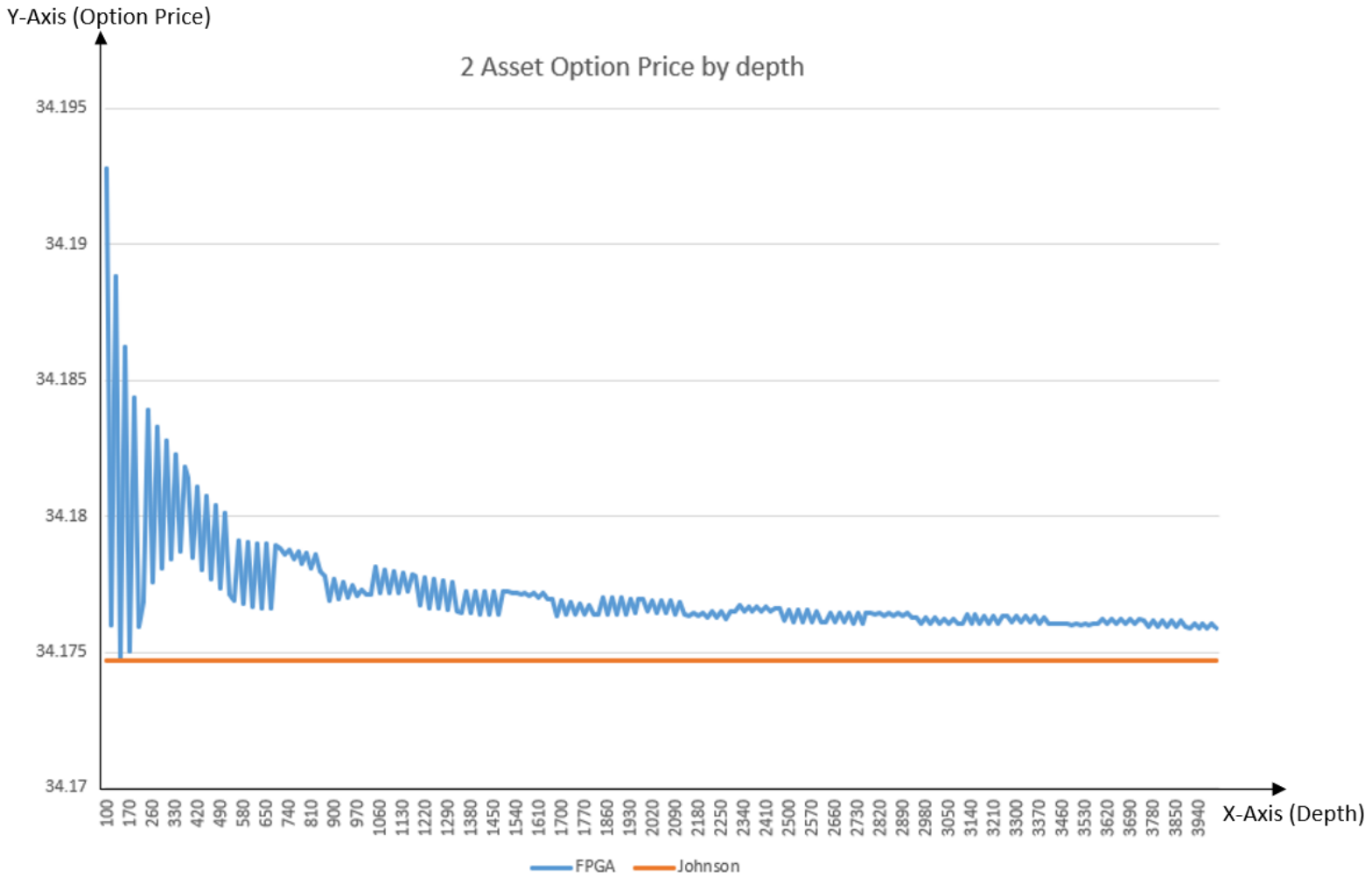


Figure 6.7: The precision of multiple executions of the two-asset option pricer (in blue) and the analytical solution (horizontal in orange) as per the Johnson analytical method[185].

Table 6.5 : Latency performance and pipelining impact as measured in seconds per option price. N/A indicates there was insufficient memory available to compute the option price for that combination of tree depth and number of compute blocks. The ”P“ case for the number of assets refers to the cases where an additional DDR bank was used to store intermediate values, which enables greater tree depths.

Number of assets	**Compute Blocks**	**Tree Depth**									
		10	20	30	40	50	100	200	500	1000	4096
2	1	0.000196	0.000224	0.000271	0.000373	0.000506	0.002124	0.013820	0.192855	1.505128	105.396617
	10	0.000241	0.000226	0.000288	0.000361	0.000453	0.001277	0.005054	0.040659	0.231130	11.104137
	32	0.000273	0.000315	0.000350	0.000382	0.000442	0.001309	0.004650	0.01445	0.147533	4.944934
3	2	0.000386	0.000621	0.001642	0.003742	0.006568	0.062300	0.692753	20.32738	290.2444	N/A
	8	0.000378	0.000531	0.001249	0.002606	0.005139	0.040773	0.365054	7.92796	N/A	N/A
	32	0.000592	0.000843	0.001398	0.002766	0.004061	0.033131	0.266418	N/A	N/A	N/A
4	1	0.000580	0.003021	0.013389	0.041318	0.101820	2.460709	N/A	N/A	N/A	N/A
	2	0.000554	0.002610	0.011145	0.033413	0.080289	1.284218	N/A	N/A	N/A	N/A
	8	0.000404	0.002423	0.010009	0.029173	0.068356	0.924900	N/A	N/A	N/A	N/A
5	1	0.000988	0.014712	0.095969	0.388596	1.186683	N/A	N/A	N/A	N/A	N/A
	3	0.000854	0.011581	0.071444	0.276264	0.809300	N/A	N/A	N/A	N/A	N/A
6	1	0.002438	0.068021	0.612620	N/A	N/A	N/A	N/A	N/A	N/A	N/A
7	1	0.004371	0.200949	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
4P	1	0.000300	0.003000	0.013700	0.044700	0.115300	2.533000	65.77430	N/A	N/A	N/A
5P	1	0.000700	0.012400	0.084400	0.353200	1.125127	46.83738	N/A	N/A	N/A	N/A
6P	1	0.001800	0.052600	0.489500	2.647200	10.11770	N/A	N/A	N/A	N/A	N/A
7P	1	0.004300	0.202400	2.661700	17.98030	82.13200	N/A	N/A	N/A	N/A	N/A

Table 6.6 : Stratix 10 resource utilization and maximum frequency in MHz (fMAX) for multiple tree depths and various numbers of compute blocks. The ”P“ case for the number of assets refers to the cases where an additional DDR bank was used to store intermediate values, which enables greater tree depths.

Number of assets	**Tree Depth**	**Compute Units**	**Logic utilization (in ALMs) in %**	**Total dedicated logic registers** in %	**Total RAM Blocks in %**	**Total DSP Blocks in %**	**fMAX**
2	1000	1	28	12	15	1	221
	4096	10	38	18	27	2	239
	4096	32	60	31	24	7	218
3	1000	2	24	12	55	1	228
	500	8	33	18	55	3	228
	250	32	70	43	60	9	228
4	180	1	26	13	58	1	204
	140	2	26	13	56	1	204
	100	8	39	20	96	3	204
5	68	1	26	14	59	1	201
	50	3	31	16	78	2	201
6	39	1	27	14	62	1	159
7	27	1	29	16	66	2	204
4P	335	1	25	13	35	1	213
5P	144	1	25	13	35	1	222
6P	82	1	25	13	35	1	221
7P	70	1	31	16	90	2	223

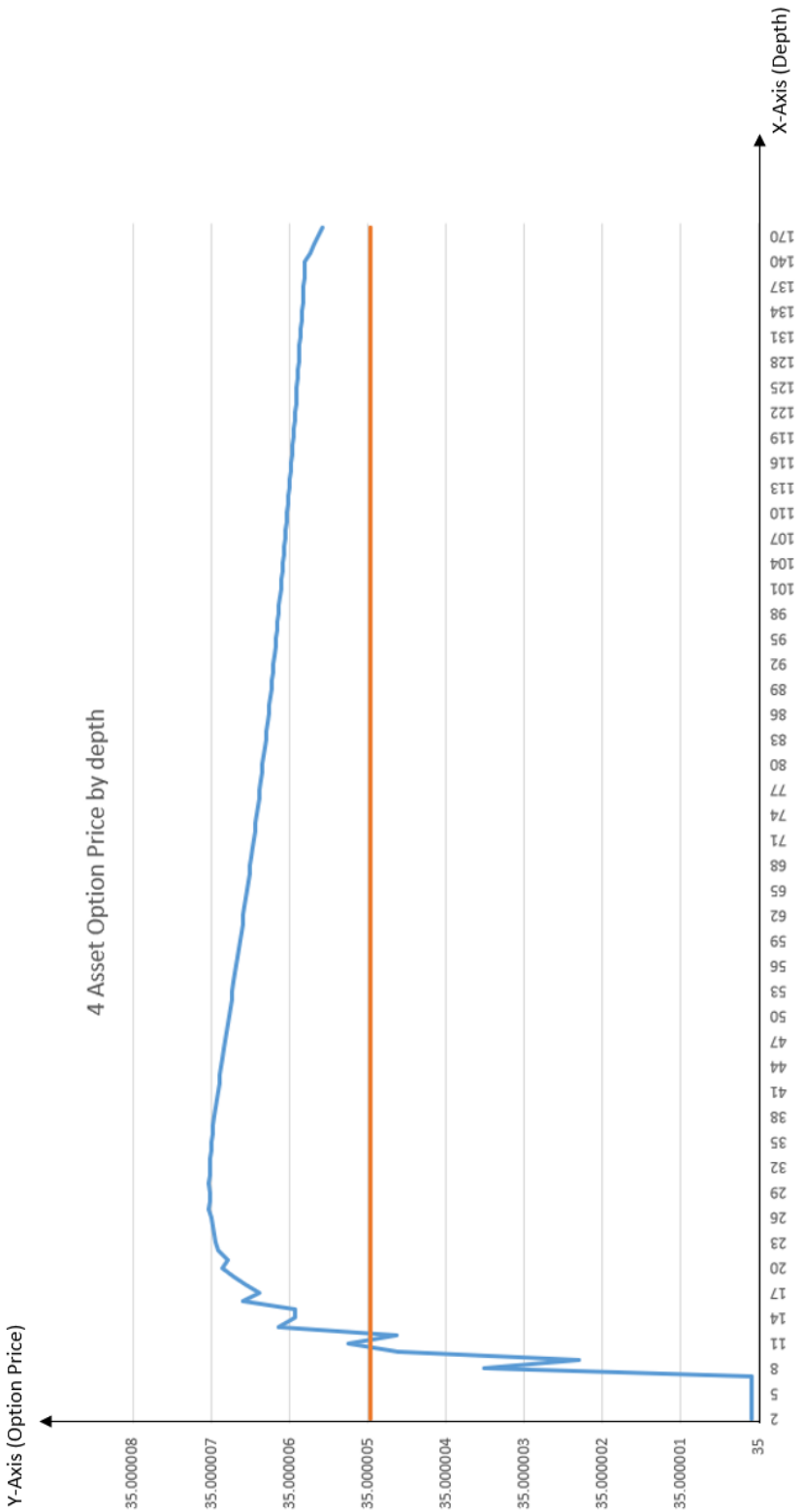


Figure 6.8: The precision of multiple executions of the four-asset option pricer (in blue) and the analytical solution (horizontal in orange) as per the Johnson analytical method[185].

