A Review of GPU Acceleration Techniques in Option Pricing Models

Daniel Michaeli

School of Science

Bachelor's thesis
Espoo 16.2.2025

Thesis supervisor and advisor:

M.Sc. Henrik Lievonen

AALTO UNIVERSITY
SCHOOL OF SCIENCE

ABSTRACT OF THE
BACHELOR'S THESIS

Author: Daniel Michaeli

Title: A Review of GPU Acceleration Techniques in Option Pricing Models

Date: 16.2.2025 Language: English Number of pages: 4+29

Degree programme: Computer Science

Supervisor and advisor: M.Sc. Henrik Lievonen

English bla bla bla

Keywords: moi, mojn, moin, morjens, moro

AALTO-UNIVERSITETET
HÖGSKOLAN FÖR TEKNIKVETENSKAPER

SAMMANDRAG AV
KANDIDATARBETET

Författare: Daniel Michaeli

Titel: Genomströmning och latens i datorsystem: en anlays av avvägningseffekter och optimeringsstrategier

Datum: 16.2.2025 Språk: Engelska Sidantal: 4+29

Utbildningsprogram: Datateknik

Ansvarslärare: M.Sc. Henrik Lievonen

Handledare: M.Sc. Henrik Lievonen

svenska bla bla bla

Nyckelord: Nyckelord på svenska,
Moi, moin, moidå, mojjdå

Contents

1 Introduction

Options are a type of financial contract between two parties that grants the right – but not the obligation – to buy or sell a specific amount of an asset at a predetermined price by a specific future date [1]. Options are thus considered a financial derivative, as their value is derived from the price of an underlying asset. Advances in both financial mathematics [2] and affordable computing power [3] have contributed to a rapid growth in the derivatives market. These developments have improved pricing models and risk management tools, making derivatives more accessible and practical. Consequently, their use has increased significantly over the last half-century, not only among investors and traders, but also among non-financial corporations [4]. Options and other derivatives facilitate both leveraged speculation and sophisticated risk management, allowing market participants to achieve more control over exposure to various financial risks.

There are many option pricing models, each based on different assumptions and with their own advantages and disadvantages. Regardless of the model chosen, computation speed remains a critical factor. In high-frequency trading, where transient market inefficiencies **(do I have to explain this?)** are quickly exploited over millions of trades, milliseconds can determine whether a strategy is profitable or not. Additionally, faster computation produces more available data within a given time frame, which aids in risk management practices that often include simulation and analysis of complex portfolios over many different scenarios. This has led to a rising interest in adapting option pricing algorithms to leverage GPU parallel computing capabilities. **(do I need to define GPU here? CS thesis after all)**

This thesis forms a literature review of common option pricing algorithms' GPU acceleration potential and limitations. The aim is to assess how dependency structures in different algorithms affect their GPU parallelization potential, what approaches see the greatest performance improvements of a GPU implementation, analyze the scalability of these solutions, and identify related bottlenecks. I have chosen to analyze the Cox-Ross-Rubinstein (CRR) binomial option pricing model, and Monte Carlo (MC) simulation-based methods. The Black-Scholes analytical formula has been excluded as its GPU implementation is purely arithmetic evaluation with no algorithmic content. Methods requiring PDE numerical solutions would offer rich parallelization challenges but demand mathematical background beyond this thesis's scope.

The Section 2 introduces formal definitions and fundamental option pricing theory. Section 3 presents a high-level overview of the GPU, how it differs architecturally from the CPU, and how it can be used for general-purpose computing. Section 4, **??**, and Section 5 review GPU implementations of each respective approach, considering the aforementioned factors. The results are summarized in Section 6 and conclusions are drawn in Section 7.

For scope restriction purposes, the following choices have been made:

- Only European and American options are considered, with the exception of the Monte Carlo methods, which are particularly suitable for pricing exotic options with complex payoff structures. The term *vanilla options* refers to both

European and American options, whereas *exotic options* denotes more complex contracts. Unless explicitly stated otherwise, all discussions and conclusions apply to options in general. When specific terms are used, the statements pertain strictly to those contract styles. Furthermore, it is assumed that the underlying asset does not pay a dividend.

- The purpose of this paper is to draw general conclusions about the GPU acceleration potential for different option pricing approaches. The referenced research makes use of varying computing architectures and performance metrics. Other differences, like the number of active cores in the baseline CPU implementation, or the extent to which further optimizations have been implemented, also play a role. This makes direct quantitative comparisons difficult. To account for this limitation, the main comparisons of interest are the relative speedups of the GPU-accelerated implementations compared to their respective baseline solutions, and to a smaller extent relative performance comparisons between different GPU-accelerated implementations.

- The models and implementations are studied purely from the perspective of computational efficiency. That is, no interest is taken in the analysis of accuracy, numerical stability, or any other unrelated aspect. The focus remains exclusively on execution time improvements and parallelization potential.

2 Option Pricing Fundamentals

In the following section, I present a sufficient theoretical foundation for understanding and pricing options. Firstly, general definitions are established along with vanilla option payoff structures **(if these are two separate subsections, can I use word along here?)**. This is followed by a section **(do I hyperlink it?)** on the key determinants that influence option prices. Next, the no-arbitrage principle and risk-neutral valuation framework are presented, explaining the theory that underpins all pricing models. The section concludes with a discussion on the inherent computational challenges of option pricing, further motivating the exploration of GPU acceleration techniques. **have not written yet this inherent computational challenges thing, perhaps not necessary?**

2.1 Definitions

The following terminology is from Chapter 1.5 in Hull's classic book "Options, Futures, and Other Derivatives" [1]:

Call Option: Contract that grants the holder the right (but not the obligation) to buy the underlying asset by a certain date for a certain price.

Put Option: Contract that grants the holder the right (but not the obligation) to sell the underlying asset by a certain date for a certain price.

Strike Price: The predetermined price at which the underlying asset can be bought (for a call option) or sold (for a put option) upon exercise.

Maturity (date): The predetermined date on which the option expires, determining the latest point at which it can be exercised.

Note the ambiguous use of "by" in the definitions above, as the specific rules for when an option can be exercised depends on the style of option. A European option can only be exercised on the maturity date, whereas an American option can be exercised at any time up to maturity. For clarity, the underlying asset will hereinafter be simply referred to as the stock, and the current stock price referred to as the *spot price*. There are always two parties to every option contract: the buyer, who takes the *long* position, pays an upfront premium to the underwriter (seller) for the right to engage in a future trade. Conversely, the underwriter, who takes the *short* position, incurs an obligation to engage in this trade, thus assuming risk. **(I don't know whether to use emph, bold, or parentheses here for terminology)**

2.2 Vanilla Option Payoff Structures

An option is a zero-sum game between the buyer and the underwriter, hence we have a pair of mirrored positions for every contract. These can be visualized by graphing

the profit at the time of exercising as a function of the spot price. Alternatively, one can consider the slightly modified *payoff* of the position, which excludes the premium paid or received to focus only on the fundamental mechanics of the option itself, regardless of what someone paid for it: profit = payoff − premium. Profit diagrams for each vanilla option position are presented below, along with practical examples for intuition. The options are based on one unit of stock. Transaction costs and taxes are ignored, and we assume liquid markets to the extent that instantaneous buying and selling of assets is possible.

For a call option, the investor is willing to pay a premium upfront in order to fix a purchase price for a later time. In other words, they expect the price of the stock to increase enough to offset this premium. Figure 1 depicts this position. As long as the spot price is less than the strike price, $S_T < K$, the buyer is at a net loss equal to the premium p. The option is said to be *out of the money (OTM)* and has no intrinsic value. When $S_T = K$ the option is considered to be *at the money (ATM)*, after which, when $S_T > K$ it is *in the money (ITM)* and has intrinsic value equal to the payoff [5]. Still, the profit remains negative as long as payoff $< p$. Only after offsetting the premium does the option truly become profitable for the buyer. Note that the "moneyness" terminology is always defined from the perspective of the long position. Assuming liquid markets, the buyer could then exercise the option to buy the stock for the strike price, only to immediately sell for the spot price to profit off the difference. In general, ITM options are exercised, as the intrinsic value helps to offset at least part of the loss from paying the premium.[1] To demonstrate options' role in risk management, as opposed to pure speculation, consider a manufacturer heavily reliant on crude oil. If this manufacturer anticipates a significant increase in oil prices a year from now, they might choose to enter a long position in a call option on crude oil. Here, the focus shifts from profit towards hedging against price risk, functioning similarly to insurance. The manufacturer willingly pays a premium to establish a price ceiling on future oil prices, and agrees on a strike price in accordance with their risk management strategy.

