UNIVERSIDADE DE CAXIAS DO SUL
ÁREA DO CONHECIMENTO DE CIÊNCIAS EXATAS E ENGENHARIAS

ALAN FANTIN

SIMULAÇÃO DE MONTE CARLO EM GPU PARA PREVISÃO DO PREÇO DE AÇÕES

CAXIAS DO SUL

2025

ALAN FANTIN

SIMULAÇÃO DE MONTE CARLO EM GPU PARA PREVISÃO DO PREÇO DE AÇÕES

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Bacharel em Ciência da Computação na Área do Conhecimento de Ciências Exatas e Engenharias da Universidade de Caxias do Sul.

Orientador: Prof. Dr. André L. Martinotto

CAXIAS DO SUL

2025

ALAN FANTIN

SIMULAÇÃO DE MONTE CARLO EM GPU PARA PREVISÃO DO PREÇO DE AÇÕES

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Bacharel em Ciência da Computação na Área do Conhecimento de Ciências Exatas e Engenharias da Universidade de Caxias do Sul.

Aprovado em 24/11/2025

BANCA EXAMINADORA

Prof. Dr. André L. Martinotto
Universidade de Caxias do Sul - UCS

Prof. Dr. Daniel Luis Notari
Universidade de Caxias do Sul - UCS

Prof. Me. Samuel Francisco Ferrigo
Universidade de Caxias do Sul - UCS

“Risco vem de você não saber o que está fazendo.”

Warren Buffett

RESUMO

O mercado financeiro oferece diversas oportunidades para a geração e manutenção de riqueza por meio do investimento em empresas cujo preço da ação apresenta potencial de valorização. Assim, torna-se cada vez mais importante a utilização de técnicas capazes de estimar o preço futuro de uma ação. Neste sentido, a simulação de Monte Carlo, associada ao movimento geométrico browniano, forma uma abordagem eficaz para estimar o comportamento futuro dos preços das ações, unindo a aleatoriedade da simulação à modelagem estocástica baseada na taxa de retorno e na volatilidade da ação. De fato, juntas elas integram dados históricos e variações aleatórias, modelando as incertezas do mercado e oferecendo uma visão das probabilidades de diferentes cenários futuros. No entanto, para alcançar previsões mais precisas, a simulação de Monte Carlo exige um alto número de simulações, o que resulta em um alto custo computacional. Neste trabalho, foi desenvolvida uma versão paralela do método de Monte Carlo, com o uso de GPU com o objetivo de acelerar a execução da simulação. Para validação da implementação foram selecionadas as ações mais relevantes dos cinco setores com maior peso no Índice Ibovespa: Itaúsa S.A. (ITSA4), Eletrobras (ELET3), Vale S.A. (VALE3), Petróleo Brasileiro S.A. (PETR4) e WEG S.A. (WEGE3). Ademais, a abordagem foi avaliada usando o próprio índice Ibovespa (IBOV). Os resultados demonstraram que os preços reais dos ativos se mantiveram dentro do intervalo de confiança de 95% em todos os cenários. A implementação paralela em GPU se mostrou numericamente consistente com a abordagem sequencial e obteve um *speedup* de até $130{,}2\times$ para $1.000.000$ de simulações.

Palavras-chave: Previsão do Preço de Ações. Simulação de Monte Carlo. Unidade de Processamento Gráfico. CUDA.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ALGORITMOS

LISTA DE ABREVIATURAS E SIGLAS

CPU *Central Processing Unit*

GPU *Graphics Processing Unit*

CUDA *Compute Unified Device Architecture*

ARIMA *Autoregressive Integrated Moving Average*

JCP Juros sobre capital próprio

ANN *Artificial Neural Networks*

GBM *Geometric Brownian Motion*

RNN *Recurrent Neural Network*

SMC Simulação de Monte Carlo

KLCI *Kuala Lumpur Composite Index*

DJIA *Dow Jones Industrial Average*

SP500 *Standard & Poor's 500*

NSE *National Stock Exchange of India*

FCD Fluxo de Caixa Descontado

SISD *Single Instruction Single Data*

SIMD *Single Instruction Multiple Data*

MISD *Multiple Instruction Single Data*

MIMD *Multiple Instruction Multiple Data*

ALU *Arithmetic Logic Unit*

SM *Streaming Multiprocessor*

GPC *Graphics Processing Cluster*

RT *Ray Tracing*

TB *Terabyte*

GB *Gigabyte*

MB *Megabyte*

KB *Kilobyte*

GHz *Gigahertz*

MHz *Megahertz*

TFLOP *Tera Floating Point Operations*

SSD *Solid State Drive*

RAM *Random Access Memory*

DDR4 *Double Data Rate 4*

DRAM *Dynamic Random Access Memory*

WSL *Windows Subsystem for Linux*

NVMe *Non-Volatile Memory Express*

CSV *Comma-Separated Values*

OpenCL *Open Computing Language*

SUMÁRIO

1 INTRODUÇÃO

Investidores adquirem ações visando obter um retorno financeiro, seja por meio da valorização dos papéis ou pelo recebimento de dividendos, que representam a distribuição dos lucros das empresas aos acionistas. As ações representam frações do patrimônio das empresas, cujo valor pode ser estimado por diferentes métodos, incluindo a análise do patrimônio líquido, o fluxo de caixa descontado e outros indicadores que medem o potencial de geração de riqueza da empresa (NETO, 2018; BREALEY; MYERS; ALLEN, 2020; OLBERT, 2024).