6.5.2.2 Latency analysis

Along with accuracy, speed performance is of prime concern for a multi-asset option pricer. FPGAs are recognised for delivering better throughput and lower latency than an implementation on general purpose CPUs for a time investment[147]. Our implementation can achieve substational speedup for this comparison. We present the latencies for our implementation for between 2 and 7 assets (inclusive) in Table 6.5 on FPGA. The latencies measured, when compared to the same algorithm on CPU, demonstrated a maximum of 43 times speedup (in the case of 8 compute blocks for 4 assets) and a minimum of 8 times speedup (in the case of 6 assets at a tree depth of 25). Our software implementation was executed on an Intel Xeon 6128 processor with 16GB of RAM. Intel OpenCL implementations default to multi-threaded execution on CPUs [187].

We also present resource utilization metrics for the various configurations in Table 6.6. As can be noted from this table, our implementation is unable to support a large number of compute blocks for more than 3 assets. For 2 and 3 assets we can support 32 compute blocks. However, that reduces to 8 in the case of 4 assets, 3 compute blocks for 5 assets and only 1 compute block for greater than 5 assets. Also noteworthy is the sharp reduction in supported tree depths for 3 or more assets, reducing from a tree depth of 250 with 32 compute blocks in the case of 3 assets, down to a tree depth of 180 with 1 compute block for 4 assets.

Comparing results from Chapters 5 and 6 demonstrates a substantial latency increase from two-asset pricing (0.005 seconds, Table 5.3) with a pipeline depth of 50, to three-asset pricing (approximately 8 seconds, Table 6.5) with a pipeline depth of 8, both at a tree depth of 500. Despite leveraging multiple compute blocks, this 1600-fold latency increase results primarily from the exponential complexity growth when adding an additional asset, significantly increasing computational demands and memory access overhead. Additionally, the shallower pipeline (8 versus 50 stages) in the three-asset case further restricts latency improvements, emphasizing inherent scalability challenges related to higher dimensional computations and DDR memory bottlenecks.

6.6 Conclusions

Multi-asset option pricing based on Pascal's simplex is an efficient method for computing the fair price of an option based on an arbitrary number of underlying assets.

Combining a pipelined and parallel approach and also low latency DRAM enables hardware acceleration of a large number of assets. Here, we present such an acceleration architecture implemented using the OpenCL framework. This new implementation takes advantage of the channel's extension provided by the Intel FPGA SDK. This approach enabled us to compute option prices between large tree depths for a small number of assets (4096 deep for two assets), a large number per second (30 option prices per second for three assets at a tree depth of 100), and also a large number of assets (7 assets for a maximum tree depth of 55).

The implementation of Pascal's simplex on FPGA is not without complications. While we have been able to make significant use of the inherent parallelization capabilities of FPGA technology, the memory usage of the simplex approach proved to be a limiting factor with regards to the depth of the tree, recall the channel size grows exponentially as the number of assets and tree depth becomes larger. Future work could concentrate on further improving the performance figures of our approach with an improved memory management system. In the next chapter we will extend the architecture to pricing for American Options.

Chapter 7

Multi asset option pricing of American Options

This chapter describes a new method for pricing American options that utilizes the OneAPI framework and a Field-Programmable Gate Array (FPGA) device. By using a multinomial model based on Pascals Simplex, the method is able to price American options with improved performance over traditional methods. The use of the OneAPI framework allows for efficient parallelization of the computations on the FPGA, resulting in improved performance and faster pricing of the options. Our analysis indicates that, for 7 assets, the method can process approximately 53.74 options per second at the maximum tree depth of 15, showcasing its potential for real-world applications. The results of the method are validated against existing pricing models, demonstrating its accuracy and viability for use in real-world option pricing applications.

7.1 Introduction

American Options are derivatives giving the holder the right, but not the obligation, to buy or sell an asset at a predetermined price on or before a specified date. Option pricing calculates a financial option's theoretical value. FPGAs, programmable integrated circuits, can speed up option pricing through parallel processing capabilities.

Well-known option pricing models include Black-Scholes [17], Binomial model [18], and Monte Carlo simulation [19]. Option pricing on FPGA in-

volves implementing a pricing model on an FPGA device for real-time calculations [148], which is particularly beneficial for market makers and traders [188]. This chapter focuses on valuing options based on multiple underlying assets using a recombining multinomial tree approach.

The chapter is structured as follows: Section 7.2 reviews related research and gives an overview of financial options and the Binomial model, as well as the theory for pricing derivatives on multiple assets using recombining multinomial trees. Tree methods are particularly natural and useful for American options, as they enable a comparison of the value of holding onto the option with the immediate exercise value at each node. Section 7.3 outlines the hardware accelerator and OneAPI kernels for reconfigurable option pricing. Section 7.4 discusses the test environment and results, and Section 7.5 concludes.

7.2 Related Work

This section reviews existing research on accelerating option pricing for American options on multiple assets, emphasizing the authors' unique approach. The focus is on alternative methods for calculating option pricers on multiple asset options and tree pricing methods.

Previous research explores valuation formulas for diverse American options on multiple assets with both convex and nonconvex payoffs [189], Monte Carlo simulation techniques [190][191] with associated convergence issues [192], and quasi-Monte Carlo methods [193]. These studies also touch on binomial-tree-based American pricing, with applications for single assets [194][195][91][76].

Our novel approach uses FPGAs and Pascal's Simplex for American Options, offering superior performance and accuracy. It has been applied to two-asset European options and multinomial pricing for European options using FPGAs in the previous chapters. Notably, this method also presents significant advancements in terms of latency and energy efficiency over existing binomial-tree model implementations on FPGAs [81].

7.2.1 Differences between models

The Black-Scholes model and the binomial option model are used to price options but differ in assumptions and calculation methods. The Black-Scholes

model assumes continuous time and geometric Brownian motion, while the binomial option model is discrete-time and uses a tree-based approach. The former prices European options but extensions to the American options case are known, and the latter prices both European and American options.

7.2.2 American Binomial Option Price Model

The American and European binomial option pricing models differ in the exercise of the option. The European model assumes exercise at the expiration date, while the American model allows exercise any time before or at expiration. The American model is more complex due to early exercise decisions.

The American Binomial Option Price Model calculates expected option values at each time point until expiration. It is widely used for its simplicity and applicability to various option types and underlying assets.

7.2.3 Recombining Multinomial Trees Based on Pascal's Simplex

The method in [159] extends the binomial tree method for pricing options on multiple assets, reducing computational intensity. Using monomial ordering as described in Section 4.3, the method calculates option prices by determining the asset price vector at the final time on each of the nodes and computing option values iteratively. This approach, leveraging Pascal's Simplex, allows for efficient parallelization on FPGA, improved performance, accurate pricing, efficient hardware utilization, and support for early exercise in American options. For European options, the value at node ν at time $t = |\nu| < T$ is determined by successor nodes' values. For American options, the acceleration device generates asset price vectors, calculates option prices at the final time, and iteratively computes option values for non-increasing values of $|\nu|$ with $|\nu| \leq T$.

The key difference between European and American option valuation lies in the flexibility of exercise. For European options, the value of the option at each node is solely based on the expected future payoff discounted to the present. For American options, the value of holding onto the option is compared with the immediate exercise value, and the maximum of the two is used as the value of the option at that node.

7.3 Proposed FPGA architecture for financial options

7.3.1 Intel OneAPI and DPC++

Intel OneAPI is a software development platform that provides a range of tools, libraries, and APIs for various domains [196]. It allows developers to write code optimized for Intel hardware and deploy it across different platforms.

DPC++ is a programming model and language extension for parallel programming on CPUs, GPUs, and other accelerators. Based on SYCL, DPC++ enables developers to express data parallelism using C++ constructs and provides libraries for common parallel algorithms.

7.3.2 Design of the Multinomial Option Pricer

In our proposed FPGA architecture for financial options, we leveraged the inherent parallelism of Field-Programmable Gate Arrays (FPGAs) to significantly accelerate the pricing of American options. By implementing a multinomial model based on Pascal's Simplex on the FPGA, we were able to generate the option price simplex in parallel, overcoming the computational limitations of traditional methods and achieving superior performance in option pricing calculations.

The option pricer, illustrated in Figure 7.1 with inputs described in Table 7.1, uses a parallelized multinomial options pricing model for American options. The design consists of several kernels that perform specific operations and work together to achieve high performance and all computations are carried out with a 64-bit precision. Furthermore, a number of the kernels (stock value, payout, sorter, and option value) can be grouped together into a single compute block which then in turn can be duplicated to create a pipeline. This pipeline improves performance.

The first set of kernels computes the stock price tree. The producer kernel takes the initial stock price, volatility, and interest rate as inputs and generates a set of payoff values for each of the tree leaves, i.e., terminal nodes. As the computations progress backward up the tree, both the stock price values and

the option prices are iteratively computed at each node. The loop mux kernel computes the maximum value of the option at each node, taking into account the option exercise possibility. This reflects the American aspect of the pricing algorithm, where the flexibility to exercise early is embedded within the computation.