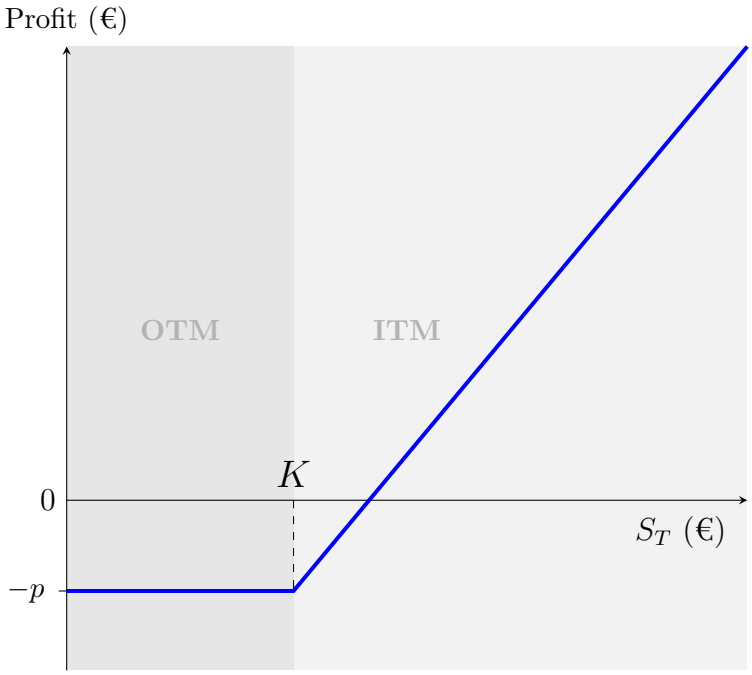


Figure 1: Profit diagram for the long position in a call option. K = strike price, S_T = spot price, p = premium. The buyer pays a premium to fix a maximum purchase price for the stock at a later point in time. At maturity, as long as $S_T < K$ there is no incentive to exercise the option, and the profit thus equals $-p$. The option is worthless, and is considered out of the money (OTM). When $S_T > K$, the option is in the money (ITM) and is assumed to be exercised as the buyer can now purchase the stock for K, immediately sell it for S_T, and offset some or all of the premium paid. However, the investment is only profitable once $S_T - K > p$.

The short call position exhibits an inverse profit profile compared to the long position: as long as the option remains OTM, the underwriter retains the full premium as profit, since the investor will not exercise the option. Once the option becomes ITM, the buyer will exercise it, forcing the underwriter to sell the stock at the strike price K despite its higher market value S_T. The premium initially offsets this adverse price difference until the breakeven point where $S_T - K = p$, after which the position becomes a loss for the underwriter. Figure 2 illustrates this position. Continuing with the previous example, this position could be taken by a crude oil producer with conviction that prices will remain below or near the strike price at maturity. The risk borne by the underwriter commands the premium.

Ignoring the premium, the payoff from the long position in a call option is defined as

$$\max(S_T - K, 0) \tag{1}$$

as the investor will only exercise when ITM to either profit or offset losses. Consequently, the payoff from the short position is defined as $-\max(S_T - K, 0) =$

$\min(K - S_T, 0)$ due to the zero-sum nature of the contract.[1][1]

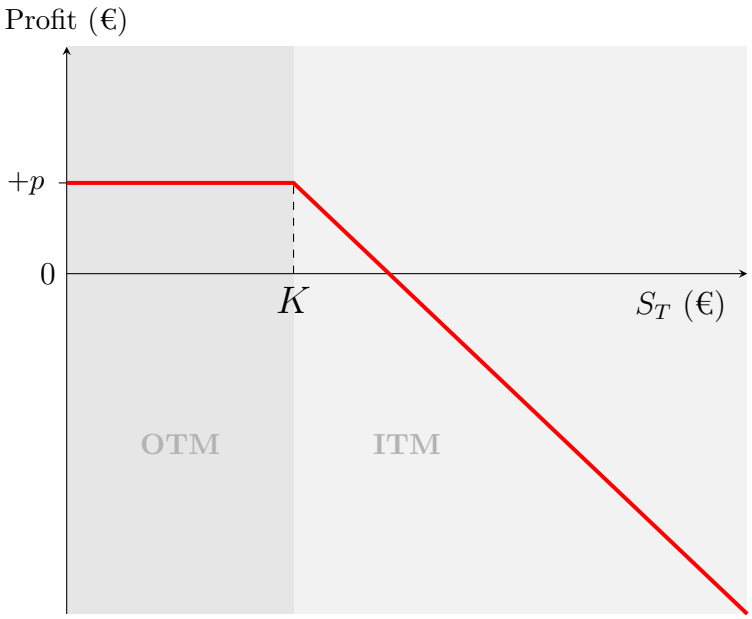
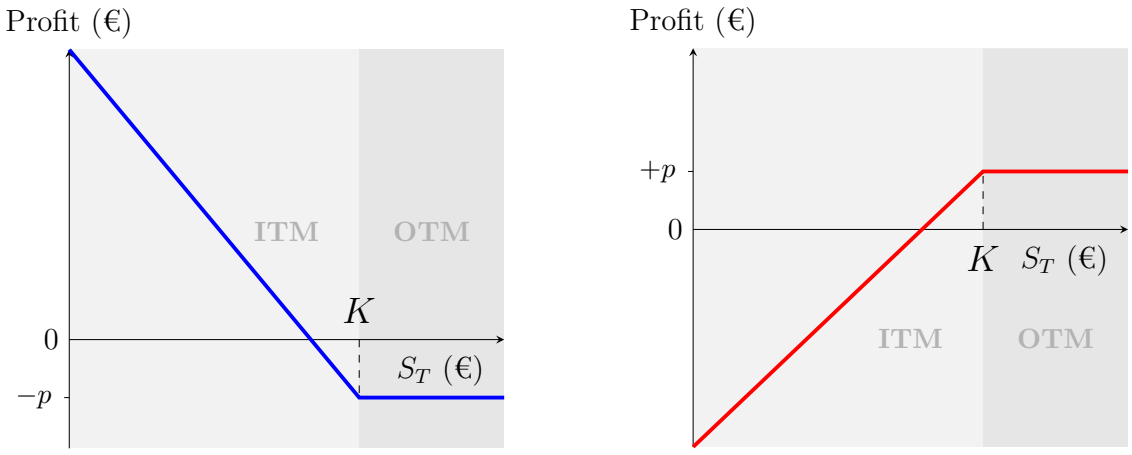


Figure 2: Profit diagram for the short position in a call option. K = strike price, S_T = spot price, p = premium. The underwriter receives a premium but is obligated to sell the stock for K at a later time point, should the option be exercised. When $S_T < K$, the option is worthless and considered out of the money (OTM), allowing the underwriter to keep the full premium as profit. When $S_T > K$ and the option is exercised in the money (ITM), the underwriter is forced to sell the stock at the strike price K despite its higher market value S_T. The profit decreases linearly as S_T increases, becoming negative once $S_T - K > p$.

For a put option, the investor is willing to pay a premium up front to secure a minimum selling price for the stock at a later time point. They expect the price of the stock to decrease sufficiently to offset this premium. Conversely, the underwriter receives the premium but assumes the obligation to purchase the stock at the strike price K, should the option be exercised. Figure 3 depicts the profit diagrams of the long and short positions of a put option. A put option is OTM when $S_T > K$ as the investor will not exercise it to sell the stock for a lower price than market value. By the same token, the option is ITM when $S_T < K$, as the investor can then buy the stock asset at market value and sell it for the strike price [5]. Again, being ITM is not sufficient to be profitable - the price difference must be large enough to offset the paid premium. An example use case would be a crude oil supplier forecasting low prices in a year's time and entering the long position to secure a minimum revenue.

