Monte Carlo Integration

for Option Pricing

Emanuele Bellini
Person code: 10633744

Luca Simei
Person code: 10714016

Luca Tramacere
Person code: 10676881

June 2024

POLITECNICO
MILANO 1863

Contents

1 Introduction

1.1 Monte Carlo methods

Monte Carlo methods encompass a broad class of computational techniques that utilize random sampling to derive numerical solutions. These methods are particularly valuable for solving a wide range of problems that might be analytically intractable or computationally intensive using traditional deterministic approaches.

A key strength of Monte Carlo methods is their applicability to high-dimensional spaces and complex systems. They are particularly effective in scenarios where conventional methods struggle, such as in multi-dimensional integrals, optimization problems, and systems with numerous interacting variables.

Monte Carlo methods are inherently non-deterministic, meaning that they produce slightly different outcomes with each execution. This variability can be leveraged to assess the uncertainty and precision of the estimated results, providing valuable insights into the reliability of the solutions. By running multiple simulations, one can obtain a distribution of outcomes that helps quantify the risk and variability inherent in the problem.

The core of Monte Carlo methods involves the random and independent generation of sample points within the problem domain, typically according to a specified probability distribution. This stochastic sampling approach enables the efficient handling of complex problems and supports a wide array of applications across various fields, including physics, engineering, finance, biology, and beyond.

Applications of Monte Carlo methods are diverse and numerous. In physics, they are used to simulate particle interactions and model physical systems at the quantum level. In finance, they help in pricing complex derivatives and assessing risk. In engineering, they assist in reliability analysis and optimization of complex systems. Their versatility and power make Monte Carlo methods indispensable tools in both theoretical research and practical problem-solving, providing robust solutions where traditional methods may fall short.

1.2 Monte Carlo Integration

Building on the general principles of Monte Carlo methods, Monte Carlo Integration is a specific application designed to estimate the integral of a function over a chosen domain using random sampling. This technique is particularly useful for high-dimensional and complex domains where traditional numerical integration methods become impractical.

To illustrate the application of Monte Carlo Integration, consider the task of estimating the integral of a function $f(x, y)$ over a circle with radius R centered at the origin. The area of the circle, $A = \pi R^2$, represents the domain over which we want to compute the integral.

The integral can be approximated using Monte Carlo Integration as follows:

$$\int f(x,y)\,dA \approx A \cdot \frac{1}{N}\sum_{i=1}^{N} f(x_i, y_i)$$

where:

- N is the number of random points generated within the circle.
- (x_i, y_i) are the coordinates of the randomly generated points within the circle.

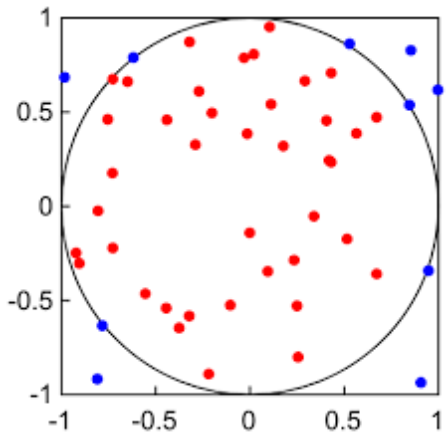

Figure 1: Monte Carlo Integration within a circle

Monte Carlo Integration estimates the integral by calculating the average value of the function $f(x, y)$ at the randomly generated points and then multiplying by the area of the circle.

To estimate the approximation error, we can use the standard deviation of the function values at the sampled points:

$$Error \approx \frac{\sigma}{\sqrt{N}}$$

where:

- σ is the standard deviation of the function values $f(x, y)$ at the randomly generated points.
- N is the number of generated points.

This Monte Carlo Integration approach can be extended to more complex surfaces, such as spheres or other geometric shapes, by appropriately adapting the integration domain and the generation of random points. The flexibility and scalability of Monte Carlo Integration make it a valuable tool for solving integrals in various fields.

2 Option Pricing

2.1 Monte Carlo Integration for Option Pricing

Option pricing, or the valuation of options, constitutes a fundamental pillar in financial practice. It enables the determination of the intrinsic value of a contract that grants its holder the right, but not the obligation, to buy or sell a financial asset (the underlying) at a predetermined price (the strike price) by a specific date (the option's expiration). This process is essential for several reasons.

Firstly, it provides investors with an estimate of the present value of an option, enabling them to make informed decisions about their financial positions. Additionally, option pricing is widely used in risk management, as it allows investors to protect their portfolios from unwanted price fluctuations through option-based hedging strategies.

Moreover, option pricing is a crucial tool for evaluating complex investment projects, such as decisions to invest in new ventures or to abandon unprofitable projects. Through real options analysis, which considers opportunities for expansion, abandonment, or delay, investors can assess the feasibility and added value of certain initiatives.

Another important aspect of option pricing is its utility in identifying arbitrage opportunities. These situations occur when there is a discrepancy between the price of an option and its intrinsic value, allowing investors to make risk-free profits by exploiting this price difference.