The next set of kernels (in the compute blocks) computes the option values. Combined, these kernels calculate the "hold-on" value of the option at a particular node in the tree. The option value kernel computes this value iteratively for each node in the price tree, using Equation 7.1 for European-style options:

$$\text{OptionValue} = \sum_{i=0}^{N-1} \left(\text{payout}[i] * \frac{1}{e^{r\frac{1}{N}}} * q_i \right) \tag{7.1}$$

For American-style options, the option value at each node is computed as the maximum of the "hold-on" value and the immediate exercise value, capturing the flexibility to exercise early.

The final set of kernels deals with output operations. Multi-tier buffer kernels handle buffering of the option values to ensure smooth data flow. The channel DDR kernels transfer data to DDR memory for storage, optimizing memory usage. Additionally, the sorter kernels organize the data in DDR memory, improving the performance of subsequent computations. Further details on these kernels are presented in Table 7.2.

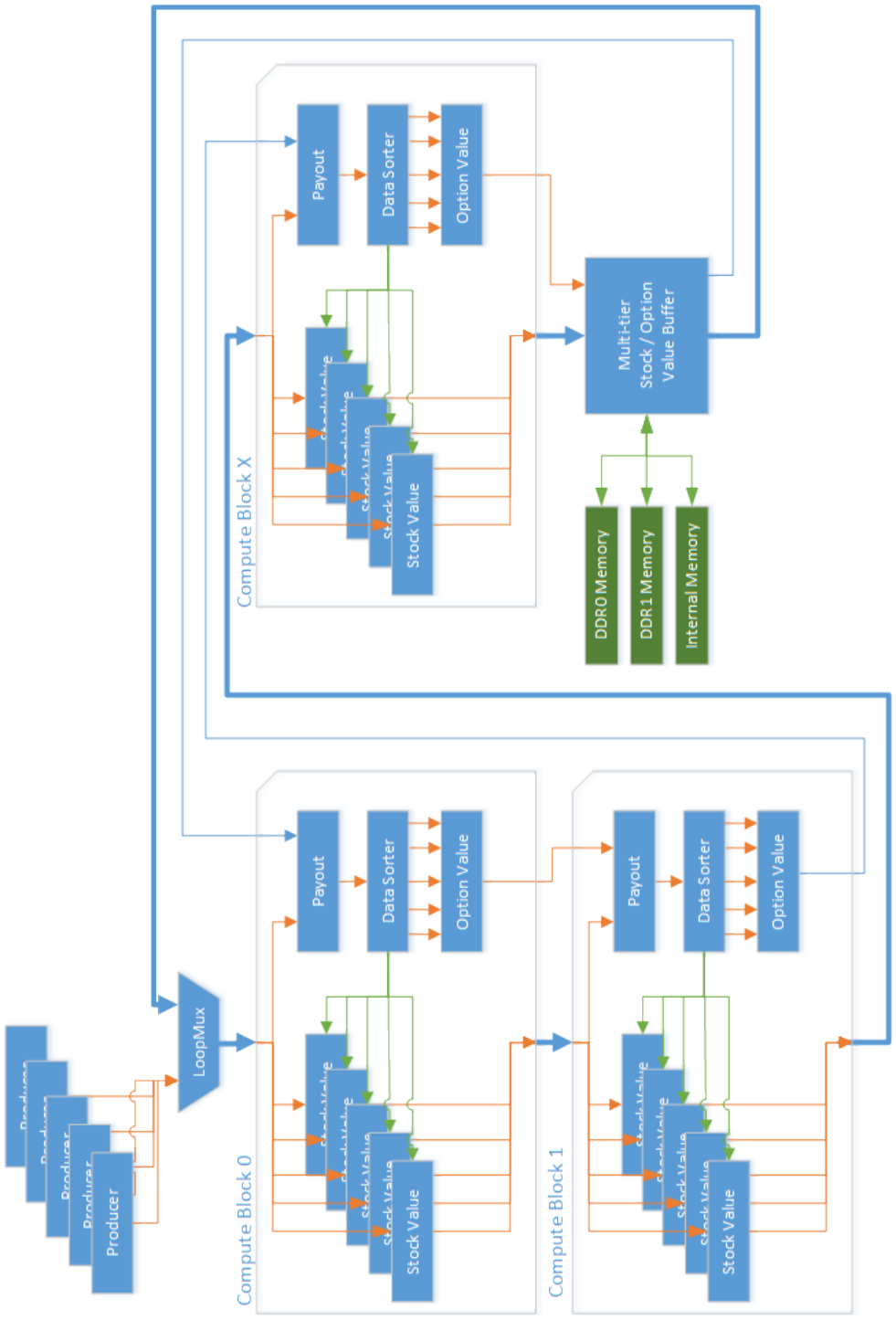


Figure 7.1: High-Level Architecture of the American Option Pricer

Table 7.1 : Inputs to the FPGA. These inputs are precomputed from the traditional option price parameters, such as ρ, σ, and r. Section II in [159] provides detailed explanations of the direction vectors $d_{i,j}$ and their roles.

Kernel	**Description**
N	Tree depth
k	Number of assets
Operation	Specifies whether to compute a call/put option on the maximum or minimum of the asset prices. This is selected by the user.
end_node_values	Defined as a $(k+1)$-element vector containing $\{(d_{k,0})^N Z_k, \cdots, (d_{0,0})^N Z_0\}$, where the asset prices Z_X (representing S_0 for each asset) are scaled by the first elements in the direction vector to the power of N.
init_step	Defined as a kxk element matrix containing $\left\{\frac{d_{k-1,k}}{d_{k-1,k-1}}, \cdots, \frac{d_{0,1}}{d_{0,0}}\right\}$. This matrix specifies the initialization steps for the asset transitions at each time step.
mult_coeff	A k-element multiplicative coefficient vector containing $\{\frac{1}{R}q_k, \frac{1}{R}q_{k-1}, \cdots, \frac{1}{R}q_0\}$, where R is defined as $e^{r(1/N)}$, and r is the risk-neutral interest rate. These coefficients are critical for discounting and probability weighting during backward computations.
share_coeff	A k-element share coefficient vector containing $\{\frac{1}{d_{k,1}}, \cdots, \frac{1}{d_{0,1}}\}$. These coefficients help adjust the weight of each direction vector for the recombining multinomial tree.
strike_vector	A k-element vector containing the strike prices for each asset. Strike prices may differ per asset.

Table 7.2: Detailed Overview of Kernels Employed in the Option Pricing Algorithm

Kernel	**Description**	**Kernel**	**Description**
producer	Generates a set of payoff values for each of the tree leaves (i.e., terminal nodes) based on the input option parameters.	loopMux	Computes the payoff value for each node of the tree by recursively traversing the tree and applying the option pricing formula. Returns option price to host.
stockValue	Computes the stock price for each node of the tree based on the initial stock price and the tree dept.	optionValue	Compute the value of an American option at the current point in the tree. Exercise early if appropriate.
dataSorter	Reorders the tree nodes to optimize memory access patterns and ensure efficient communication between processing units. This involves reorganizing the pre-sorted tree data according to depth and processing status.	payout	Computes the value of the option at each node of the multinomial tree if the option is exercised, considering the payoff function (e.g., intrinsic value for American options).
writeMultiTierBuffer	Writes the reordered tree data to a multi-tier buffer.	readMultiTierBuffer	Reads the reordered tree data from the multi-tier buffer.
channelDDR	Reads data from a pipe and writes it to a designated buffer in DDR memory.	sorterChannelBuffer	Sorts the tree data in a designated buffer in DDR memory using a FIFO buffer.

7.4 Results

We used the Intel DevCloud, a cloud-based platform with access to various Intel hardware configurations, including FPGAs, for development and testing purposes. This allowed us to run experiments on a powerful FPGA device without any hardware investment.

We used the following versions: Target Family, Device, Board Stratix 10, 1SX280HN2F43E2VG, pac_s10_usm. The SYCL version used was 2023.0.0, and the Quartus version was 19.2.0 Build 57 Pro.

7.4.1 Accuracy Analysis

We evaluate our accuracy against the results presented in [197]. In that paper the author presents a number of simulation prices for multiple assets, up to 15, for multiple different approaches, e.g, Finite Difference (FD)[198]. In Figure 7.2, we present our results compared to results from [197] for Min-Puts with parameter values: $S_i(0) = 100$, $K = 100$, $T = 0.5$, $r = 0.06$, and $\sigma_i = 0.6$ which are the same as the parameters in the referenced papers. Our results are broadly in line with the FD and MC results. However, we observed an average variance of between ± 0.05 and ± 0.3.

We also took note of the early exercise depth which is dependent on the input values, for some input values there was an early exercise depth, for others a late exercise depth. We have also tested our implementation for up to seven assets, which yields graphs that are similarly aligned but not exact matches to the finite difference and Monte Carlo results.

7.4.2 Latency Analysis

Our measurements for the execution of our implementation with two assets indicate that the relationship between latency and tree depth appears to be polynomial. Specifically, the latency more than doubles as the tree depth increases from 500 to 1000 and continues to grow at a similar rate for depths beyond 1000. At a tree depth of 16,000, the latency exceeds 220 million microseconds, equivalent to more than 220 seconds.

We also plotted the latency for each of the supported asset cases, i.e., 2 to 7, for the maximum depth supported for 7 assets which is a tree of depth 15.

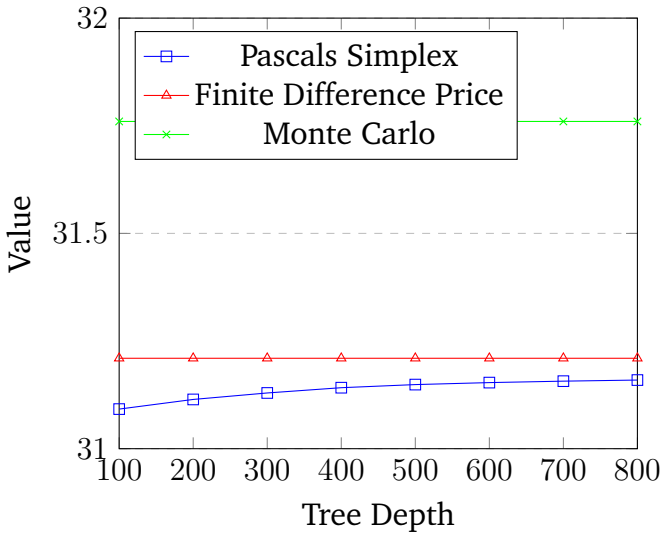


Figure 7.2: Comparison of Pascals Simplex, Finite Difference Price, and Monte Carlo methods for valuing an option on three financial assets.