[1]For American-style options with the right to early-exercise, the payoff is calculated using S_τ instead of S_T, where $\tau \leq T$ is the chosen point of exercise.

Again, a premium is paid to hedge against a risk of price swings. The short side would be taken by a market participant who anticipates crude oil prices not to fall enough to offset the received premium.



(a) Long put option payoff

(b) Short put option payoff

Figure 3: Profit diagrams for long (a) and short (b) positions in a put option. K = strike price, S_T = spot price, p = premium. For the long position, the buyer pays a premium to secure a minimum selling price for the stock at a future time point. When $S_T < K$ the option is in the money (ITM) and would be exercised, as the underwriter is forced to buy the stock for K despite its lower market value S_T. When $S_T > K$ the option is out of the money (OTM) with no incentive to exercise, resulting in a loss $-p$. For the short position, the underwriter receives the premium but assumes the obligation to purchase the stock at K, should the option be exercised. The underwriter profits fully when the option remains OTM, but faces decreasing profits when ITM as the spot price decreases. The position becomes unprofitable once $K - S_T > p$.

The payoff from the long position in a put option is defined as

$$\max(K - S_T, 0) \tag{2}$$

as being ITM now means $K > S_T$. Conversely, the payoff in the corresponding short position is $-\max(K - S_T, 0) = \min(S_T - K, 0)$ [1].

The long position in any option contract is never obligated to exercise, and the loss is therefore limited to the premium paid. On the other hand, the underwriter is always obligated to engage in a disadvantageous trade with the buyer. Additionally, since underlying asset prices have no theoretical upper bound but are limited to non-negative values, the short call position faces potentially unlimited losses. Table 1 aggregates information about the extreme-case profit and loss incurred from the different option types and positions.

The objective of pricing models is to find the so-called fair value of the option p. Although market prices form through the dynamic interaction of buyers and sellers, they are still largely anchored by theoretical pricing frameworks. These models provide

Table 1: Extreme-case upside and downside potential for different option positions. p = premium paid or received, K = strike price. The long position in a call option has unlimited upside potential due to theoretically unbounded underlying asset prices, with losses limited to the premium paid. The corresponding short position mirrors this behavior with unlimited downside risk. Put options have bounded payoffs as asset prices cannot fall below zero. The maximum gain for a long put position is $K - p$ when the spot price reaches zero, and the maximum loss for the corresponding short position is $-(K - p)$.

	Call Option	**Put Option**
Long	**Maximum Gain:** $+\infty$ **Maximum Loss:** $-p$	**Maximum Gain:** $K - p$ **Maximum Loss:** $-p$
Short	**Maximum Gain:** $+p$ **Maximum Loss:** $-\infty$	**Maximum Gain:** $+p$ **Maximum Loss:** $-(K-p)$

a rational foundation for valuing derivatives based on mathematical and economic theory. While real-world prices may show short-term deviations, markets tend to converge toward theoretically justified values over time, as persistent mispricing creates exploitable opportunities. The theoretical fair value serves as a crucial reference point that informs trading decisions and risk management strategies.

2.3 Price Determinants

Hull et al. [5] define six factors influencing the price of the option. While these factors typically serve as variables in pricing models, their effects can be intuitively understood even without formal mathematical frameworks. When analyzing a factor's impact, it is usually done in a "ceteris paribus" manner. In practice, however, this isolation rarely occurs, as markets quickly incorporate all available information into prices [6], causing interrelated changes among the factors themselves. For clarity, the following analysis considers the long position, though the same reasoning applies symmetrically to the short position, as both sides value the same option.

The stock price relative to the strike price is perhaps the most obvious factor. The effect on the payoff is thoroughly explained in Section 2.2, and the price naturally increases with the payoff. Thus, the price of a call option increases with the stock price, while the inverse relationship is observed for a put option.

Volatility, a statistical measure of the variation in asset prices over a specific time period, has a positive effect on both call and put options. This positive relationship can be motivated by the limited downside – unlimited upside nature of options (**??**). Higher stock price volatility increases the probability of significant movements, making it more likely the option will move ITM. The benefit is especially pronounced for OTM positions, where substantial price movement is needed to avoid expiring worthless, but holds for ITM positions as well. In the latter case, the unlimited potential for becoming even deeper ITM outweighs the limited downside risk of moving OTM.

Time to maturity has a more complex relationship with the price, and depends on

both the option style and the context. For American options with possibility of early exercise, it makes sense that an expiration date further in the future leads to more opportunity for favorable movements in the stock price, thereby commanding a higher price. This time value argument is similar to that made for volatility, where the asymmetric payoff structure means additional potential for gain typically outweighs the risk of further losses. While this time value argument also holds for European options, the lack of early-exercise ability introduces complications. Hull et al. [5] identify specific scenarios where American options derive significant additional value from the early exercise privilege. One example is when a future dividend payment to shareholders is expected, which theoretically decreases the share price and thereby the value of the corresponding call option. Similarly, a put option might be more valuable with a shorter time to maturity in the case of a very deep ITM position, such as when the underlying company has filed for bankruptcy. The option will almost certainly expire ITM, and thus by the *time value of money* **(should I explain this in a footnote in greater detail?)** principle this cash flow is worth more the earlier it is received [7].

The risk-free rate represents the return on investment one can expect from a so called risk-free investment, typically government-backed treasury securities. It serves as a benchmark against which investors compare expected returns of riskier investments. An increase in the risk-free rate typically leads to investors demanding higher returns across all risk levels. In option pricing, the risk-free rate primarily affects the time value component through the discounting of future cash flows. For a call option, a higher risk-free rate reduces the present value of the strike price that will be paid in the future[2]. This economic advantage for the buyer is reflected in the market through a higher option price. Conversely, for put options, a higher risk-free rate decreases the present value of the strike price to be received in the future, reducing the economic benefit to the holder and thereby decreasing the option's price. In practice, however, interest rates impact stock prices as well, making this relationship more complex when applied to actual markets.

The final determinant considers the dividend policy of the underlying company. Theoretically, a company's share price is reduced by the dividend amount on the ex-dividend date, as this cash is directly distributed from the company's assets to shareholders [8]. For call options, expected dividends decrease value because they reduce the expected future price of the stock without providing any benefit to the option buyer. Conversely, put options increase in value with higher expected dividends, as they decrease the value of the stock. This dividend effect is particularly significant for longer-dated options, where multiple dividend payments may occur before expiration, and for options near the money, where the dividend-induced price movement can meaningfully change the moneyness of the option.

[2]The present value of a future cash flow is calculated by discounting it using an appropriate rate. This rate reflects both the time value of money and opportunity cost. The risk-free rate represents the minimum discount rate, as it corresponds to the return available from risk-free investments. For risky cash flows, an additional risk premium is added to this base rate.

2.4 No-Arbitrage Pricing and Risk-Neutral Valuation

Previously, all main "ingredients" for the pricing formulas were introduced. This section outlines the foundational theoretical framework that underpins all option pricing models, bringing us one step closer to formulating the models to be GPU-accelerated. In line with the thesis scope, the focus will be on intuition rather than mathematical rigor. Avid readers are encouraged to explore the references in greater detail [1, 9, 10]. The assumptions presented in Section 2.2 continue to hold, as will the referral of the underlying asset as the stock. The setup considers European-style options, but in this single-period setting, the results are equally valid for American options, as early exercise provides no advantage.

A natural starting point is to consider how other financial assets are valued. The theoretical value of an asset is commonly understood as the discounted sum of all future cash flows it produces. Under uncertainty – as is the case of all risky assets – a standard approach would be to discount the *expected value* of these cash flows, which represents a theoretical long-term average of the returns **(basic math and financial theory, do I need src for this?)**. Calculating this expected value, however, requires defining a set of possible outcomes and assigning probabilities to each. This is a fundamental limitation of pricing models, as the estimates are often questionable and, at least in part, subjective. Nonetheless, starting out with a simplistic model will lead to interesting and useful revelations that can be expanded on.