Monte Carlo methods play an essential role in option pricing, particularly through Monte Carlo Integration. This approach allows for the evaluation of complex options and estimation of their present value by simulating numerous random scenarios. Monte Carlo Integration is particularly useful when traditional mathematical models cannot be solved analytically or when many interacting variables are present.

Due to the nature of Monte Carlo simulation in option pricing, which involves generating a large number of theoretical prices and taking the average, there is a potential issue with higher volatility in the simulated prices. This means that the simulated prices may exhibit extreme values, leading to a deviation in the simulation results. To address this problem, the Antithetic Variate method has been employed in order to reduce volatility.

By generating a large number of random sample paths for the underlying asset's price, Monte Carlo methods can model the potential outcomes and calculate the expected payoff of the option. This stochastic approach provides a flexible and powerful tool for tackling the complexities inherent in option pricing, making it indispensable in modern financial practice.

2.2 Brownian Motion for Option Pricing

Brownian motion, named after the botanist Robert Brown, describes the random and erratic movement of particles suspended in a fluid. In the context of finance, Brownian motion is used to model the unpredictable and continuous fluctuations in asset prices over time. This stochastic process is fundamental to many financial models, including the pricing of options.

In the context of option pricing, Brownian motion plays a crucial role in modeling the random behavior of asset prices over time. The price dynamics of financial assets are often assumed to follow a stochastic process known as geometric Brownian motion (GBM). This model captures the continuous and erratic nature of asset price movements and forms the foundation for the Black-Scholes option pricing model.

The geometric Brownian motion can be described by the stochastic differential equation:

$$dS_t = \mu S_t \, dt + \sigma S_t \, dW_t$$

where:

- S_t is the price of the asset at time t.
- μ is the drift rate, representing the expected return of the asset.
- σ is the volatility of the asset, indicating the degree of randomness in the price movements.
- W_t is a Wiener process or standard Brownian motion.

To use Brownian motion for option pricing, the asset price paths are simulated over time using the GBM model. These simulations generate a large number of possible future price scenarios, allowing for the estimation of the option's expected payoff. The steps involved in this process are as follows:

1. **Simulate Asset Price Paths**: Using the GBM model, generate multiple price paths for the underlying asset over the option's life. This involves discretizing the time interval and using the formula:
$$S_{t+\Delta t} = S_t \exp\left((\mu - \frac{1}{2}\sigma^2)\Delta t + \sigma\sqrt{\Delta t}\, Z\right)$$
where Δt is a small time increment and Z is a standard normal random variable.

2. **Calculate Payoffs**: For each simulated price path, calculate the payoff of the option at expiration. For a European call option, the payoff is:
$$Payoff = \max(S_T - K, 0)$$
where S_T is the asset price at expiration and K is the strike price.

3. **Discount Payoffs to Present Value**: Discount the payoffs back to the present value using the risk-free interest rate r:
$$DiscountedPayoff = Payoff \times e^{-rT}$$
where T is the time to expiration.

4. **Average the Discounted Payoffs**: Compute the average of the discounted payoffs across all simulated paths to estimate the option's price:
$$OptionPrice = \frac{1}{N}\sum_{i=1}^{N} DiscountedPayoff_i$$
where N is the number of simulated paths.

2.3 The Black-Scholes Formula

The Black-Scholes formula is a cornerstone of modern financial theory, providing a closed-form solution for the pricing of European options. Developed by Fischer Black, Myron Scholes, and Robert Merton in the early 1970s, this model revolutionized the field of finance by offering a method to evaluate options using a mathematical framework grounded in the assumptions of continuous trading and Brownian motion.

At its core, the Black-Scholes model assumes that the price of the underlying asset follows a geometric Brownian motion, characterized by constant drift and volatility. The model also presumes that markets are frictionless, there are no arbitrage opportunities, and the risk-free interest rate is constant over the option's life. These assumptions, while simplifying reality, provide a robust foundation for the analytical derivation of option prices.

The Black-Scholes formula for a European call option is given by:

$$C = S_0 N(d_1) - K e^{-rT} N(d_2)$$

where:

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

$$d_2 = d_1 - \sigma\sqrt{T}$$

Here:

- C is the price of the European call option.
- S_0 is the current price of the underlying asset.
- K is the strike price of the option.
- r is the risk-free interest rate.
- T is the time to expiration of the option.
- σ is the volatility of the underlying asset.
- $N(\cdot)$ is the cumulative distribution function of the standard normal distribution.

For a European put option, the Black-Scholes formula is:

$$P = K e^{-rT} N(-d_2) - S_0 N(-d_1)$$

where P represents the price of the put option, and the other variables are defined as above.

The elegance of the Black-Scholes model lies in its ability to transform the complex problem of option pricing into a manageable analytical formula, enabling traders and financial institutions to price options quickly and consistently. Despite its assumptions, the Black-Scholes formula remains widely used due to its simplicity and the insights it provides into the dynamics of option prices.