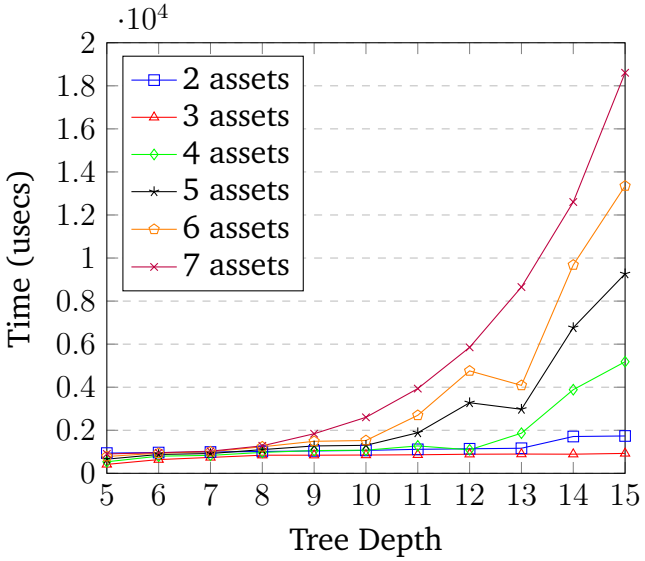


Figure 7.3: Latency Comparison for American Option Pricing with Two to Seven Underlying Assets

This is presented in Figure 7.3. We observe a similar trend in the latency for each of the 3-7 case as the 2 asset case.

7.4.3 Resource Utilization Analysis

Table 7.3 provides information about the computational requirements of the option pricing implementation for different numbers of assets, taking into account the limitations of DDR memory. The table shows that, as the number of assets increases from two to seven, the number of Application Logic Units

(ALUTs), Flip-Flops (FFs), Random Access Memories (RAMs), Digital Signal Processors (DSPs), and Multipliers (MLABs) also increases, indicating that the computational resources required for the implementation increase as the complexity of the pricing model increases. However, the implementation only supports up to seven assets due to the constraints of DDR memory. Hence, we never require 100% of the FPGA resources. Based on resource utilization for the seven-asset implementation, the Intel Early Power Estimator (EPE) reported an estimated power consumption of **47.624 W**.

Table 7.3: Resource Utilization (as a Percentage of Total) for Option Pricing Across Different Numbers of Underlying Assets on the Intel FPGA PAC Stratix 10 SX D5005.

Assets	ALUTs (%)	FFs (%)	RAMs (%)	DSPs (%)	MLABs (%)
2	38%	36%	38%	32%	22%
3	40%	38%	41%	32%	24%
4	42%	40%	44%	32%	31%
5	45%	42%	48%	32%	38%
6	49%	44%	51%	33%	48%
7	49%	44%	51%	33%	48%

7.4.4 Comparative Analysis

In comparison to other FPGA-based implementations for American option pricing, our proposed architecture uniquely supports multiple assets and leverages the inherent parallelism of FPGAs with a multinomial model based on Pascal's Simplex. This approach significantly outperforms traditional methods. For instance, the LSMC method on a Xilinx Virtex-4 XC4VSX55[91] and the binomial option pricing model using OpenCL[76] both report a 20 times speed-up and the ability to evaluate over 2000 options/s with less than 20W power usage. Adapting this implementation for two assets necessitates calculating all possible outcome combinations, which could potentially lead to a computation speed of approximately 3 options per second for tree depth $N = 1024$ compared to the simplex approach which achieves 40 options per second for the same N.

Comparison of OpenCL and oneAPI Implementations. While a direct implementation of the seven-asset case in OpenCL was not conducted in this study, the oneAPI implementation can be compared qualitatively to the ear-

lier OpenCL work in Chapter 5 and Chapter 6. The oneAPI design achieved a throughput of **53.74 options/sec** for the seven-asset configuration, with an estimated power consumption of **47.624 W**. In contrast, the OpenCL implementation for two assets achieved 6250 options/sec at 39.3 W. Given the increased complexity of multi-asset pricing, especially with deeper pipelines and more memory interaction, a similar seven-asset implementation in OpenCL would likely require additional design effort around memory management and DDR access, potentially leading to lower resource efficiency. Future work may involve implementing the seven-asset case in OpenCL to enable a direct performance and power comparison.

CPU Execution and Scalability. To evaluate the portability and scalability of the oneAPI implementation, the kernel was executed on an Intel Xeon Platinum 8272CL CPU (2.60 GHz) using the DPC++ compiler on the Intel DevCloud. For the seven-asset configuration, the implementation achieved a throughput of approximately **5.2 options/sec** on a single core. Scaling across multiple threads showed near-linear improvements up to 8 threads, reaching **22.1 options/sec** on 8 cores, and **28.4 options/sec** on 16 cores. Although this is lower than the FPGA result of 53.74 options/sec, it demonstrates the functional portability of the oneAPI implementation and highlights the trade-offs between CPU parallelism and FPGA pipelining.

7.5 Conclusion

The use of FPGAs for American multinomial option pricing can significantly expedite the calculation process. By leveraging parallel processing capabilities, FPGAs reduce the time required for determining theoretical option values, a crucial tool for risk management and informed investment decisions.

Intel OneAPI is a robust platform suitable for developing FPGA applications, including option pricing models. OneAPI allows developers to use a common set of tools, libraries, and APIs, optimized for Intel hardware and deployable across various platforms.

The use of FPGAs and OneAPI provides a fast and efficient solution for pricing American multinomial options. This combination is valuable to market makers and traders who need to quickly assess the value of these financial instruments. With ongoing research and development, there is potential to unlock

even greater performance and precision, solidifying the role of FPGA in option pricing within the financial industry.

In the next chapter we will review the outcomes from this thesis and present what future research directions can be taken based on this work.

Chapter 8

Conclusion and Future Work

This thesis has demonstrated the considerable potential of FPGA architectures for Pascal's Simplex-based multinomial option pricing, particularly in improving performance, scalability, and energy efficiency compared to traditional computational platforms.

The research began by establishing a foundation in option pricing, FPGA architectures, and parallel computing frameworks such as OpenCL and OneAPI. Recognizing the significant computational demands of financial models, it highlighted the importance of hardware acceleration to address these challenges, setting the stage for practical applications.

A review of existing FPGA applications in option pricing revealed notable advantages in speed and energy efficiency. However, a gap was identified: the lack of research into FPGA implementations for Pascal's Simplex-based multinomial models. This observation served as the basis for the novel contributions that followed.

The core of this work focused on implementing FPGA architectures for both two-asset and multi-asset option pricing. A theoretical framework was developed to translate multinomial tree methods into hardware implementations, optimizing Pascal's Simplex for efficient FPGA deployment. This groundwork laid the foundation for significant performance improvements in financial computations.

In implementing the two-asset FPGA architecture, the research achieved a performance of 200 option value calculations per second for a simulation com-

prising 500 time steps. Here, "time step" refers to the number of discrete intervals in the multinomial tree simulation. This performance metric demonstrated how an FPGA's pipelined and parallel computing capabilities could outperform conventional methods in both speed and accuracy. The findings offered a high-performance alternative for complex financial computations, improving latency and resource utilization.

Building on this success, the analysis extended to multi-asset option pricing, where the FPGA architecture achieved speedups of up to 43 times faster than traditional methods. This solution proved to be not only faster but also highly scalable, allowing for the pricing of multiple assets without a proportional increase in computational cost. Despite the increased complexity of the model, FPGA resource utilization remained efficient, underscoring its capability to handle large-scale financial applications.

Further exploration into FPGA implementation for American option pricing yielded additional performance enhancements. By leveraging Intel's OneAPI framework, the FPGA architecture priced up to 53.74 options per second for seven underlying assets at maximum tree depth. These results demonstrated the suitability of FPGA technology for real-time financial analytics, where both speed and accuracy are paramount.

Overall, the research contributed a novel approach to option pricing by harnessing the power of FPGAs and optimized hardware implementations. It filled a gap in existing research and showcased how advanced hardware acceleration can add to computational finance, opening new possibilities for future exploration.

A critical area for future research is addressing the growth in FIFO depths, particularly the exponential (in the case of the number of assets) or polynomial scaling (in the case of tree depth). This growth significantly impacts memory usage and performance, making the development of optimized FIFO strategies essential for scalability in FPGA-based systems.

Another promising avenue is to explore the MC-Tree method [199]. For single-asset options, the MC-Tree method combines Monte Carlo (MC) simulations with a recombining binomial tree based on Pascal's triangle. For multi-asset options, the MC-Tree method leverages recombining multinomial trees based on Pascal's simplex. Future research could extend this method to higher-dimensional cases, focusing on improving accuracy by optimizing entropy and

managing computational complexity.

Additionally, the techniques developed in this thesis could be applied to other domains beyond option pricing. For instance:

1. Efficient computation of event probabilities for multivariate Gaussian distributions.
2. Multivariate polynomial system solving and multivariate polynomial optimization [200].

Looking to the future, the integration of FPGA-based solutions into real-world trading platforms presents a significant opportunity. These platforms require real-time decision-making and high-frequency trading capabilities, where both speed and accuracy are critical. By deploying FPGAs in live trading environments, financial institutions could benefit from faster, more energy-efficient computations, enabling quicker responses to market movements and better risk management.

Additionally, hybrid-core architectures combining FPGAs with CPUs or GPUs present another exciting research direction. Such systems could further enhance performance and scalability for complex financial models, pushing the boundaries of what is achievable in computational finance. In addition, future exploration of emerging machine Learning techniques could be explored.

Ultimately, this thesis has laid the foundation for further research into specialized hardware for the financial industry. By demonstrating the effectiveness of FPGA-based systems for complex computations, it has opened new avenues for high-performance trading platforms and real-time financial analytics.