The *no-arbitrage principle* is the essential assumption for what is to come. Arbitrage describes the act of simultaneous buying and selling of the same asset in different markets to make risk-free profit off price differences. Imagine the price of tomatoes being higher in one of two neighboring towns. Perhaps the tomato farmers were really successful in town A this year, and the increased supply dropped the prices somewhat. This information has not yet been priced into the tomatoes in town B. The astute tradesman could then buy tomatoes in town A and immediately sell them in town B to pocket the difference, without taking any position on future tomato price movements. This, again, assumes ideal conditions, like enough liquidity to always be able to both buy and sell, as well as ignoring friction costs like transportation. The idea behind our assumption is that this strategy ultimately is self-eliminating: as the tradesman continues to introduce more tomatoes to the total supply in town B they will drive the price down until an equilibrium has been reached. Other people may take notice and decide to adopt the same strategy, eliminating the opportunity even faster. The information about the larger supply in town A has now been priced into the tomatoes in town B. This example is not too far-fetched from reality. Modern digital markets do not perfectly adhere to the assumptions laid out above, and short-term arbitrage opportunities can exist. Nevertheless, as sophisticated market participants exploit these inefficiencies they quickly disappear due to subsequent price corrections. Thus, the no-arbitrage principle is a reasonable assumption, at least over longer time periods. If our pricing methodology is based on summing future cash flows, it logically follows that in absence of arbitrage opportunities two assets producing the same cash flows must be priced equally [1, 11].

If it is possible to find an existing asset with identical future cash flows to those of the option, and whose price is already known, then by the no-arbitrage principle, the price of that asset should theoretically equal the price of the option. Hull et al. [5] consider a one-step model as the simplest framework to apply this idea. In this discrete-time model, we consider the stock price evolution over a single period. Let S_0 denote the current stock price, and let f be the current price of an option on one share, with time T to maturity. The stock price is assumed to move to one of two possible future values: S_0u or S_0d ($u > 1$, $d < 1$). These correspond to the option payoffs f_u and f_d, respectively. Figure 4 visualizes the model setup. This restriction makes it possible to construct a so-called *replicating portfolio* that perfectly matches the option's payoff in both states. This portfolio consists of Δ shares of the stock, as well as B amount of a risk-free asset (e.g., a savings account) growing at the risk-free rate r. After time T, the portfolio will take on one of two values, depending on whether the stock price increased or decreased. By equating the value of the portfolio in each case with the corresponding option payoff, one obtains a system of two equations and two unknowns, Δ and B.

$$\begin{cases} \Delta(S_0u) + B(1+r) = f_u \\ \Delta(S_0d) + B(1+r) = f_d \end{cases}$$

Solving for these variables yields

$$\begin{cases} \Delta = \frac{f_u - f_d}{S_0(u-d)} \\ B = \frac{f_d u - f_u d}{(u-d)(1+r)} \end{cases}$$

It is evidently possible to construct a portfolio with identical payoff structure as the option itself, according to

$$\frac{f_u - f_d}{S_0(u-d)} S_0 + \frac{f_d u - f_u d}{(u-d)(1+r)} \tag{3}$$

Under the no-arbitrage principle, the price of the option must thus equal the price of constructing the replicating portfolio, i.e., buying Δ shares and investing B in a risk-free asset. Otherwise, one could buy the cheaper alternative while selling the more expensive to obtain a risk-free profit. A negative Δ implies shorting the underlying asset, and a negative B implies borrowing money at the risk-free rate instead of lending (saving) it.

The derivation assumed neither a call nor a put option, and works therefore as a pricing model for any European option. A more interesting observation, however, is the fact that this pricing approach contains no information about the expected return of the stock, or the probability with which it will increase or decrease in value. The authors motivate this intuitively by the fact that such expectations are already reflected in the stock price itself, and need not be explicitly accounted for when pricing the derivative. The option is priced relative to the underlying stock.

This also leads to the fundamental concept of *risk-neutral* valuation. Equation (3) can be rewritten as

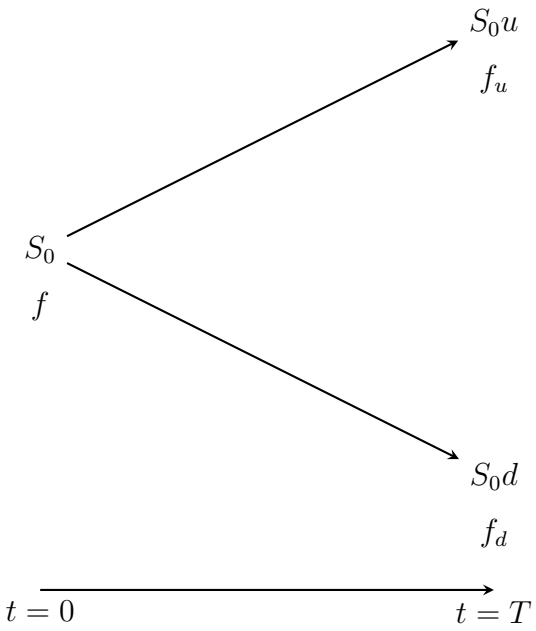


Figure 4: One-period binomial model showing stock price evolution and corresponding option payoffs. The stock price moves from S_0 to either S_0u or S_0d, with corresponding option values f_u and f_d. The replicating portfolio approach determines the option price f without requiring knowledge of the actual probabilities of up or down movements.

$$\frac{1}{1+r}\left(f_u\frac{(1+r)-d}{u-d}+f_d\frac{u-(1+r)}{u-d}\right),$$

where the weights $\tilde{p} := \frac{(1+r)-d}{u-d}$ and $\tilde{q} := \frac{u-(1+r)}{u-d}$ sum to one and lie between 0 and 1 under the no-arbitrage condition $d \leq 1+r \leq u$. These are called risk-neutral probabilities, and the option price can thus be calculated as the expected value of its payoff using the risk-neutral probabilities, discounted at the risk-free rate

$$\frac{1}{1+r}(f_u \cdot \tilde{p} + f_d \cdot (1-\tilde{p})). \tag{4}$$

While there exist many misconceptions about risk-neutral valuation, it can simply be thought of as an algebraic reformulation of the problem into a more familiar version of discounting expected future cash flows. The risk-neutral probabilities are not predictions of any real-life events. They are mathematical weights that look and behave like probabilities, and produce the same price as the replicating portfolio while maintaining the no-arbitrage assumption. The term risk-neutral probability stems from the fact that the reformulation discounts by the risk-free rate, resembling how a risk-neutral investor would discount cash flows. This elegant reformulation enables the use of more tools from probability theory, and elements of it will be found in all of the models to be introduced later. The full derivation of this reformulation is provided in Appendix A. [9] [10]

3 The GPU and Parallel Computing

The graphics processing unit (GPU) is a processor specified for graphics-related processing. Starting in the 1980s, the demand for 2D and 3D graphically based consumer software, mainly driven by video games, surged. In practice, this entailed performing a large amount of floating-point operations per seconds (FLOPS) associated with graphical operations like shading and rasterization. Front-runners of delivering affordable graphics computing capabilities included NVIDIA and ATI Technologies (later acquired by Advanced Micro Devices (AMD)).[12] [13]

Hennessy and Patterson [14] loosely define *latency* as the time between the start and completion of a single event, and *throughput* as the total amount of work performed in a given time. These metrics (and specific variants thereof) are widely used in computing performance contexts. While both central processing units (CPUs) and GPUs have seen performance improvements on both ends, the authors show that relative throughput improvements tend to outweigh relative latency improvements. This is not unreasonable - increasing throughput is (within limits) mostly a manner of providing more of the same resources already available. Latency improvements have, on the other hand, already reached closer to the theoretical limits imposed by the laws of physics [14] [12]. Suomela's summary of statistics on trends for both clock speed and instruction latencies corroborate these claims (cite PPC website?).