In practice, the Black-Scholes model is often used in conjunction with Monte Carlo methods and other numerical techniques to price more complex options and derivatives. The Black-Scholes formula is particularly useful for European options, where it serves as a benchmark for comparing the approximate results obtained through Monte Carlo simulations with analytical predictions. This comparison is especially relevant for single-asset simulations, as the results tend to be more precise. For multiple-asset simulations, the accuracy decreases with the increasing number of assets, making the Black-Scholes formula less effective for validation.

For Asian options, the Black-Scholes formula cannot be applied, and Monte Carlo integration becomes one of the few feasible instruments for pricing, as it can handle the path dependency and complexity inherent in such options.

By integrating the Black-Scholes formula into the broader toolkit of Monte Carlo methods and Brownian motion modeling, financial analysts and traders can approach option pricing with a comprehensive and flexible set of tools, ensuring robust and accurate valuations in a wide range of market conditions.

2.4 Brownian Motion for Option Pricing with Multiple Assets

The concept of Brownian motion can be extended to model the price dynamics of multiple assets simultaneously. In a multi-asset setting, the prices of the assets can be modeled using correlated geometric Brownian motions. This approach is particularly useful for pricing options on multiple underlying assets, such as basket options or options on the minimum or maximum of several assets.

The price dynamics of multiple assets can be described by a system of stochastic differential equations:

$$dS_{i,t} = \mu_i S_{i,t}\, dt + \sigma_i S_{i,t}\, dW_{i,t} \quad for \quad i = 1, 2, \ldots, n$$

where:

- $S_{i,t}$ is the price of the i-th asset at time t.
- μ_i is the drift rate of the i-th asset.
- σ_i is the volatility of the i-th asset.
- $dW_{i,t}$ are Wiener processes that may be correlated.

The correlation between the Wiener processes $dW_{i,t}$ is captured by the correlation matrix ρ, where ρ_{ij} represents the correlation between the i-th and j-th Wiener processes.

The steps to simulate price paths and calculate option prices for multiple assets are similar to those for a single asset, with the additional complexity of handling the correlations between assets.

By leveraging Brownian motion and Monte Carlo methods, it becomes feasible to price complex multi-asset options and account for the interdependencies between different assets. This approach is highly flexible and can be adapted to various types of multi-asset options, making it a powerful tool in modern financial engineering.

2.5 Asian Options

In addition to European options, the Monte Carlo integration method can also be applied to price Asian options. Asian options depend on the average price of the underlying asset over a specified period, rather than its price at maturity. This characteristic introduces additional complexity compared to European options but can be effectively addressed using Monte Carlo methods. By simulating the average asset prices over the option's life and calculating the corresponding payoffs, Monte Carlo integration provides a reliable framework for pricing Asian options. This extension underscores the versatility of Monte Carlo methods in option pricing, allowing for the valuation of a broader range of derivative contracts.

The payoff of an Asian option is typically calculated as the difference between the average price of the underlying asset during the option's life and the strike price, if positive, or zero otherwise. Mathematically, the payoff for a call Asian option can be expressed as:

$$Payoff = \max\left(\left(\frac{1}{n}\sum_{i=1}^{n} S_i\right) - K, 0\right)$$

where:

- S_i is the price of the underlying asset at time i.
- K is the strike price of the option.
- n is the number of observations (or fixing dates) over the option's life.

Similarly, for a put Asian option, the payoff is given by:

$$Payoff = \max\left(K - \left(\frac{1}{n}\sum_{i=1}^{n} S_i\right), 0\right)$$

2.6 Conclusion

In conclusion, option pricing is a fundamental aspect of financial practice, essential for informed decision-making, risk management, evaluating investment projects, and identifying arbitrage opportunities. Monte Carlo methods, particularly Monte Carlo Integration, provide a powerful and flexible approach to estimate the present value of options by simulating numerous random scenarios. By modeling the price dynamics of financial assets as a geometric Brownian motion, we can predict future values and estimate option payoffs with high accuracy.

Brownian motion plays a crucial role in these simulations, offering a robust model for the random behavior of asset prices over time. When applied to multiple assets, correlated geometric Brownian motions can account for the interdependencies between different assets, making it feasible to price complex multi-asset options.

The Black-Scholes formula, grounded in the assumptions of continuous trading and Brownian motion, offers a closed-form solution for European options. Despite its simplifying assumptions, it remains a cornerstone of modern financial theory due to its elegance and practicality. By using

Monte Carlo methods to simulate asset price paths and confirm results with the Black-Scholes formula, we ensure robust and accurate option pricing.

This integrated approach not only enhances the accuracy of predictions, particularly for a lower number of assets, but also demonstrates the practical application of computational methods in finance. Leveraging both Monte Carlo simulations and the Black-Scholes model provides financial analysts and traders with a comprehensive toolkit for tackling the complexities of option pricing in a variety of market conditions.

3 Implementation

3.1 Project Objectives

The project aims to showcase the versatility and practical applications of Monte Carlo Integration through two distinct approaches: a finance-oriented project and a general implementation.