References

[1] G. Poitras, "The early history of option contracts," in *Vinzenz Bronzin's Option Pricing Models: Exposition and Appraisal* (W. Hafner and H. Zimmermann, eds.), pp. 487–518, Springer, 2009.

[2] Z. Zhang, *Option Pricing and Valuation of Contingent Cash Flow*, pp. 119–150. Singapore: Springer Nature Singapore, 2023.

[3] R. Harris, "The institutional dynamics of early modern Eurasian trade: The commenda and the corporation," *Journal of Economic Behavior & Organization*, vol. 71, no. 3, pp. 606–622, 2009.

[4] E. Stringham, "The extralegal development of securities trading in seventeenth-century Amsterdam," *The Quarterly Review of Economics and Finance*, vol. 43, no. 2, pp. 321–344, 2003.

[5] J. E. Finnerty, "The Chicago board options exchange and market efficiency," *Journal of Financial and Quantitative Analysis*, vol. 13, no. 1, pp. 29–38, 1978.

[6] "CBOE's beginnings in the words of our founding president." `https://www.cboe.com/insights/posts/cboes-beginnings-in-the-words-of-our-founding-president/`. (Accessed on 08/26/2023).

[7] R. C. Merton, "Paul Samuelson and financial economics," *The American Economist*, vol. 50, no. 2, pp. 9–31, 2006.

[8] P. G. Zhang, *Exotic options: a guide to second generation options*. World scientific, 1997.

[9] P. H. Ritchken, "On pricing barrier options," *The J. of Derivatives*, vol. 3, no. 2, 1995.

[10] S. Browne, "Reaching goals by a deadline: Digital options and continuous-time active portfolio management," *Advances in Applied Probability*, vol. 31, no. 2, pp. 551–577, 1999.

[11] L. C. G. Rogers and Z. Shi, "The value of an Asian option," *Journal of Applied Probability*, vol. 32, no. 4, pp. 1077–1088, 1995.

[12] A. Conze and Viswanathan, "Path dependent options: The case of look-back options," *The Journal of Finance*, vol. 46, no. 5, pp. 1893–1907, 1991.

[13] P. Ouwehand and G. West, "Pricing rainbow options," *Wilmott magazine*, vol. 5, pp. 74–80, 2006.

[14] R. Geske, "The valuation of compound options," *Journal of Financial Economics*, vol. 7, no. 1, pp. 63–81, 1979.

[15] K. Ramaswamy and S. M. Sundaresan, "The valuation of options on futures contracts," *The Journal of Finance*, vol. 40, no. 5, pp. 1319–1340, 1985.

[16] J. Hull and A. White, "How to value employee stock options," *Financial Analysts Journal*, vol. 60, no. 1, pp. 114–119, 2004.

[17] F. Black and M. Scholes, "The pricing of options and corporate liabilities," *Journal of Political Economy*, vol. 81, no. 3, pp. 637–654, 1973.

[18] J. C. Cox, S. A. Ross, and M. Rubinstein, "Option pricing: A simplified approach," *Journal of Financial Economics*, vol. 7, no. 3, pp. 229–263, 1979.

[19] P. P. Boyle, "Options: A Monte Carlo approach," *Journal of Financial Economics*, vol. 4, no. 3, pp. 323–338, 1977.

[20] J. Zhao, M. Davison, and R. M. Corless, "Compact finite difference method for American option pricing," *Journal of Computational and Applied Mathematics*, vol. 206, no. 1, pp. 306–321, 2007.

[21] K. K. Langat, J. I. Mwaniki, and G. K. Kiprop, "Pricing options using trinomial lattice method," *Journal of Finance and Economics*, vol. 7, no. 3, pp. 81–87, 2019.

[22] N. H. Bingham and R. Kiesel, *Risk-neutral valuation: Pricing and hedging of financial derivatives*. Springer Science & Business Media, 2013.

[23] B. Peccerillo, M. Mannino, A. Mondelli, and S. Bartolini, "A survey on hardware accelerators: Taxonomy, trends, challenges, and perspectives," *Journal of Systems Architecture*, vol. 129, p. 102561, 2022.

[24] J. Nurmi, *Processor design: system-on-chip computing for ASICs and FPGAs*. Springer Science & Business Media, 2007.

[25] M. Castro, D. L. Vilarino, Y. Torres, and D. R. Llanos, "The role of field-programmable gate arrays in the acceleration of modern high-performance computing workloads," *Computer*, vol. 57, pp. 66–76, jul 2024.

[26] G. E. Moon, H. Kwon, G. Jeong, P. Chatarasi, S. Rajamanickam, and T. Krishna, "Evaluating spatial accelerator architectures with tiled matrix-matrix multiplication," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 4, pp. 1002–1014, 2021.

[27] B. D. Sutter, P. Raghavan, and A. Lambrechts, "Coarse-grained reconfigurable array architectures," *Handbook of signal processing systems*, pp. 427–472, 2019.

[28] B. Peccerillo, M. Mannino, A. Mondelli, and S. Bartolini, "A survey on hardware accelerators: Taxonomy, trends, challenges, and perspectives," *Journal of Systems Architecture*, vol. 129, p. 102561, 2022.

[29] J. Yu, G. Lemieux, and C. Eagleston, "Vector processing as a soft-core CPU accelerator," in *Proceedings of the 16th international ACM/SIGDA symposium on Field programmable gate arrays*, pp. 222–232, 2008.

[30] A. Vajda and A. Vajda, "Multi-core and many-core processor architectures," *Programming Many-Core Chips*, pp. 9–43, 2011.

[31] W. J. Dally, S. W. Keckler, and D. B. Kirk, "Evolution of the graphics processing unit (GPU)," *IEEE Micro*, vol. 41, no. 6, pp. 42–51, 2021.

[32] A. Becher, L. BG, D. Broneske, T. Drewes, B. Gurumurthy, K. Meyer-Wegener, T. Pionteck, G. Saake, J. Teich, and S. Wildermann, "Integration of FPGAs in database management systems: challenges and opportunities," *Datenbank-Spektrum*, vol. 18, pp. 145–156, 2018.

[33] J. Casper and K. Olukotun, "Hardware acceleration of database operations," in *Proceedings of the 2014 ACM/SIGDA international symposium on Field-programmable gate arrays*, pp. 151–160, 2014.

[34] R. A. Bridges, N. Imam, and T. M. Mintz, "Understanding GPU power: A survey of profiling, modeling, and simulation methods," *ACM Computing Surveys (CSUR)*, vol. 49, no. 3, pp. 1–27, 2016.

[35] M. Bhattacharya, P. Calafiura, T. Childers, M. Dewing, Z. Dong, O. Gutsche, S. Habib, X. Ju, M. Kirby, K. Knoepfel, *et al.*, "Portability: a necessary approach for future scientific software," *arXiv preprint arXiv:2203.09945*, 2022.

[36] M. Owaida, G. Alonso, L. Fogliarini, A. Hock-Koon, and P.-E. Melet, "Lowering the latency of data processing pipelines through FPGA based hardware acceleration," *Proceedings of the VLDB Endowment*, vol. 13, no. 1, pp. 71–85, 2019.

[37] J. C. Schabel and P. D. Franzon, "Exploring the tradeoffs of application-specific processing," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 8, no. 3, pp. 531–542, 2018.

[38] M. Sefton, I. Rönkkö, and M. Bellgran, "Hardware start-ups in the scale-up process of production–A mapping of challenges," in *SPS2020*, pp. 378–391, IOS Press, 2020.

[39] F. Butollo, C. Staritz, F. Maile, and T. Wuttke, "The end of globalized production? Supply-chain resilience, technological sovereignty, and enduring global interdependencies in the post-pandemic era," *Critical Sociology*, p. 08969205241239872, 2024.

[40] M. E. Jennex, D. Amoroso, and O. Adelakun, "E-commerce infrastructure success factors for small companies in developing economies," *Electronic Commerce Research*, vol. 4, pp. 263–286, 2004.

[41] A. Alqoud, D. Schaefer, and J. Milisavljevic-Syed, "Industry 4.0: a systematic review of legacy manufacturing system digital retrofitting," *Manufacturing Review*, vol. 9, p. 32, 2022.

[42] N. R. Council, C. on Physical Sciences, Mathematics, Applications, C. Science, T. Board, C. on Intellectual Property Rights, and the Emerging Information Infrastructure, *The digital dilemma: Intellectual property in the information age*. National Academies Press, 2000.

[43] U. Farooq, Z. Marrakchi, H. Mehrez, U. Farooq, Z. Marrakchi, and H. Mehrez, "FPGA architectures: An overview," *Tree-Based Heteroge-*

neous FPGA Architectures: Application Specific Exploration and Optimization, pp. 7–48, 2012.

[44] Y.-D. Bae, S.-I. Park, and I.-C. Park, "A single-chip programmable platform based on a multithreaded processor and configurable logic clusters," *IEEE Journal of Solid-State Circuits*, vol. 38, no. 10, pp. 1703–1711, 2003.

[45] V. Betz and J. Rose, "FPGA routing architecture: Segmentation and buffering to optimize speed and density," in *Proceedings of the 1999 ACM/SIGDA seventh international symposium on Field programmable gate arrays*, pp. 59–68, 1999.

[46] A. Boutros and V. Betz, "FPGA architecture: Principles and progression," *IEEE Circuits and Systems Magazine*, vol. 21, no. 2, pp. 4–29, 2021.

[47] G. Chatziparaskevas, A. Brokalakis, and I. Papaefstathiou, "An FPGA-based parallel processor for Black-Scholes option pricing using finite differences schemes," in *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 709–714, EDA Consortium, 2012.

[48] B. Kitchenham, O. P. Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, "Systematic literature reviews in software engineering–a systematic literature review," *Information and software technology*, vol. 51, no. 1, pp. 7–15, 2009.

[49] D. Budgen and P. Brereton, "Performing systematic literature reviews in software engineering," in *Proceedings of the 28th international conference on Software engineering*, pp. 1051–1052, 2006.

[50] A. O. Mahony and E. Popovici, "A systematic review of blockchain hardware acceleration architectures," in *2019 30th Irish signals and systems conference (ISSC)*, pp. 1–6, IEEE, 2019.