This, in combination with the fact that graphics-related calculations are well-structured for concurrency exploitation (covered in Section 3.2), led to the evolution

towards the exceptionally throughput-optimized GPU architectures of today's age. At the same time, there grew an increasing interest in the utilization of these parallel computing capabilities for other purposes. However, these advances alone were insufficient in unlocking the potential for so called general-purpose GPU (GPGPU) programming, as the hardware lacked the flexibility needed for general-purpose computing, and no GPGPU programming models and tools were readily available. The initial graphics-intended hardware and software ecosystems effectively forced developers to mask their problems as those found in the graphics domain. Only after advances in these areas could GPGPU programming truly flourish. In 2007, NVIDIA released the compute unified device architecture *CUDA* and corresponding software tools that greatly removed earlier restrictions in favor of GPGPU-programming. The hardware was now directly accessible without considering graphics-related interfaces, and the software tool for writing CUDA applications was a simple extension layer on top of familiar C/C++. Today, GPGPU applications are found in a wide variety of fields, including computational finance. [12] [13]

3.1 CPU and GPU Architecture

Figure 5 represents a very simple model of a modern CPU and GPU. The *arithmetic logic unit* (ALU) is responsible for performing bitwise operations on data, *registers* (not explicitly visible in figure) for storing memory addresses, instructions, and data to operate on, as well as additional supporting components, e.g. for control of flow, or communication with the external environment. The CPU operates in a loop of two-staged fetch-execute instruction cycles, where an instruction is first loaded into the so-called instruction register. After this, the instruction is decoded and executed, which in practice means either data processing, data transfer, or control logic related to the instruction sequence. This loop continues until the program halts due to either finishing or due to an interruption by another module.[15]

CPUs have with time adapted to improve performance by utilizing various parallelism mechanisms. Instruction-level parallelism (ILP) refers to a set of techniques the processor uses for executing multiple instructions, either in part or fully, simultaneously. Common examples of these are pipelining, branch prediction and speculative execution. Processor architectures often contain *single instruction multiple data* (SIMD) capabilities through large vector registers capable of storing multiples of the same data type, and instructions that can operate on these at once. And to top it off, most modern processors are multi-core, meaning they literally contain multiple instances of the main aforementioned components for use. This enables a higher thread-level parallelism (TLP), where a *thread* — informally defined as an independent sequence of instructions that can be scheduled and executed as a separate unit of work **do I need src??** — can run on each core simultaneously. Despite this, the CPU remains primarily latency-optimized, and does not hold a candle to the typical GPU throughput. [14]

In contrast, while lacking many of the sophistication that makes the CPU excel at sequential and "control-complex" tasks, the GPU fits in more raw computation power for a massively throughput-optimized architecture that excels at the specialized

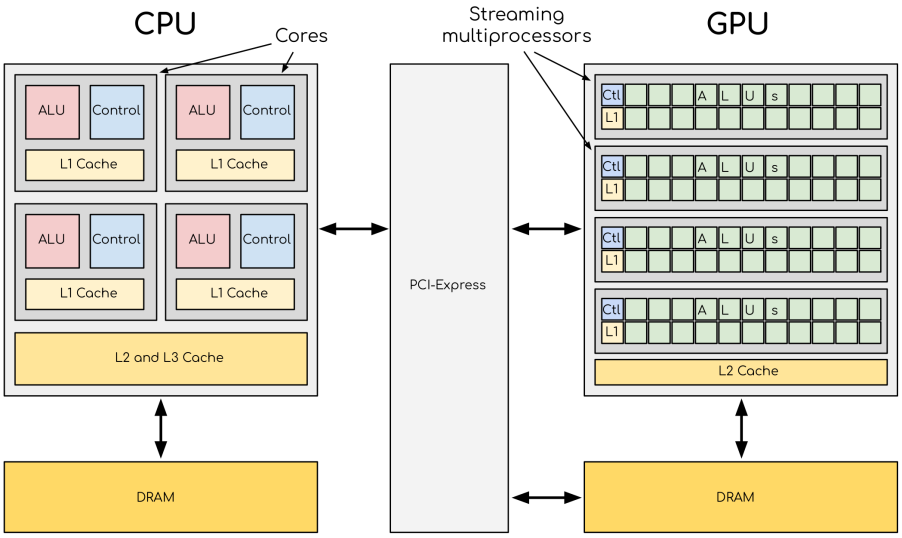


Figure 5: Simple diagrams of modern multi-core CPU and GPU architectures, showcasing their main respective components. Each CPU core contains an arithmetic logic unit (ALU) for performing bitwise operations and a control unit for handling program flow. Hierarchical (L1-L3) on-chip cache memory minimizes data transfer bottlenecks between the (D)RAM and the cores. On the GPU, streaming multiprocessors correspond to cores. The design clearly emphasizes parallel computing power over control and latency optimization. The PCI-Express serves as a data transfer interface between the processors. source: https://enccs.github.io/gpu-programming/2-gpu-ecosystem/ OK TO DO THIS??

task of parallel numerical computation. The term single instruction multiple thread (SIMT) is often used to describe GPU computation, as the concurrency is enabled on thread-level by a massive amount of threads running scalar computations in parallel, as opposed to single-threaded computation using SIMD and ILP methods (or multi-threaded on a comparatively few cores). Suomela (cite PPC) has nicely summarized these differing parallelism paradigms. A modern CUDA-based GPU containing many independent *streaming multiprocessors* (SM), which can be considered as a group of many *streaming processors* (SP) (read: ALUs) sharing some control units and instruction memory registers. Threads are grouped in to groups of 32, called *warps*, which always execute in lock-step, i.e. simultaneous execution on their respective data. The threads in a warp are thus dispatched to a corresponding amount of SPs using these shared resources. Similarly to a multi-core CPU, all SMs share a higher-latency global memory (akin to RAM), and are individually allocated a small but fast on-chip cache for minimizing data transfer bottlenecks, among other things (to be discussed later). [12] [13] (cite PPC)

3.2 Parallel Computing Fundamentals and GPU suitability

Types of parallelism (data, task, instruction) Amdahl's Law and theoretical speedup limits Dependency chains and their impact Synchronization requirements Charac-

teristics of "GPU-friendly" algorithms Identifying parallelizable components MAKE POINT THAT GRAPHICS CALCULATIONS ON PIXESL ETC WERE LARGELY PARALLELIZABLE DUE TO LOW DEPENDENCY CHAINS Common computation patterns in finance Performance metrics and evaluation

3.3 (GP)GPU Programming Model

add * footnote explaining warps vs blocks, link to Suomela PPC Thread hierarchy (threads, blocks, grids) Memory spaces (global, shared, constant, texture)

-> WHY THIS? difference between logical view vs hardware view. Logical for structuring problem (indexing) according to its "natural dimensions" better, and hardware independence no matter how many SMs available et.c. + shared memory constructs for synching work!

Hardware-Level Concerns (Managed by GPU)

Warp formation and execution (32 threads in lockstep) Warp scheduling on available SMs Thread masking during branch divergence Memory coalescing at warp level Register allocation per thread L1/L2 cache management SM resource allocation Load balancing across SMs

Programmer-Level Concerns (Your Focus)

Block and grid dimensions that match problem structure Thread indexing within problem domain Shared memory allocation and usage Managing data transfers between CPU and GPU Minimizing thread divergence Optimizing memory access patterns Balancing resources per thread/block Maximizing occupancy (concurrent warps) Synchronization between threads when needed Decomposing problem into parallel components

The key insight is that programmers think in terms of the problem structure (blocks/grids), while the hardware manages the execution details (warps/SMs). This separation of concerns is what makes CUDA both powerful and accessible.RetryClaude can make mistakes. Please double-check responses.