In the finance-oriented project, Monte Carlo Integration serves as a powerful tool for pricing financial options, particularly those dependent on multiple underlying assets. Leveraging the concept of Brownian motion and geometric Brownian motion models, the project simulates asset price paths over time to estimate option payoffs. By computing integrals of hyperrectangles in dimensions corresponding to the number of assets involved, the project facilitates the valuation of complex multi-asset options. These computations are complemented by data extracted from CSV files or provided by the user, offering flexibility and real-world applicability. Moreover, the project explores parallelization techniques to optimize the computation process, ensuring efficient and accurate results even for large volumes of data.

In contrast, the general implementation, presented at the hands-on, focuses on performing Monte Carlo Integration over selected domains, such as the hypercube, hypersphere, and hyperrectangle. This approach offers a comprehensive exploration of Monte Carlo Integration's capabilities across different geometric shapes, allowing for the approximate calculation of integrals in C++ over a variable number of dimensions.

Through these implementations, the project seeks to demonstrate the practical utility and effectiveness of Monte Carlo Integration in both finance and general mathematical contexts, catering to the diverse needs of users ranging from financial analysts to mathematicians and software developers.

3.2 Technological Framework

The project operates within a robust technological framework carefully chosen to facilitate development efficiency and computational performance. Ubuntu 22.04 serves as the development environment due to its intuitive usability and seamless compatibility with essential libraries required for the project's implementation.

Compilation is orchestrated through CMake 3.22.1, ensuring a streamlined and organized build process. GCC 11.4.0 is selected as the compiler for its reliability and comprehensive support for the omp.h library, which plays a pivotal role in parallelizing the random generation of points, a critical aspect of the Monte Carlo Integration method.

To explore and assess parallelization strategies, the project evaluates both OpenMP and CUDA 12.4. OpenMP stands out for its simplicity and efficiency in parallelizing code across multicore architectures, while CUDA offers a GPU-based approach, ideal for handling highly parallel computations efficiently.

The entire codebase is developed using C++17, a language renowned for its versatility and performance in scientific programming. Leveraging C++17's rich feature set, including support for functional and object-oriented programming paradigms, allows for the creation of clear, modular,

and high-performance code, crucial for intricate computational tasks such as Monte Carlo Integration.

In the general implementation of the project, the muParserX library is utilized to enable users to select functions for integration seamlessly. However, as the project evolved towards implementing a robust option pricing model in the finance-oriented version, the focus shifted, leading to the discontinuation of muParserX in favor of a more tailored and efficient approach.

3.3 Code Structure

The codebase for this project is organized to support both the general application of Monte Carlo Integration and its specific use in financial option pricing. This organization emphasizes clarity and modularity, ensuring that the code is maintainable and scalable for future extensions.

The project leverages CMake to manage the build process flexibly and efficiently. With CMake, two main executables are generated: `mainOmp` and `mainCuda`. These executables enable testing of the program with either OpenMP or CUDA, offering flexibility in performance optimization. For CUDA-based tests, the NVIDIA Toolkit and a compatible NVIDIA GPU are required.

The project directory is structured as follows:

- `build/`: This directory contains all the build files generated by CMake. Keeping these files separate from the source code maintains a clean and focused codebase. This directory is not included in the repository, as specified in the `.gitignore` file, and must be built locally.
- `CUDA/`: This directory includes CUDA-specific build configurations and files, facilitating CUDA-based parallel computations.
- `data/`: This folder stores CSV files with asset data extracted from Yahoo Finance, used for option pricing simulations.
- `external/`: This directory includes the external muParserX library and dependencies, ensuring that all required third-party resources are consolidated in one place.
- `include/`: This folder is divided into subfolders for organizing header files, covering general Monte Carlo Integration algorithms, financial option pricing algorithms, and user input management. This separation supports modularity and ease of maintenance.
- `src/`: This folder mirrors the structure of the `include/` folder but contains the corresponding C++ source files. By separating interface declarations (headers) from their implementations (source files), the modularity and readability of the code are enhanced.

This directory structure is based on best practices in software engineering, ensuring modularity, maintainability, and scalability. The separation of concerns between different domains (general integration vs. financial applications) enables clear and efficient development workflows.

The `data` directory's subfolders facilitate organized storage of input data, which is crucial for accurate and flexible simulations. The `include` and `src` directories mirror each other, promoting a

clean architecture where interface declarations are kept separate from their implementations, making debugging and testing more straightforward.

Parallelization using OpenMP and CUDA is crucial for optimizing the computational efficiency of Monte Carlo methods. OpenMP is employed for its straightforward implementation and effectiveness in utilizing multi-core CPU environments. In contrast, CUDA leverages the parallel processing power of NVIDIA GPUs, significantly accelerating the simulation processes by handling thousands of concurrent threads. This dual approach ensures that the project can efficiently handle various computational loads, whether on multi-core CPUs or powerful GPUs.

This structured approach ensures that the project is not only easy to maintain and extend but also capable of adapting to different computational environments. By carefully organizing the code and leveraging powerful parallel computing techniques, the project achieves both clarity and high performance.