[51] C. Zhang, H. Yu, Y. Zhou, and H. Jiang, "High-performance and energy-efficient FPGA-GPU-CPU heterogeneous system implementation," in *Advances in Parallel & Distributed Processing, and Applications* (H. R. Arabnia, L. Deligiannidis, M. R. Grimaila, D. D. Hodson, K. Joe, M. Sekijima, and F. G. Tinetti, eds.), (Cham), pp. 477–492, Springer International Publishing, 2021.

[52] A. O. Mahony, G. Zeidan, B. Hanzon, and E. Popovici, "A parallel and pipelined implementation of a Pascal-Simplex based multi-asset option pricer on FPGA using OpenCL," *Microprocessors and Microsystems*, vol. 90, p. 104508, 2022.

[53] A. M. Monteiro and A. A. Santos, "Parallel computing in finance for estimating risk-neutral densities through option prices," *Journal of Parallel and Distributed Computing*, vol. 173, pp. 61–69, 2023.

[54] E. Panova, V. Volokitin, A. Gorshkov, and I. Meyerov, "Black-Scholes option pricing on Intel CPUs and GPUs: Implementation on SYCL and optimization techniques," in *Supercomputing: 8th Russian Supercomputing Days, RuSCDays 2022, Moscow, Russia, September 26–27, 2022, Revised Selected Papers*, pp. 48–62, Springer, 2022.

[55] A. O. Mahony, G. Zeidan, B. Hanzon, and E. Popovici, "A parallel and pipelined implementation of a Pascal-Simplex based two asset option pricer on FPGA using OpenCL," in *2020 IEEE Nordic Circuits and Systems Conference (NorCAS)*, pp. 1–6, IEEE, 2020.

[56] R. Bruce, J. Setoain, R. Chamberlain, M. Devlin, and R. M. Badia, "Implementing closed-form expressions on FPGAs using the NAL, with comparison to CUDA GPU and Cell BE implementations," 2023.

[57] C. Xiong and N. Xu, "Performance comparison of BLAS on CPU, GPU and FPGA," in *2020 IEEE 9th Joint International Information Technology and Artificial Intelligence Conference (ITAIC)*, vol. 9, pp. 193–197, 2020.

[58] Y. S. Elshakhs, K. M. Deliparaschos, T. Charalambous, G. Oliva, and A. Zolotas, "A comprehensive survey on Delaunay triangulation: Applications, algorithms, and implementations over CPUs, GPUs, and FPGAs," *IEEE Access*, vol. 12, pp. 12562–12585, 2024.

[59] E. Nurvitadhi, D. Sheffield, J. Sim, A. Mishra, G. Venkatesh, and D. Marr, "Accelerating binarized neural networks: Comparison of FPGA, CPU, GPU, and asic," in *2016 International Conference on Field-Programmable Technology (FPT)*, pp. 77–84, IEEE, 2016.

[60] R. C. Merton, "Theory of rational option pricing," *The Bell Journal of economics and management science*, pp. 141–183, 1973.

[61] S. Tandon, "A programmable architecture for real-time derivative trading," *Master's Thesis, University of Edinburgh*, 2003.

[62] R. Baxter, S. Booth, M. Bull, G. Cawood, J. Perry, M. Parsons, A. Simpson, A. Trew, A. McCormick, G. Smart, *et al.*, "Maxwell-a 64 FPGA supercomputer," in *Second NASA/ESA Conference on Adaptive Hardware and Systems (AHS 2007)*, pp. 287–294, IEEE, 2007.

[63] D. B. Thomas and W. Luk, "Sampling from the multivariate Gaussian distribution using reconfigurable hardware," in *15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2007)*, pp. 3–12, IEEE, 2007.

[64] J. Castillo, J. L. Bosque, E. Castillo, P. Huerta, and J. I. Martínez, "Hardware accelerated Monte Carlo financial simulation over low cost FPGA cluster," in *2009 IEEE International Symposium on Parallel & Distributed Processing*, pp. 1–8, IEEE, 2009.

[65] C. Patel, M. Srikanth, K. C. Kumar, and S. Sivanantham, "Monte-Carlo Black-Scholes implementation using OpenCL standard," *Indian journal of science and technology*, vol. 8, pp. 1–5, 2016.

[66] C. Choo, L. Malhotra, and A. Munjal, "FPGA-based design of Black Scholes financial model for high performance trading," *Journal of information and communication convergence engineering*, vol. 11, no. 3, pp. 190–198, 2013.

[67] G. Inggs, S. Fleming, D. B. Thomas, and W. Luk, *Is High Level Synthesis Ready for Business? An Option Pricing Case Study*, pp. 97–115. Cham: Springer International Publishing, 2015.

[68] M. A. B. Guerrero, *MultiCorePricer: A Monte-Carlo Pricing Engine for Financial Derivatives*. PhD thesis, Master Thesis Project, Swiss Federal Institute of Technology Zurich, 2015.

[69] N. K. Pham, K. M. M. Aung, and A. Kumar, "Automatic framework to generate reconfigurable accelerators for option pricing applications," in *2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, pp. 1–8, IEEE, 2016.

[70] G. Inggs, "Algorithmic trading: A brief, computational finance case study on data centre FPGAs," *arXiv preprint arXiv:1607.05069*, 2016.

[71] L. Ma, *Low power and high performance heterogeneous computing on FPGAs*. PhD thesis, Politecnico di Torino, 2019.

[72] M. C. S. de Castro, "Implementação e avaliação de co-processadores para black-Scholes em FPGA," *Cadernos do IME-Série Informática*, vol. 42, p. 7, 2019.

[73] A. Rodrigues and V. Moreira, "Multicorepricer: A Monte-Carlo pricing engine for financial derivatives 2," *Topics in Computational Finance*, p. 27, 2019.

[74] Y. Li, L. Zhang, Y. Dai, and Y. Sun, "A word-length optimized parallel framework for accelerating option pricing model," in *2021 IEEE 23rd Int Conf on High Performance Computing & Communications; 7th Int Conf on Data Science & Systems; 19th Int Conf on Smart City; 7th Int Conf on Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys)*, pp. 654–661, IEEE, 2021.

[75] S. Wang, H. Huan, S. F. Wong, and J. Yen, "FPGA based implied volatility calculation with multi-section method," in *2022 IEEE 20th International Conference on Industrial Informatics (INDIN)*, pp. 507–514, IEEE, 2022.

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

[77] Q. Jin, D. B. Thomas, W. Luk, and B. Cope, "Exploring reconfigurable architectures for binomial-tree pricing models," in *International Workshop on Applied Reconfigurable Computing*, pp. 245–255, Springer, 2008.

[78] C. Wynnyk and M. Magdon-Ismail, "Pricing the American option using reconfigurable hardware," in *2009 International Conference on Computational Science and Engineering*, vol. 2, pp. 532–536, IEEE, 2009.

[79] Q. Jin, D. B. Thomas, W. Luk, and B. Cope, "Exploring reconfigurable architectures for tree-based option pricing models," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 2, no. 4, p. 21, 2009.

[80] P. Fabry and D. Thomas, "Efficient reconfigurable architecture for pricing exotic options," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 10, no. 4, pp. 1–22, 2017.

[81] A. Tavakkoli and D. B. Thomas, "Low-latency option pricing using systolic binomial trees," in *2014 International Conference on Field-Programmable Technology (FPT)*, pp. 44–51, 2014.

[82] A. Tavakkoli and D. B. Thomas, "A high-level design framework for the automatic generation of high-throughput systolic binomial-tree solvers," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 2, pp. 341–354, 2017.

[83] U. I. Minhas, R. Woods, and G. Karakonstantis, "Exploring functional acceleration of OpenCL on FPGAs and GPUs through platform-independent optimizations," in *International Symposium on Applied Reconfigurable Computing*, pp. 551–563, Springer, 2018.

[84] F. A. Longstaff and E. S. Schwartz, "Valuing american options by simulation: a simple least-squares approach," *The review of financial studies*, vol. 14, no. 1, pp. 113–147, 2001.

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

[86] J. K. Anlauf, "Pricing of derivatives by fast, hardware-based Monte-Carlo simulation," tech. rep., Working paper, Institute for Information, University of Bonn. 53117, 2005.

[87] J. A. Bower, D. B. Thomas, W. Luk, and O. Mencer, "A reconfigurable simulation framework for financial computation," in *2006 IEEE International Conference on Reconfigurable Computing and FPGA's (ReConFig 2006)*, pp. 1–9, IEEE, 2006.

[88] G. W. Morris and M. Aubury, "Design space exploration of the european option benchmark using hyperstreams," in *2007 International Conference on Field Programmable Logic and Applications*, pp. 5–10, IEEE, 2007.

[89] X. Tian, K. Benkrid, and X. Gu, "High performance Monte-Carlo based option pricing on FPGAs.," *Engineering Letters*, vol. 16, no. 3, 2008.

[90] X. Tian and K. Benkrid, "Design and implementation of a high performance financial Monte-Carlo simulation engine on an FPGA supercom-

puter," in *2008 International Conference on Field-Programmable Technology*, pp. 81–88, IEEE, 2008.

[91] X. Tian and K. Benkrid, "American option pricing on reconfigurable hardware using Least-Squares Monte Carlo method," in *2009 International Conference on Field-Programmable Technology*, pp. 263–270, 2009.

[92] N. A. Woods and T. VanCourt, "FPGA acceleration of quasi-Monte Carlo in finance," in *2008 International Conference on Field Programmable Logic and Applications*, pp. 335–340, IEEE, 2008.

[93] X. Tian and K. Benkrid, "Implementation of the Longstaff and Schwartz American option pricing model on FPGA," *Journal of Signal Processing Systems*, vol. 67, pp. 79–91, 2012.

[94] D. B. Thomas, "Acceleration of financial Monte-Carlo simulations using FPGAs," in *2010 IEEE Workshop on High Performance Computational Finance*, pp. 1–6, IEEE, 2010.

[95] H. Anson, D. B. Thomas, K. H. Tsoi, and W. Luk, "Dynamic scheduling Monte-Carlo framework for multi-accelerator heterogeneous clusters," in *2010 International Conference on Field-Programmable Technology*, pp. 233–240, IEEE, 2010.