Execution model and scheduling CUDA programming paradigm

3.4 WHY GPU?

the need for computational power:

This is a crucial question that gets to the heart of why GPU acceleration is valuable in option pricing. The justification comes from several real-world financial applications:

Risk Management: Financial institutions need to calculate Value-at-Risk (VaR) and other risk metrics for portfolios containing thousands of options. This often requires:

Recalculating all option prices under multiple scenarios Computing Greeks to understand sensitivity to market movements Running these calculations daily or even intraday

Regulatory Requirements: Banking regulations like Basel III require comprehensive stress testing of portfolios, involving:

Calculating option values under hundreds of different market scenarios Reporting results within strict timeframes

Electronic Market Making: Firms that provide liquidity in options markets need to:

Continuously update prices for hundreds or thousands of options Adjust their quotes as market conditions change Maintain accurate hedging calculations (based on Greeks)

Portfolio Optimization: Investment firms need to:

Evaluate complex trading strategies across multiple options Test different portfolio compositions Optimize hedging positions using sensitivity metrics

This computational demand is documented in various sources:

Academic papers like "GPU Computing in Finance: Recent Applications and Developments" by Giles & Xiaosong (2014) discuss these requirements Financial industry whitepapers from firms like NVIDIA and JP Morgan Case studies from banks that have implemented GPU solutions for options pricing

The computational demands become especially intense during market stress when quick recalculation is critical, making the case for GPU acceleration even stronger.

- why GPU, why optimize for throughput vs latency

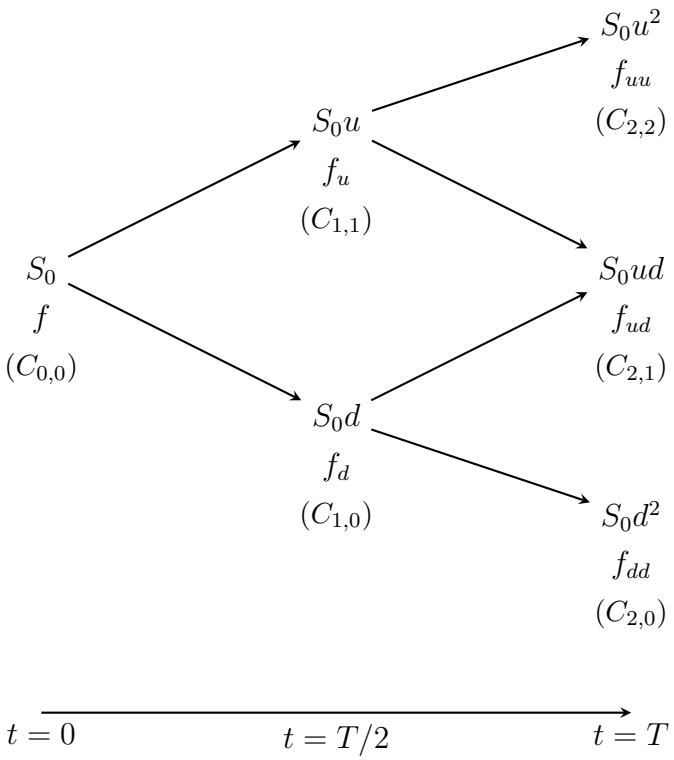


Figure 6: Two-period binomial lattice. Stock prices evolve forward through time, creating an expanding set of possible outcomes. Option payoffs are calculated at the terminal nodes, after which option values at interior nodes can be calculated using Equation (4). The symbols $C_{i,j}$ represent the algorithmic notation for the option values, where i is the time point and j is the number of up-ticks the stock price has moved.

4 GPU Acceleration of the Cox-Ross-Rubinstein Binomial Model

4.1 The Cox-Ross-Rubinstein Model

The Cox-Ross-Rubinstein (CRR) model extends the single-period framework into a multi-period setting, providing a more realistic model of asset price evolution over time. The model maintains the same assumptions as the single-period case: discrete time periods and constant multiplicative factors u and d for price movements. Over multiple periods, this creates a binomial lattice of possible price paths. This lattice expands outward from the initial stock price, generating multiple possible final states for both the stock and the corresponding option payoff. The CRR model solves for the initial option price using backward induction. Starting at the terminal nodes, where option payoffs are calculated using Equations (1) and (2), the algorithm moves backward through the lattice. Each interior node is treated as a one-period binomial problem, and the option value is calculated using Equation (4). Figure 6 visualizes a two-period lattice, but this can be extended to an arbitrary number of periods. [16]

A detail not yet discussed is the choice of the factors u and d. The authors of the original paper have defined $u = e^{\sigma\sqrt{T/n}}$ and $d = e^{-\sigma\sqrt{T/n}}$, where σ is the annualized volatility (e.g. estimated from historical data) of the log-return of the asset, T is the total time in years, and n is the number of time periods. This is useful for several reasons: firstly, it allows for the model to converge to the famous Black-Scholes-Merton (BSM) model as $n \to \infty$. Secondly, the lattice recombines for all permutations of a price path. This ensures computational feasibility as the number of states grows linearly instead of exponentially. Another detail to note is that the discount rate used in the period calculations must be adjusted such that it yields the same return as the initial risk-free rate used in the one-period example. A continuous compounding rate $e^{rT/n}$ is often used in lieu of the discrete version. [16] [5]

The implementation of the CRR model comprises three steps: first, compute the price of the underlying at all the terminal nodes. Then, calculate the corresponding option payoffs for the terminal nodes. Finally, backtrack the lattice to compute the option values at nodes in the previous period, working towards the initial node. The following pseudocode is based on the Python implementation by Jonathon Emerick [17]. Let S_0 be the initial stock price and n the number of periods. The nodes of the recombining lattice can be expressed in terms of indices i and j, where $0 \leq i \leq n$ are points in time, ranging from the start to maturity, and $0 \leq j \leq i$ are the nodes at time point i based on the number of up-ticks j the stock price has moved. In other words, the stock price at each node is defined as $S_{i,j} = S_0 u^j d^{i-j}$. In the same manner, $C_{i,j}$ represents the option price at the corresponding node.

A key advantage of the CRR and other lattice-based models is their natural ability to price American options, which generally cannot be valued using closed-form solutions like the BSM model [11]. For European options, the backward induction step in Algorithm 1 calculates the option value as

$$C[j] \leftarrow disc \cdot (p \cdot C[j+1] + (1-p) \cdot C[j]).$$

For American options, this calculation is modified to include the early exercise possibility. Hence, the option value becomes the maximum of immediate exercise versus holding:

$$C[j] \leftarrow \max(\text{Exercise Value}, disc \cdot (p \cdot C[j+1] + (1-p) \cdot C[j])),$$

where Exercise Value equals $\max(K - S[j], 0)$ for put options or $\max(S[j] - K, 0)$ for call options, representing the payoff from immediate exercise at that node.

4.2 Naive GPU-acceleration

The sequential algorithm has an asymptotic time complexity of $O(n^2)$ due to the nested loop in step 3. Clearly, processing each node at every time step becomes intensive as n grows large. There is a clear dependency chain between the nodes of subsequent time steps, as every node is computed using its children. There are, however, no dependencies among nodes within the same time step, and these

Algorithm 1: CRR European Option Pricing

Data: S_0 (initial stock price), K (strike price), T (time to maturity), r (risk-free rate), n (periods), σ (volatility), type $\in \{\text{CALL}, \text{PUT}\}$
Result: Option price at $t = 0$
$\Delta t \leftarrow T/n$;
$u \leftarrow e^{\sigma\sqrt{\Delta t}}$;
$d \leftarrow 1/u$;
$disc \leftarrow e^{-r\Delta t}$;
$p \leftarrow (e^{r\Delta t} - d)/(u - d)$;
/* Compute stock prices at terminal nodes */
Initialize $S[0..n]$;
$S[0] \leftarrow S_0 \cdot d^n$;
for $j \leftarrow 1$ **to** n **do**
  $S[j] \leftarrow S[j-1] \cdot u/d$;
end
/* Compute option payoffs at terminal nodes */
Initialize $C[0..n]$;
for $j \leftarrow 0$ **to** n **do**
  if *type* $= CALL$ **then**
    $C[j] \leftarrow \max(0, S[j] - K)$;
  end
  else
    $C[j] \leftarrow \max(0, K - S[j])$;
  end
end
/* Backward induction through the lattice */
for $i \leftarrow n - 1$ **to** 0 **do**
  for $j \leftarrow 0$ **to** i **do**
    $C[j] \leftarrow disc \cdot (p \cdot C[j+1] + (1-p) \cdot C[j])$;
  end
end
return $C[0]$;