3.4 Code Operation and Execution

The code operates in a streamlined and user-interactive manner, beginning with the execution of the `mainOmp` function. Upon launch, the user is prompted to choose between simulating general Monte Carlo integration over a specified domain or delving into the financial application for option pricing. The focus here will be on the latter.

When the finance application is selected, the `financeComputation` method in `optionpricer.cpp` is invoked. This method orchestrates various sub-methods to execute the primary algorithm. Throughout this process, the user is prompted for several choices, each managed by specific methods.

Firstly, the `getOptionTypeFromUser` method allows the user to select between European and Asian options. European options can only be exercised at expiration, while Asian options consider the average price of the underlying asset over a certain period, which can provide a more stable and less manipulable option value.

Following this, the `getAssetCountTypeFromUser` method enables the choice of whether the simulation will involve a single asset or multiple assets. It is noteworthy that the precision of the simulation is typically higher with a lower number of assets, as demonstrated by the Black-Scholes formula. Single-asset simulations are more straightforward and tend to yield more accurate results, whereas multiple-asset simulations introduce additional complexity and potential imprecision due to correlations and increased dimensionality.

Next, the `loadAssets` method reads CSV files from the data folder and populates the Asset objects accordingly. This step ensures that the simulation uses realistic and up-to-date market data. The payoff function, which determines the financial outcome based on the asset prices, is then constructed using the `createPayoffFunction` method. This function is crucial for evaluating the option's value at maturity.

To establish the hyperrectangle domain for the simulation, the `getIntegrationBounds` method generates it based on the assets and their count, thereby setting the stage for the Monte Carlo sim-

ulation. This domain defines the range over which the asset prices can vary during the simulation.

The Monte Carlo simulation itself is executed 10 times, with each iteration generating 10^5 points. These points are generated in parallel using OpenMP pragmas. The `generateRandomPoints` method is designed to generate each dimension for each point in parallel, effectively avoiding race conditions by utilizing a critical pragma. By iterating the Monte Carlo integration 10 times, the program obtains a better mean of the results, improving the accuracy and reliability of the approximations. Monte Carlo methods are particularly powerful for option pricing because they can handle the stochastic nature of asset prices and provide estimates for options that do not have closed-form solutions.

Finally, the Black-Scholes formula is computed to ensure that the approximate result is consistent and accurate, especially for single-asset simulations, since the results tend to be more precise compared to multiple-asset simulations. For European options, this provides a benchmark for comparing the approximate results obtained through Monte Carlo simulations with analytical predictions. For Asian options, the Black-Scholes formula cannot be applied, and Monte Carlo integration becomes one of the few feasible instruments for pricing, as it can handle the path dependency and complexity inherent in such options. The results, including the precision indicated by the error and the time of execution, are then written to an `output.txt` file for further analysis. This structured approach, combining user interactivity with efficient parallel computation, ensures robust and precise option pricing simulations.

In addition to the primary functionality, the codebase includes `mainCUDA`, an executable generated using CMake, which allows users to leverage an NVIDIA GPU for enhanced performance through CUDA. By executing `mainCUDA`, users can compare the performance of GPU-based computations with the CPU-based parallel processing provided by OpenMP. CUDA can significantly accelerate the simulation by parallelizing the workload across the GPU's many cores, making it ideal for handling the large number of random points generated during Monte Carlo simulations.

For those interested in the general application of Monte Carlo integration, the codebase also supports this functionality. Users can choose to integrate over a freely chosen function, facilitated by the muParserX library, and specify a domain of their choice, including hypercube, hyperrectangle, and hypersphere configurations. This flexibility allows for a broad range of applications beyond financial computations, demonstrating the versatility and power of Monte Carlo methods in solving complex integrals and simulations across various fields.

4 Results

4.1 Outcomes

The project utilized the Monte Carlo integration method to price financial options, conducting simulations for both single and multiple assets. Through the implementation, we were able to make detailed comparisons of the outcomes produced under various optimization levels while leveraging OpenMP and CUDA for parallel processing.

In the case of single asset simulations, the results were remarkably accurate, aligning closely with the theoretical values provided by the Black-Scholes formula. This high level of precision was attributed to the relatively low complexity of modeling a single asset, which resulted in reduced variance in the Monte Carlo estimates. The simulations conducted across different optimization levels, including O0, O1, O2, O3, and Ofast, demonstrated consistent accuracy. Notably, the Ofast optimization level not only provided the fastest execution times but did so without any significant loss of accuracy, highlighting its efficiency.

Conversely, multiple asset simulations presented a greater challenge due to the increased dimensionality and the need to account for inter-asset correlations. Despite this complexity, the Monte Carlo integration method proved robust, delivering reliable option pricing. The performance gains from higher optimization levels were particularly evident in these simulations, underscoring the crucial role of compiler optimizations in handling more computationally demanding tasks.

4.2 Numerical Results

The numerical results for European and Asian options, for both single and multiple assets, were systematically documented. These results highlighted the impact of different optimization flags and the effectiveness of Monte Carlo simulations in option pricing. For instance, the European option pricing for a single asset showed decreasing errors and faster execution times with increasing optimization levels, from O0 to Ofast.