[96] X. Tian and K. Benkrid, "Fixed-point arithmetic error estimation in Monte-Carlo simulations," in *2010 International Conference on Reconfigurable Computing and FPGAs*, pp. 202–207, IEEE, 2010.

[97] 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," in *2011 International Conference on Reconfigurable Computing and FPGAs*, pp. 468–474, IEEE, 2011.

[98] J. S. Hegner, J. Sindholt, and A. Nannarelli, "Design of power efficient FPGA based hardware accelerators for financial applications," in *NORCHIP 2012*, pp. 1–4, IEEE, 2012.

[99] D. Diamantopoulos, R. Polig, B. Ringlein, M. Purandare, B. Weiss, C. Hagleitner, M. Lantz, and F. Abel, "Acceleration-as-a-μservice: A cloud-native Monte-Carlo option pricing engine on CPUs, GPUs and

disaggregated FPGAs," in *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*, pp. 726–729, IEEE, 2021.

[100] Q. Liu, T. Todman, K. H. Tsoi, and W. Luk, "Convex models for accelerating applications on FPGA-based clusters," in *2010 International Conference on Field-Programmable Technology*, pp. 495–498, IEEE, 2010.

[101] X. Tian and K. Benkrid, "Libor market model simulation on an FPGA parallel machine," in *2010 VI Southern Programmable Logic Conference (SPL)*, pp. 9–14, IEEE, 2010.

[102] X. Tian and K. Benkrid, "High-performance quasi-Monte Carlo financial simulation: FPGA vs. GPP vs. GPU," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 3, no. 4, pp. 1–22, 2010.

[103] B. Betkaoui, D. B. Thomas, and W. Luk, "Comparing performance and energy efficiency of FPGAs and GPUs for high productivity computing," in *Field-Programmable Technology (FPT), 2010 International Conference on*, pp. 94–101, IEEE, 2010.

[104] G. C. T. Chow, A. H. T. Tse, Q. Jin, W. Luk, P. H. Leong, and D. B. Thomas, "A mixed precision Monte Carlo methodology for reconfigurable accelerator systems," in *Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays*, pp. 57–66, 2012.

[105] C. de Schryver, P. Torruella, and N. Wehn, "A multi-level Monte Carlo FPGA accelerator for option pricing in the Heston model," in *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 248–253, IEEE, 2013.

[106] Q. Jin, *Optimising Financial Computation for Reconfigurable Hardware*. PhD Thesis, Imperial College London, 2013. Available via CORE repository.

[107] 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," *Journal of Systems Architecture*, vol. 59, no. 3, pp. 135–143, 2013.

[108] M. De Jong, V.-M. Sima, K. Bertels, and D. Thomas, "FPGA-accelerated Monte-Carlo integration using stratified sampling and Brownian bridges," in *2014 International Conference on Field-Programmable Technology (FPT)*, pp. 68–75, IEEE, 2014.

[109] M. de Jong, "Hardware acceleration of Monte-Carlo integration in finance," 2014.

[110] S. Omland, M. Hefter, K. Ritter, C. Brugger, C. De Schryver, N. Wehn, and A. Kostiuk, *Exploiting Mixed-Precision Arithmetics in a Multilevel Monte Carlo Approach on FPGAs*, pp. 191–220. Cham: Springer International Publishing, 2015.

[111] J. Choi, R. L. Lian, S. Brown, and J. Anderson, "A unified software approach to specify pipeline and spatial parallelism in FPGA hardware," in *2016 IEEE 27th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pp. 75–82, 2016.

[112] A. Lomuscio, A. Nannarelli, and M. Re, "FPGA acceleration by dynamically-loaded hardware libraries," *Energy*, vol. 1, p. 3, 2016.

[113] F. B. 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.

[114] K. Setetemela and S. Winberg, "Systematic design of an ideal toolflow for accelerating big data applications on FPGA platforms," in *2018 IEEE 9th International Conference on Mechanical and Intelligent Manufacturing Technologies (ICMIMT)*, pp. 202–206, IEEE, 2018.

[115] C. Brugger, C. de Schryver, and N. Wehn, "Hyper: A runtime reconfigurable architecture for Monte Carlo option pricing in the Heston model," in *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 1–8, 2014.

[116] J. K. Toft and A. Nannarelli, "Energy efficient FPGA based hardware accelerators for financial applications," in *2014 NORCHIP*, pp. 1–6, IEEE, 2014.

[117] E. S. Schwartz, "The valuation of warrants: Implementing a new approach," *Journal of Financial Economics*, vol. 4, no. 1, pp. 79–93, 1977.

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

[119] Q. Jin, D. B. Thomas, and W. Luk, "Exploring reconfigurable architectures for explicit finite difference option pricing models," in *Field Pro-*

grammable Logic and Applications, 2009. FPL 2009. International Conference on, pp. 73–78, IEEE, 2009.

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

[121] P. Albicocco, D. Papini, and A. Nannarelli, "Direct measurement of power dissipated by Monte Carlo simulations on CPU and FPGA platforms," *IMM Technical Report 2012-18*, 2012.

[122] S. L. Heston, "A closed-form solution for options with stochastic volatility with applications to bond and currency options," *The review of financial studies*, vol. 6, no. 2, pp. 327–343, 1993.

[123] C. de Schryver, M. Jung, N. Wehn, H. Marxen, A. Kostiuk, and R. Korn, "Energy efficient acceleration and evaluation of financial computations towards real-time pricing," in *Knowledge-Based and Intelligent Information and Engineering Systems: 15th International Conference, KES 2011, Kaiserslautern, Germany, September 12-14, 2011, Proceedings, Part IV 15*, pp. 177–186, Springer, 2011.

[124] C. Stumm, "Investigate the hardware description language Chisel-A case study implementing the Heston model," 2013.

[125] Q. Wu, Y. Ha, A. Kumar, S. Luo, A. Li, and S. Mohamed, "A heterogeneous platform with GPU and FPGA for power efficient high performance computing," in *2014 international symposium on integrated circuits (ISIC)*, pp. 220–223, IEEE, 2014.

[126] M. Klaisoongnoen, N. Brown, and O. T. Brown, "Low-power option Greeks: Efficiency-driven market risk analysis using FPGAs," in *International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies*, pp. 95–101, 2022.

[127] A. D. Andricopoulos, M. Widdicks, P. W. Duck, and D. P. Newton, "Universal option valuation using quadrature methods," *Journal of Financial Economics*, vol. 67, no. 3, pp. 447–471, 2003.

[128] H. Anson, D. B. Thomas, and W. Luk, "Accelerating quadrature methods for option valuation," in *2009 17th IEEE Symposium on Field Programmable Custom Computing Machines*, pp. 29–36, IEEE, 2009.

[129] Q. Jin, D. B. Thomas, and W. Luk, "Automated application acceleration using software to hardware transformation," in *2009 International Conference on Field-Programmable Technology*, pp. 411–414, IEEE, 2009.

[130] H. Anson, D. Thomas, and W. Luk, "Design exploration of quadrature methods in option pricing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 5, pp. 818–826, 2011.

[131] A. H. Tse, G. C. Chow, Q. Jin, D. B. Thomas, and W. Luk, "Optimising performance of quadrature methods with reduced precision," in *Reconfigurable Computing: Architectures, Tools and Applications: 8th International Symposium, ARC 2012, Hong Kong, China, March 19-23, 2012. Proceedings 8*, pp. 251–263, Springer, 2012.

[132] A. Kaganov, P. Chow, and A. Lakhany, "FPGA acceleration of Monte-Carlo based credit derivative pricing," in *2008 International Conference on Field Programmable Logic and Applications*, pp. 329–334, IEEE, 2008.

[133] A. Kaganov, *Hardware acceleration of Monte-Carlo structural financial instrument pricing using a Gaussian Copula model*. University of Toronto, 2008.

[134] A. Kaganov, A. Lakhany, and P. Chow, "FPGA acceleration of multifactor CDO pricing," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 4, no. 2, pp. 1–17, 2011.

[135] D. B. Thomas and W. Luk, "Multivariate Gaussian random number generation targeting reconfigurable hardware," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 1, no. 2, pp. 1–29, 2008.

[136] C. Saiprasert, C.-S. Bouganis, and G. A. Constantinides, "Design of a financial application driven multivariate Gaussian random number generator for an FPGA," in *Reconfigurable Computing: Architectures, Tools and Applications: 6th International Symposium, ARC 2010, Bangkok, Thailand, March 17-19, 2010. Proceedings 6*, pp. 182–193, Springer, 2010.

[137] 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)*, pp. 278–285, IEEE, 2017.

[138] L. Ma, F. B. 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)*, pp. 157–162, IEEE, 2016.

[139] 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," *2015 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, pp. 1–6, 2015.

[140] J. A. Varela, C. Brugger, S. Tang, N. Wehn, and R. Korn, *Pricing High-Dimensional American Options on Hybrid CPU/FPGA Systems*, pp. 143–166. Cham: Springer International Publishing, 2015.

[141] A. M. Nestorov, E. Reggiani, H. Palikareva, P. Burovskiy, T. Becker, and M. D. Santambrogio, "A scalable dataflow implementation of Curran's approximation algorithm," in *2017 IEEE international parallel and distributed processing symposium workshops (IPDPSW)*, pp. 150–157, IEEE, 2017.

[142] F. Peverelli, M. Rabozzi, E. Del Sozzo, and M. D. Santambrogio, "Oxigen: a tool for automatic acceleration of C functions into dataflow FPGA-based kernels," in *2018 IEEE international parallel and distributed processing symposium workshops (IPDPSW)*, pp. 91–98, IEEE, 2018.

[143] F. Peverelli, M. Rabozzi, S. Cardamone, E. Del Sozzo, A. J. Thom, M. D. Santambrogio, and L. Di Tucci, "Automated acceleration of dataflow-oriented C applications on FPGA-based systems," in *2019 IEEE 27th annual international symposium on field-programmable custom computing machines (FCCM)*, pp. 313–313, IEEE, 2019.

[144] S. Ibraev, "Acceleration of the strike calculation for foreign exchange options using FPGA," *Available at SSRN 3540371*, 2020.

[145] S. Ibraev and M. Deng, "FPGA application to option price inversion," *Available at SSRN 4124011*, 2022.