could at least in theory be computed in parallel. We thus seem to have rather sizable potential speedups for large values of n. Kolb and Pharr [18] implement a GPU-accelerated version of this nature. Similarly to the sequential version, the terminal stock and option values are calculated first, after which the tree backtracking ensues. For each time step, an amount of threads equal to the amount of nodes are spawned, letting each thread compute a single node from its child nodes in the previous iteration. Threads are synchronized in each time step, ensuring no data races occur when moving down the dependency chain. Intermediate results are stored in two alternating arrays from which to read and write data. For maximum GPU utilization the algorithm is run in parallel for "a thousand or so independent options". The full source code (written in Cg as this was in 2005, before even the existence of CUDA) **(consider making this a footnote!)** can be found in the original article [18]. Given enough parallelization, the practical time complexity for a fixed n can thus be reduced to $O(n)$ **(should I use this terminology when it's not asymptotic?)**. The authors used $n = 1024$ and have plotted the achieved throughput (options/s) as a function of the total amount of American options. The sequential CPU version achieves a constant throughput of around 110 **(?see graph)** options/s. The GPU-accelerated version performs worse at roughly $n < 5$, likely due to the overhead associated with GPU code, but increasingly outperforms the CPU version above this threshold. This throughput then tapers off to a a constant around 1150 options/s, yielding a 10x speedup. The limit can be attributed to maximum utilization of all parallel computing capability, or hitting another bottleneck before reaching this point (e.g. in memory bandwidth). Factors like the latter also likely explain the general S-shape curve, as opposed to a linear function **(these are my own conclusions, is this ok?)**.

At large values of n, there is a real possibility of the naive parallelization above running into synchronization overhead bottlenecks between threads. Since each thread reads two child nodes to calculate the current node, the algorithm requires the threads to communicate at every time step. Furthermore, this communication overhead is exacerbated once the lattice is so big it has to be split up over multiple blocks, as inter-block communication is restricted to using the much slower global memory (REWRITE THIS BETTER AFTER FINISHING THE GPU CHAPTER). At some point it is more efficient to sacrifice parallelism for less communication overhead, which can be done by letting a thread compute more nodes over several time steps. While this means more sequential computation, it also avoids having to communicate intermediate values between threads at every time step. A naive approach would be to assign multiple nodes across several periods to each thread. This, however, introduces a substantial amount of redundant calculations, as nodes must be computed multiple times by several threads. More sophisticated approaches have been proposed, such as the "triangle method" by Suo et. al. [19]

4.3 Additional Parallelization Across Time

Ganesan, Chamberlain and Buhler [20] present another approach that aims to utilize parallelism over the time periods in addition to within them. This method focuses

more on pricing a single option, and achieves parallelism over different periods by a clever reformulation of the problem that breaks the direct dependencies between the first and last set of nodes. Equation (4) relates the (European) option values at time i to the prices at the next time point $i+1$. By repeatedly substituting the same formula into itself and expanding, one obtains a general formula that relates the option values at the nodes at a previous time $i-m$ with the current time i:

$$F_{i-m}(j) = e^{-mr\Delta t} \sum_{k=0}^{m} c_k F_i(j+k) \tag{5}$$

where $0 \leq j \leq i-m$ are the nodes at time $i-m$, and the coefficients $c_k, 0 \leq k \leq m$ are the corresponding (risk-neutral) probabilities of the different possible price paths from node j at time $i-m$ to the nodes $j+k$ at time i. Just like the one-period version computes the current value using a weighted sum (risk-neutral expected value) of the values in the future two states, this general formula extends the weighted sum to include all the nodes over several future periods that affect its value. The coefficients in this formula follow the same pattern as the option values themselves, and can be computed through backward induction. Starting from time t itself, where each coefficient represents the direct relationship of a node with itself, these coefficients are built up as we move backward in time. With each step backward, they are updated using the risk-neutral probabilities to reflect all possible paths between the two time points.

The key insight is that while the coefficients in this reformulation still exhibits the same sequential dependency chains between sequential time points, they can be computed independently for different segments of the lattice. Unlike for the actual option values, there is no strict requirement for starting at the terminal nodes when calculating relative option values. Using this, the authors present the following outline of a parallel algorithm:

PSEUDO ALGORITHM

1. Divide the lattice of N time steps into p partitions, each of length N/p 2. For all partitions except the rightmost one, calculate coefficients that relate the option values at the left boundary to those at the right boundary. This is done recursively and in parallel, backtracking from the right edge to the left. Since the rightmost partition already has the option values available at its right edge (terminal nodes) we can simply compute the option prices normally here. 3. Once all partition coefficients have been calculated, use equation (5) to sequentially calculate the option value at each partition boundary, rapidly propagating towards the initial node while "skipping" many intermediate calculations.

Figure ABC depicts the algorithm.
[3]

Again, asymptotically, we still have $O(n^2)$ as we do computations for every node in the tree. The coefficient calculations is directly comparable to the actual option

[3]For what it's worth, the authors state that this can also be applied to American options by "filling in the intermediary nodes". I have been thinking about this for long now, and this still does not make sense to me. I am skeptical to claim that this is doable without having seen a better explanation on the matter. Hence, my performance analysis ignores the contribution of this step.

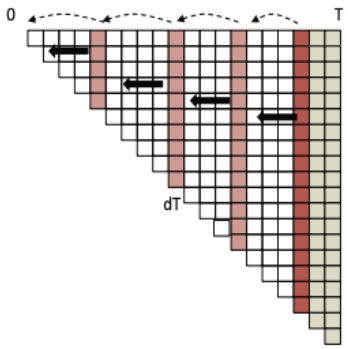


Figure 7: A visualization of the parallelized CRR algorithm using partitioning. BLA BLA BLA lorem ipsum. MAKE YOUR OWN PICTURE OF THIS, OR CHECK IF THE FIGURES ARE OK TO USE UNDER COPYRIGHT

value calculations from the previous version, but we expect a factor p speedup due to performing this work in parallel over all partitions. We then have to calculate the actual option values at all the nodes at the partition boundaries using (5). The sum contains an amount of terms equal to the partition width plus one, $n/p + 1$. Given enough parallel computing resources to calculate this sum for all the nodes simultaneously, this is in total an $O(n)$ operation. In practice, however, the dependency chain of the work has been reduced enough to reduce overall execution time, and the authors report a 2x speedup over a general parallel algorithm. Furthermore, they derive the optimal number of partitions p for minimal execution time, and find it to be directly proportional to the square root of the amount of time steps, but inversely proportional to the square root of the latency of propagation between the partitions. Dividing the lattice into more partitions improves execution time due to parallelism up to a certain point, where the communication and boundary calculation overhead between the partitions exceed the former speedups. The paper compares this tradeoff by plotting execution time of an n=1000 size problem as a function of the number of partitions for different communication latencies (do I mention this, add it in the thesis or what?)[4].

5 GPU Acceleration of Monte Carlo Methods

5.1 Monte Carlo Methods

Monte Carlo (MC) methods are a class of computational algorithms that approximate solutions to problems through random sampling. Unlike analytical and determin-

[4]The original paper simplifies the analysis by only considering the communication overhead between partitions as opposed to also including the option value calculations at the boundaries. The conclusions, however, remain the same, as the choice of p does not affect the computational cost of calculating the option values at the boundaries. More partitions means more boundaries to calculate in, but the smaller partition width means less summation terms. (Do I need some appendix deriving this stuff? This is my analysis based on the paper.)

istic numerical methods that rely heavily on domain-specific mathematical theory, MC methods are relatively simple and model-agnostic. They depend primarily on the *law of large numbers*, which states that the sample average tends toward the true expected value with increasing sample size [21]. Hence, with a well-chosen sampling distribution whose expectation matches the target quantity, the average of a large number of simulated outcomes provides a reliable approximation of the solution. While true random number generation is difficult, there exist many so called pseudorandom number generation algorithms, whose deterministic sequences are statistically indistinguishable from true randomness. To achieve sampling from a specific probability distribution, the general approach is to first use a *linear congruential generator* that samples numbers from a uniform distribution $U(0, 1)$, and then apply a transformation to map these to the desired distribution. A common example is the Box-Muller transform for a standard normal mapping $U(0, 1) \to N(0, 1)$. [22]

Many software libraries contain easy to use random number generators from various distributions. This, coupled with the availability of affordable computing power, has made MC methods hugely popular for brute force problem solving across different domains. Their generality, however, still comes at a significant computational cost. While analytical and other numerical methods exploit problem-specific structure for a faster convergence, MC methods typically require a large number of iterations to produce accurate results. Consequently, MC methods are often employed for high-dimensional problems where traditional approaches fail or become exceedingly slow [22]. In derivatives pricing, MC methods are mostly used for exotic options with complex payoff structures, e.g. multi-asset options whose payoff depend on more than one underlying asset [1, 11].