The presented data reflects multiple program executions utilizing OpenMP and various optimization levels. To ensure precision and reliability, numerous simulations were conducted. This methodological approach offers a comprehensive understanding of the program's performance across different optimization configurations.

Table 1: European Option Pricing Results (Single Asset) - Part 1

Points	Error (Ofast)	Payoff (Ofast)	Time (Ofast)	Error (O0)	Payoff (O0)	Time (O0)
10,000	0.016858	4.948278	0.074697	-	-	-
100,000	0.004155	4.940459	0.479514	0.016818	4.986042	1.103399
1,000,000	0.001343	4.943498	4.381570	0.006945	5.045334	11.024888
10,000,000	0.000384	4.942043	46.930865	-	-	-

Table 2: European Option Pricing Results (Single Asset) - Part 2

Points	Error (O1)	Payoff (O1)	Time (O1)	Error (O2)	Payoff (O2)	Time (O2)
10,000	-	-	-	-	-	-
100,000	0.004764	4.943998	0.462706	0.004103	4.940822	0.424841
1,000,000	0.001592	4.941425	4.453893	0.001397	4.944275	4.106172
10,000,000	-	-	-	-	-	-

Table 3: European Option Pricing Results (Single Asset) - Part 3

Points	Error (O3)	Payoff (O3)	Time (O3)
10,000	-	-	-
100,000	0.004288	4.942004	0.438480
1,000,000	0.001103	4.939415	4.524580
10,000,000	-	-	-

Table 4: European Option Pricing Results (Multiple Assets) - Part 1

Points	Error (Ofast)	Payoff (Ofast)	Time (Ofast)	Error (O0)	Payoff (O0)	Time (O0)
10,000	0.074586	27.319413	0.183077	-	-	-
100,000	0.020111	27.322424	1.487495	0.109775	29.001092	5.818025
1,000,000	0.010860	27.416204	14.885225	0.039648	30.140890	63.529781
10,000,000	0.002847	27.406125	171.477199	-	-	-

Table 5: European Option Pricing Results (Multiple Assets) - Part 2

Points	Error (O1)	Payoff (O1)	Time (O1)	Error (O2)	Payoff (O2)	Time (O2)
10,000	-	-	-	-	-	-
100,000	0.035986	27.358937	1.647054	0.025174	27.351031	1.419641
1,000,000	0.006652	27.310613	17.042342	0.008600	27.355243	18.547455
10,000,000	-	-	-	-	-	-

Table 6: European Option Pricing Results (Multiple Assets) - Part 3

Points	Error (O3)	Payoff (O3)	Time (O3)
10,000	-	-	-
100,000	0.030187	27.392773	1.553511
1,000,000	0.007408	27.312033	16.303890
10,000,000	-	-	-

Table 7: Asian Option Pricing Results (Single Asset) - Part 1

Points	Error (Ofast)	Payoff (Ofast)	Time (Ofast)	Error (O0)	Payoff (O0)	Time (O0)
10,000	0.003764	2.659256	0.090097	-	-	-
100,000	0.001357	2.655184	0.561960	0.008158	2.678853	1.400950
1,000,000	0.000763	2.656907	5.364755	0.002841	2.682425	11.847951
10,000,000	0.000256	2.656802	45.260789	-	-	-

Table 8: Asian Option Pricing Results (Single Asset) - Part 2

Points	Error (O1)	Payoff (O1)	Time (O1)	Error (O2)	Payoff (O2)	Time (O2)
10,000	-	-	-	-	-	-
100,000	0.001702	2.656587	0.513874	0.001528	2.655457	0.507939
1,000,000	0.000944	2.658118	4.463348	0.000794	2.656065	4.134283
10,000,000	-	-	-	-	-	-

Table 9: Asian Option Pricing Results (Single Asset) - Part 3

Points	Error (O3)	Payoff (O3)	Time (O3)
10,000	-	-	-
100,000	0.001235	2.654678	0.527509
1,000,000	0.000820	2.657390	4.364036
10,000,000	-	-	-

Table 10: Asian Option Pricing Results (Multiple Assets) - Part 1

Points	Error (Ofast)	Payoff (Ofast)	Time (Ofast)	Error (O0)	Payoff (O0)	Time (O0)
10,000	0.031576	14.689813	0.186293	-	-	-
100,000	0.008638	14.686162	1.522627	0.066186	15.512163	6.227299
1,000,000	0.006579	14.722350	15.737213	0.027596	15.882251	66.314966
10,000,000	0.001688	14.716924	170.954029	-	-	-

Table 11: Asian Option Pricing Results (Multiple Assets) - Part 2

Points	Error (O1)	Payoff (O1)	Time (O1)	Error (O2)	Payoff (O2)	Time (O2)
10,000	-	-	-	-	-	-
100,000	0.013464	14.683690	1.749617	0.010992	14.692391	1.527129
1,000,000	0.004193	14.665599	19.223399	0.005510	14.708005	16.135343
10,000,000	-	-	-	-	-	-