[146] G. W. Morris, D. B. Thomas, and W. Luk, "FPGA accelerated low-latency market data feed processing," in *2009 17th IEEE Symposium on High Performance Interconnects*, pp. 83–89, IEEE, 2009.

[147] J. W. Lockwood, A. Gupte, N. Mehta, M. Blott, T. English, and K. Vissers, "A low-latency library in FPGA hardware for high-frequency trading (HFT)," in *2012 IEEE 20th annual symposium on high-performance interconnects*, pp. 9–16, IEEE, 2012.

[148] C. Leber, B. Geib, and H. Litz, "High frequency trading acceleration using FPGAs," in *2011 21st International Conference on Field Programmable Logic and Applications*, pp. 317–322, IEEE, 2011.

[149] E. Del Sozzo, D. Conficconi, M. D. Santambrogio, and K. Sano, "Senju: A framework for the design of highly parallel FPGA-based iterative stencil loop accelerators," in *Proceedings of the 2023 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pp. 233–233, 2023.

[150] H. Lázaro García, "Design and implementation of a multi-level Monte Carlo accelerator for option pricing on the Zynq-7000 EPP," B.S. thesis, 2013.

[151] S. Palmer, "Accelerating implicit finite difference schemes using a hardware optimized tridiagonal solver for FPGAs," *arXiv preprint arXiv:1402.5094*, 2014.

[152] C. Starke, V. Grossmann, L. Wienbrandt, and M. Schimmler, "An FPGA implementation of an investment strategy processor," *Procedia Computer Science*, vol. 9, pp. 1880–1889, 2012.

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

[154] M. Kurek, T. Becker, T. C. Chau, and W. Luk, "Automating optimization of reconfigurable designs," in *2014 IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines*, pp. 210–213, 2014.

[155] T. S. Czajkowski, U. Aydonat, D. Denisenko, J. Freeman, M. Kinsner, D. Neto, J. Wong, P. Yiannacouras, and D. P. Singh, "From OpenCL to high-performance hardware on FPGAs," in *Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on*, pp. 531–534, IEEE, 2012.

[156] T. S. Czajkowski, D. Neto, M. Kinsner, U. Aydonat, J. Wong, D. Denisenko, P. Yiannacouras, J. Freeman, D. P. Singh, and S. D. Brown, "OpenCL for FPGAs: Prototyping a compiler," in *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, p. 1, The World Congress in Computer Science, 2012.

[157] K. Krommydas, A. E. Helal, A. Verma, and W.-C. Feng, "Bridging the performance-programmability gap for FPGAs via OpenCL: A case study with opendwarfs," tech. rep., Department of Computer Science, Virginia Polytechnic Institute & State , 2016.

[158] M. de Castro, D. L. Vilariño, Y. Torres, and D. R. Llanos, "The role of Field-Programmable Gate Arrays in the acceleration of modern high-performance computing workloads," *Computer*, vol. 57, no. 7, pp. 66–76, 2024.

[159] D. Sierag and B. Hanzon, "Pricing derivatives on multiple assets: recombining multinomial trees based on Pascal's Simplex," *Annals of Operations Research*, pp. 1–27, 2017.

[160] G. H. Golub and C. F. Van Loan, *Matrix computations (3rd ed.)*. USA: Johns Hopkins University Press, 1996.

[161] J. Lambert, S. Lee, J. S. Vetter, and A. Malony, "In-depth optimization with the OpenACC-to-FPGA framework on an Arria 10 FPGA," in *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 460–470, IEEE, 2020.

[162] G. P. M. Virgilio, "When spread bites fast–volatility and wide bid-ask spread in a mixed high-frequency and low-frequency environment," *Research in International Business and Finance*, vol. 51, p. 101066, 2020.

[163] C. de Schryver, P. Torruella, and N. Wehn, "A multi-level Monte Carlo FPGA accelerator for option pricing in the Heston model," in *2013 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 248–253, March 2013.

[164] P. Sundararajan, "High-performance computing using FPGAs," *Xilinx White Paper: FPGAs*, pp. 1–15, 2010.

[165] K. J. Leslie and M. P. Michaels, "The real power of real options," *The McKinsey Quarterly*, no. 3, pp. 4–23, 1997.

[166] B. Zhang and C. W. Oosterlee, "Acceleration of option pricing technique on graphics processing units," *Concurrency and Computation: Practice and Experience*, vol. 26, no. 9, pp. 1626–1639, 2014.

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

[168] R. Sridharan, G. Cooke, K. Hill, H. Lam, and A. George, "FPGA-based reconfigurable computing for pricing multi-asset barrier options," in *2012 Symposium on Application Accelerators in High Performance Computing*, pp. 34–43, IEEE, 2012.

[169] S. Fadugba, C. Nwozo, J. Okunlola, O. Adeyemo, and A. Ajayi, "On the accuracy of binomial model and Monte Carlo method for pricing European options," *International Journal of Mathematics and Statistics Studies, European Centre for Research Training and Development*, vol. 1, pp. 38–54, 2013.

[170] Q. Jin, W. Luk, and D. B. Thomas, "On comparing financial option price solvers on FPGA," in *Field-Programmable Custom Computing Machines (FCCM), 2011 IEEE 19th Annual International Symposium on*, pp. 89–92, IEEE, 2011.

[171] A. H. T. Tse, D. B. Thomas, and W. Luk, "Accelerating Quadrature Methods for option valuation," in *2009 17th IEEE Symposium on Field Programmable Custom Computing Machines*, pp. 29–36, 2009.

[172] H. Anson, D. B. Thomas, and W. Luk, "Option pricing with multi-dimensional quadrature architectures," in *2009 International Conference on Field-Programmable Technology*, pp. 427–430, IEEE, 2009.

[173] N. Zhang, E. Lim, K. L. Man, and C.-U. Lei, "CPU-GPU hybrid parallel binomial american option pricing," *Lecture Notes in Engineering and Computer Science*, 2012.

[174] B. Betkaoui, D. B. Thomas, and W. Luk, "Comparing performance and energy efficiency of FPGAs and GPUs for high productivity computing," in *2010 International Conference on Field-Programmable Technology*, pp. 94–101, 2010.

[175] R. A. Brealey, S. C. Myers, F. Allen, and P. Mohanty, *Principles of corporate finance*. Tata McGraw-Hill Education, 2012.

[176] H. U. Gerber, "Mathematical fun with the compound binomial process," *ASTIN Bulletin: The Journal of the IAA*, vol. 18, no. 2, pp. 161–168, 1988.

[177] D. Cox, J. Little, and D. O'Shea, "Ideals, varieties, algorithms: An introduction to computational algebraic geometry and commutative algebra, Springer-Verlag, New York," 1992.

[178] D. A. Cox, J. Little, and D. O'Shea, *Using Algebraic Geometry*, vol. 185. Springer Science & Business Media, 2005. See Definition 3.3 for detailed explanation.

[179] X. Tian, R. Han, L. Wang, G. Lu, and J. Zhan, "Latency critical big data computing in finance," *The Journal of Finance and Data Science*, vol. 1, no. 1, pp. 33–41, 2015.

[180] A. O. Mahony, G. Zeidan, B. Hanzon, and E. Popovici, "A parallel and pipelined implementation of a Pascal-Simplex based two asset option pricer on FPGA using OpenCL," in *2020 IEEE Nordic Circuits and Systems Conference (NorCAS)*, pp. 1–6, 2020.

[181] C. Ikamari, P. Ngare, and P. Weke, "Multi-asset option pricing using an information-based model," *Scientific African*, vol. 10, p. e00564, 2020. (Accessed on 27/04/2021).

[182] X. Yu, "Development of financial option pricing system based on FPGA and machine learning," *Microprocessors and Microsystems*, vol. 81, p. 103708, 2021.

[183] A. A. Purkayastha, S. Rogers, S. A. Shiddibhavi, and H. Tabkhi, "LLVM-based automation of memory decoupling for OpenCL applications on FPGAs," *Microprocessors and Microsystems*, vol. 72, p. 102909, 2020. (Accessed on 27/04/2021).

[184] Intel Corporation, "FPGA Acceleration with Intel DevCloud," 2021. (Accessed on 27/04/2021).

[185] H. Johnson, "Options on the maximum or the minimum of several assets," *Journal of Financial and Quantitative analysis*, pp. 277–283, 1987.

[186] W. Hurlimann, "Valuation of fixed and variable rate mortgages: binomial tree versus analytical approximations," *Decisions in Economics and Finance*, vol. 35, no. 2, pp. 171–202, 2012.

[187] Intel Corporation, "Intel opencl application developer guide." `https://www.intel.com/content/www/us/en/developer/articles/technical/opencl-application-developer-guide.html`, 2024. Accessed: 2025-03-08.

[188] R. Martin, "Wall Street's quest to process data at the speed of light," *Information Week*, vol. 4, no. 21, p. 07, 2007.

[189] M. Broadie and J. Detemple, "The valuation of American options on multiple assets," *Mathematical Finance*, vol. 7, no. 3, pp. 241–286, 1997.

[190] M. Broadie and P. Glasserman, "Pricing American-style securities using simulation," *Journal of Economic Dynamics and Control*, vol. 21, no. 8-9, pp. 1323–1352, 1997.

[191] L. Andersen and M. Broadie, "Primal-dual simulation algorithm for pricing multidimensional American options," *Management Science*, vol. 50, no. 9, pp. 1222–1234, 2004.

[192] S. Tezuka, "Financial applications of Monte Carlo and quasi-Monte Carlo methods," *Lecture Notes in Statistics*, pp. 303–332, 1998.

[193] J. W. Wan, K. Lai, A. W. Kolkiewicz, and K. Tan, "A parallel quasi-Monte Carlo approach to pricing American options on multiple assets," *International Journal of High Performance Computing and Networking*, vol. 4, no. 5/6, pp. 321–330, 2006.

[194] M. Broadie and J. Detemple, "American option valuation: new bounds, approximations, and a comparison of existing methods," *The Review of Financial Studies*, vol. 9, no. 4, pp. 1211–1250, 1996.

[195] M. Gaudenzi and F. Pressacco, "An efficient binomial method for pricing American options," *Decisions in Economics and Finance*, vol. 26, pp. 1–17, 2003.

[196] K. Obata, H. M. Waidyasooriya, and M. Hariyama, "Implementation of an FPGA-oriented complex number computation library using Intel