Option pricing by MC simulation relies on the same risk-neutral valuation framework presented earlier in Section 2.4. The construction and dynamic maintenance of a replicating portfolio naturally extends to continuous-time models. While the BSM model provides an analytical solution to a stochastic differential equation for the expected discounted payoff, the MC approach instead estimates this value numerically by computing the discounted average of many simulated option payoffs, each obtained from a risk-neutral stock price path [23]. The result is always an approximation, but converges toward the theoretical value as the number of simulated paths increases. The stock price dynamics used for MC simulation are typically a discretized version of the *geometric Brownian motion* process used in the BSM model – the same continuous-time limit to which the CRR model also converges.

CONCLUSIONS TABLE WITH ALL RESULTS FROM ALL EXPERIMENTS WITH NOTES OR STANDARDIZAATION SOMEHOW???

Tässä osassa kuvataan käytetty tutkimusaineisto ja tutkimuksen metodologiset valinnat, sekä kerrotaan tutkimuksen toteutustapa ja käytetyt menetelmät.

Tutkimustuloksien merkitystä on aina syytä arvioida ja tarkastella kriittisesti. Joskus tarkastelu voi olla tässä osassa, mutta se voidaan myös jättää viimeiseen osaan, jolloin viimeisen osan nimeksi tulee »Tarkastelu». Tutkimustulosten merkitystä voi arvioida myös »Johtopäätökset»-otsikon alla viimeisessä osassa.

Tässä osassa on syytä myös arvioida tutkimustulosten luotettavuutta. Jos tutkimustulosten merkitystä arvioidaan »Tarkastelu»-osassa, voi luotettavuuden arviointi olla myös siellä.

6 Summary

7 Conclusions

Opinnäytteen tekijä vastaa siitä, että opinnäyte on tässä dokumentissa ja opinnäytteen tekemistä käsittelevillä luennoilla sekä harjoituksissa annettujen ohjeiden mukainen muotoseikoiltaan, rakenteeltaan ja ulkoasultaan.[**?**]

A Derivation of Risk-Neutral Probability Form

In this appendix, we derive the risk-neutral probability form of the one-period binomial model option pricing formula.

Starting from (3):

$$\begin{aligned}
&\frac{f_u - f_d}{S_0(u-d)} S_0 + \frac{f_d \cdot u - f_u \cdot d}{(u-d)(1+r)} \\
&= \frac{f_u - f_d}{u-d} + \frac{f_d \cdot u - f_u \cdot d}{(u-d)(1+r)} \\
&= \frac{(f_u - f_d)(1+r)}{(u-d)(1+r)} + \frac{f_d \cdot u - f_u \cdot d}{(u-d)(1+r)} \\
&= \frac{f_u(1+r) - f_d(1+r) + f_d \cdot u - f_u \cdot d}{(u-d)(1+r)} \\
&= \frac{f_u[(1+r) - d] + f_d[u - (1+r)]}{(u-d)(1+r)} \\
&= \frac{1}{1+r} \cdot \frac{f_u[(1+r) - d] + f_d[u - (1+r)]}{u-d} \\
&= \frac{1}{1+r} \left(f_u \cdot \frac{(1+r) - d}{u-d} + f_d \cdot \frac{u - (1+r)}{u-d} \right)
\end{aligned}$$

This final form reveals the risk-neutral probabilities: $\tilde{p} = \frac{(1+r)-d}{u-d}$ and $\tilde{q} = 1 - \tilde{p} = \frac{u-(1+r)}{u-d}$.

The no-arbitrage condition ensures $\tilde{p}$ and $\tilde{q}$ are correctly bounded by 0 and 1. If $1 + r > u$, investors could profit by shorting the stock and investing in the risk-free asset. Conversely, if $d > 1 + r$, they could earn guaranteed profits. Therefore, $d \leq 1 + r \leq u$ must hold, which leads to $0 \leq \tilde{p}, \tilde{q} \leq 1$. Furthermore, $\tilde{p} + \tilde{q} = 1$. Thus, our risk-neutral probabilities form a valid probability distribution over the two possible outcomes.

References

[1] J. C. Hull and S. Basu, *Options, futures, and other derivatives*. Pearson Education India, 2016.

[2] R. C. Merton, "Influence of mathematical models in finance on practice: past, present and future," *Philosophical Transactions of the Royal Society of London. Series A: Physical and Engineering Sciences*, vol. 347, no. 1684, pp. 451–463, 1994.

[3] W. D. Nordhaus, "Two centuries of productivity growth in computing," *The Journal of Economic History*, vol. 67, no. 1, pp. 128–159, 2007.

[4] S. M. Bartram, G. W. Brown, and F. R. Fehle, "International evidence on financial derivatives usage," *Financial management*, vol. 38, no. 1, pp. 185–206, 2009.

[5] J. Hull, S. Treepongkaruna, D. Colwell, R. Heaney, and D. Pitt, *Fundamentals of futures and options markets*. Pearson Higher Education AU, 2013.

[6] E. F. Fama, "Efficient capital markets," *Journal of finance*, vol. 25, no. 2, pp. 383–417, 1970.

[7] J. B. Berk and P. M. DeMarzo, *Corporate finance*. Pearson Education, 2007.

[8] F. Modigliani and M. H. Miller, "The cost of capital, corporation finance and the theory of investment," *The American economic review*, vol. 48, no. 3, pp. 261–297, 1958.

[9] N. Gisiger, "Risk-neutral probabilities explained," 2010.

[10] J. Tham, "Risk-neutral valuation: A gentle introduction (1)," *Available at SSRN 290044*, 2001.

[11] P. Wilmott, *Paul Wilmott on quantitative finance*. John Wiley & Sons, 2013.

[12] J. Sanders and E. Kandrot, *CUDA by example: an introduction to general-purpose GPU programming*. Addison-Wesley Professional, 2010.

[13] D. B. Kirk and W. H. Wen-Mei, *Programming massively parallel processors: a hands-on approach*. Morgan kaufmann, 2016.

[14] J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach*. Elsevier, 2011.

[15] W. Stallings, *Operating systems: internals and design principles*. Prentice Hall Press, 2011.

[16] 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.

[17] J. Emerick, "youtube-tutorials," https://github.com/TheQuantPy/youtube-tutorials/blob/main/2021/003%20Jul-Sep/2021-07-06%20Binomial%20Option%20Pricing%20Model%20_%20Theory%20_%20Implementation%20in%20Python.ipynb, 2021.

[18] M. Pharr and R. Fernando, *GPU Gems 2: Programming techniques for high-performance graphics and general-purpose computation (gpu gems)*. Addison-Wesley Professional, 2005.

[19] S. Suo, R. Zhu, R. Attridge, and J. Wan, "Gpu option pricing," in *Proceedings of the 8th Workshop on High Performance Computational Finance*, 2015, pp. 1–6.

[20] N. Ganesan, R. D. Chamberlain, and J. Buhler, "Acceleration of binomial options pricing via parallelizing along time-axis on a gpu," in *Proceedings of Symposium on Application Accelerators in High Performance Computing*, 2009.

[21] S. M. Ross, "Front matter," in *Introduction to Probability and Statistics for Engineers and Scientists (Sixth Edition)*, sixth edition ed., S. M. Ross, Ed. Academic Press, 2021, pp. i–iii. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B978012824346600003X

[22] J. E. Gentle, *Random number generation and Monte Carlo methods*. Springer, 2003, vol. 381.

[23] P. P. Boyle, "Options: A monte carlo approach," *Journal of financial economics*, vol. 4, no. 3, pp. 323–338, 1977.