Table 12: Asian Option Pricing Results (Multiple Assets) - Part 3

Points	Error	Payoff	Time
10,000	-	-	-
100,000	0.022505	15.103968	1.585776
1,000,000	0.004146	14.744058	17.069430
10,000,000	-	-	-

To confront the performance obtained with and without OpenMP parallelization, the serial execution of the various possible simulations of the project has been analyzed:

Table 13: European Option Pricing Results (Serial Execution) - Single Asset

Points	Error	Payoff	Time
10,000	0.009968	4.935917	0.001549
100,000	0.003587	4.937828	0.013542
1,000,000	0.001758	4.947218	0.136650
10,000,000	0.000434	4.946424	1.371626

Table 14: European Option Pricing Results (Serial Execution) - Multiple Assets

Points	Error	Payoff	Time
10,000	0.071997	27.347423	0.004937
100,000	0.025581	27.346945	0.047387
1,000,000	0.011261	27.404828	0.473355
10,000,000	0.002773	27.369339	5.195766

Table 15: Asian Option Pricing Results (Serial Execution) - Single Asset

Points	Error (O3)	Payoff (O3)	Time (O3)
10,000	0.005729	2.457136	0.190957
100,000	0.002054	2.458880	1.892366
1,000,000	0.000952	2.461380	18.731896
10,000,000	-	-	-

Table 16: Asian Option Pricing Results (Serial Execution) - Multiple Assets

Points	Error (O3)	Payoff (O3)	Time (O3)
10,000	0.050078	13.631498	0.870556
100,000	0.015541	13.622734	8.823234
1,000,000	0.005250	13.608565	87.756155
10,000,000	-	-	-

In the pursuit of enhancing computational efficiency and harnessing the power of parallel processing, CUDA (Compute Unified Device Architecture) emerges as a prominent framework for accelerating numerical computations on NVIDIA GPUs. Leveraging CUDA for option pricing tasks enables substantial speedups, particularly in Monte Carlo simulations. This section presents the results of European and Asian option pricing, both for single and multiple assets, employing CUDA-based computations across different numbers of points:

Table 17: European Option Pricing Results (CUDA) - Single Asset

Points	Variance	Final Price	Time
10,000	0.106337	5.234325	1.931632
100,000	0.033627	5.231685	1.283737
1,000,000	0.010609	5.279687	1.926679
10,000,000	0.003217	5.279687	1.531401
100,000,000	0.000522	5.279687	4.191018

Table 18: European Option Pricing Results (CUDA) - Multiple Assets

Points	Variance	Final Price	Time
10,000	0.305407	28.121069	1.211986
100,000	0.096571	28.121909	1.284757
1,000,000	0.030479	27.705841	1.349557
10,000,000	0.009592	27.705841	1.469689
100,000,000	0.001457	27.705841	9.139033

Table 19: Asian Option Pricing Results (CUDA) - Single Asset

Points	Variance	Final Price	Time
10,000	0.106465	2.954506	1.298278
100,000	0.033655	2.953772	0.622970
1,000,000	0.010618	2.948552	0.612464
10,000,000	0.003220	2.948552	2.367895
100,000,000	0.000523	2.948552	12.255583

Table 20: Asian Option Pricing Results (CUDA) - Multiple Assets

Points	Variance	Final Price	Time
10,000	0.301759	54.862915	0.617824
100,000	0.095419	54.797688	1.303360
1,000,000	0.030116	54.489738	1.704406
10,000,000	0.009476	54.489738	5.418788
100,000,000	0.001379	54.489738	42.520080

4.3 Performance Analysis

To understand the performance implications of different optimization flags in conjunction with OpenMP, we conducted thorough profiling, revealing several key insights.

Firstly, execution time varied significantly depending on the level of optimization applied. With no optimization (O0), the simulations took the longest to run, reflecting the absence of performance enhancements. Moderate improvements were observed at the O1 and O2 levels, with O2 showing a substantial speed increase over O1 due to more aggressive optimization strategies. The highest levels of optimization, O3 and Ofast, offered the best performance, with Ofast slightly outperforming O3 thanks to advanced optimization techniques that minimized execution times.

Parallel efficiency, facilitated by OpenMP, lead to an increasing performance in the case of the simulation for Asian Option Pricing, but not in the one for the European Option Pricing. In order to avoid data races, that would have lead to inaccurate results, several critical pragmas were introduced, reason why the execution time of the simulation have suffered. Moreover, another issue that has been observed is the overhead introduced by the creation of the threads. No great improvements has been observed with respect to the serial execution with optimization flags. From this the chioce to adopt also the implementation of the CUDA code, which allowed to obtain better performances in all the possibile cases of simulation of the project. It's noteworthy that by comparing execution times across different numbers of threads, it has been observed a notable reduction in execution time as thread count increased, demonstrating good scalability. However, beyond a certain point, the benefits of additional threads diminished due to the overhead associated with thread management.

The parallelization of the computational workload of the Monte Carlo simulation for financial option pricing with OpenMP leveraged the multi-core architecture of the Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz.

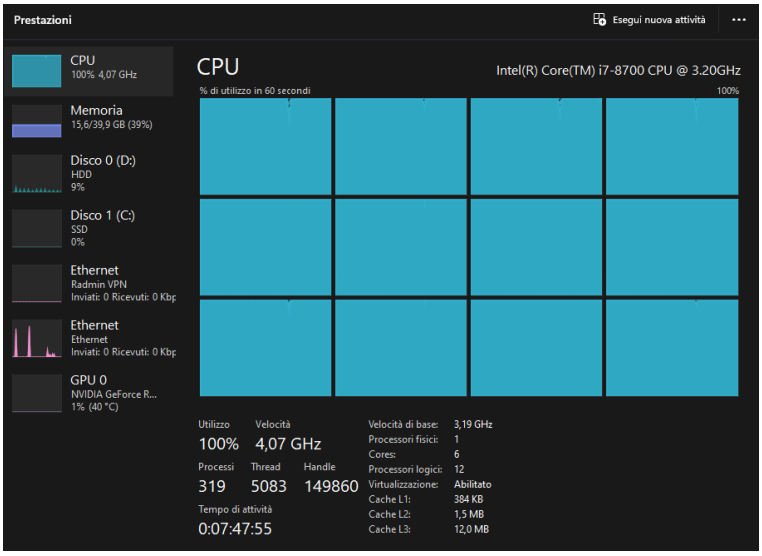

Figure 2: CPU Utilization

As illustrated in the CPU performance screenshot, the CPU utilization reached 100% across all cores when running the parallelized tasks. The high utilization rate demonstrates the effectiveness of OpenMP in maximizing the CPU's processing power. The CPU maintained a speed of approximately 4.07 GHz, which is above its base speed of 3.19 GHz, indicating that the processor was operating in a turbo-boost mode to handle the increased computational demand.

4.4 Error Analysis

The accuracy of the Monte Carlo integration was evaluated through error analysis. This standard error, was used to assess the precision of the simulations. By plotting the error values against the number of points on a log-log scale, we confirmed the inverse square root relationship, indicating that increasing the number of samples significantly reduces the error.

The error in Monte Carlo integration can be estimated using the standard deviation of the sampled values. The error, often referred to as the standard error, decreases with the square root of the number of samples. The formula for the error

$$Error \approx \frac{\sigma}{\sqrt{N}}$$

where:

- σ is the standard deviation of the sample values.
- N is the number of generated points.

Given this formula, the error is inversely proportional to the square root of the number of samples. This implies that if the number of samples increases by a factor of 100, the error decreases by a

factor of 10.

To create a graphical representation of the error, we'll use the provided error values and plot them against the number of points (samples) on a log-log scale.

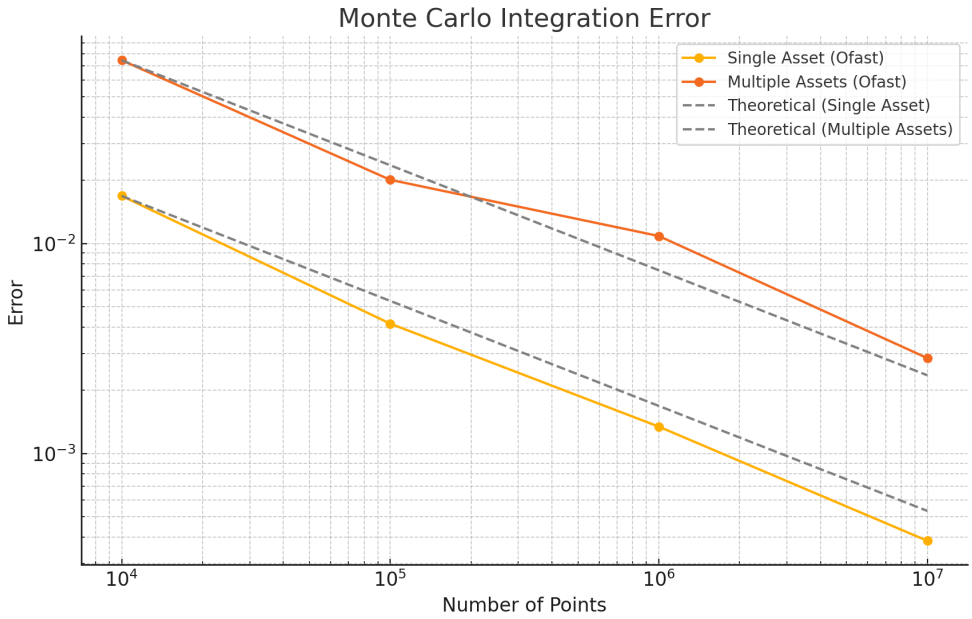

Figure 3: Monte Carlo Integration Error

In the figure above, the dashed lines represent the theoretical error reduction based on initial errors for both single and multiple assets. The plot confirms that the error decreases as expected with an increase in the number of simulation points.