Segundo Vuong *et al.* (2024a), a previsão do preço de uma ação pode ser realizada por meio de diferentes técnicas, entre as quais se destacam: as técnicas baseadas em estatísticas; o reconhecimento de padrões com aprendizado de máquina; e a análise de sentimentos. As técnicas baseadas em estatística se baseiam na utilização de modelos matemáticos que representam o comportamento histórico dos preços dos ativos, sendo que dentre essas destaca-se o modelo ARIMA (*Autoregressive Integrated Moving Average*) (KYPER, 2011; ADEBIYI; ADEWUMI; AYO, 2014; PRAKHAR *et al.*, 2022). Por outro lado, as abordagens baseadas em reconhecimento de padrões com uso de aprendizado de máquina diferenciam-se pela capacidade de capturar características mais complexas, sendo que os métodos mais empregados são as redes neurais artificiais (ANN do inglês *Artificial Neural Networks*) e as redes neurais recorrentes (RNN do inglês *Recurrent Neural Network*) (WANG; GUO, 2020; CHEN *et al.*, 2021). Por fim, as técnicas baseadas na análise de sentimentos caracterizam-se pela análise de textos e notícias que afetam o mercado. Tradicionalmente, essas técnicas utilizam-se de abordagens baseadas na mineração de dados e processamento de linguagem natural (SHAH; ISAH; ZULKERNINE, 2019; PADMANAYANA; VARSHA; K, 2021; KABBANI; USTA, 2022).

Entre as técnicas estatísticas, uma que recebe destaque é a simulação de Monte Carlo, pois permite a execução de diferentes instâncias de modelos estatísticos, em conjunto com a adição de coeficientes aleatórios às entradas. Esse processo possibilita modelar a incerteza e a variabilidade, gerando uma gama de resultados possíveis em vez de uma única previsão. Dessa forma, torna-se possível calcular a probabilidade associada a diversos cenários (RAYCHAUDHURI, 2008; DOAN *et al.*, 2010; ESTEMBER; MARAñA, 2016; SIDDIAUI; PATIL, 2018). Diferentes estudos na literatura evidenciam o potencial do uso da simulação de Monte Carlo para a previsão do preço de ações, apontando sua eficiência quando comparada às outras técnicas (ESTEMBER; MARAñA, 2016; ARNAUT-BERILO; ZAIMOVIć; Nedzmija Turbo-Merdan, 2017; SIDDIAUI; PATIL, 2018; NAGARAJAN; PRABHAKARAN, 2019; PARUNGROJRAT; KIDSOM, 2019; SINGH, 2021).

No entanto, para se obter previsões mais precisas, é necessário executar a simulação de um grande número de cenários, cada uma com elevado número de iterações, o que apresenta um elevado custo computacional (SEVRANI, 2011; KOçAK, 2020; POUDEL, 2022). Uma estratégia

amplamente adotada para mitigar esse elevado custo computacional é a utilização da programação paralela, que consiste na execução simultânea de diversas tarefas (ALERSTAM; SVENSSON; ANDERSSON-ENGELS, 2008; ANDERSON *et al.*, 2012). Essa abordagem é especialmente vantajosa quando as tarefas podem ser executadas de maneira quase que independente, como ocorre nas simulações que utilizam o método de Monte Carlo (KIRKBY; DELPY, 1997; SEVRANI, 2011; LEYVA-SUáREZ; HERRERA; de la Cruz, 2015; WEIGEL, 2017).

Uma das arquiteturas paralelas que tem recebido destaque nos últimos anos são as unidades de processamento gráfico (GPUs do inglês *Graphics Processing Unit*). Atualmente, as GPUs estão presentes em diversos computadores de uso pessoal e se destacam pelo elevado número de núcleos de processamento, oferecendo desempenho superior em relação às unidades de processamento central (CPUs do inglês *Central Processing Unit*), especialmente em tarefas que executam múltiplos cálculos simultaneamente (JUAN; RAFAEL; JUNIOR, 2012; LONDHE *et al.*, 2013; NAVARRO; HITSCHFELD; MATEU, 2013; RAM, 2023; CHUNDURU, 2024).

Neste trabalho foi desenvolvida uma implementação paralela da simulação de Monte Carlo para prever o preço de ações. Essa implementação foi desenvolvida para ser executada em uma arquitetura de GPU, utilizando a plataforma CUDA (*Compute Unified Device Architecture*). Para avaliação do modelo, foram utilizados ativos disponíveis na bolsa de valores brasileira B3. A seleção dos ativos foi feita de acordo com a relevância das empresas dentro do Ibovespa, principal índice da bolsa brasileira. A implementação paralela foi avaliada por meio de uma comparação entre o tempo de execução da implementação sequencial e da paralela. Os testes foram realizados utilizando uma GPU *NVIDIA RTX 4060*.

1.1 OBJETIVOS

Este trabalho apresentou como principal objetivo desenvolver uma versão paralela em GPU da simulação de Monte Carlo para a previsão do preço de ações. Para alcançar este objetivo, os seguintes objetivos específicos foram realizados:

- Desenvolver uma implementação sequencial da simulação de Monte Carlo para a previsão dos preços de ações;
- Definir as ações a serem utilizadas na avaliação do modelo;
- Avaliar o modelo considerando a capacidade de prever a variação nos preços das ações;
- Desenvolver uma versão paralela do modelo para GPU;
- Comparar o tempo de execução entre a implementação sequencial e a paralela.

1.2 ESTRUTURA DO TRABALHO

A estrutura deste trabalho está organizada da seguinte forma:

- Este capítulo apresentou uma breve introdução ao trabalho, destacando sua motivação e objetivos.
- O Capítulo 2 introduz o método de simulação de Monte Carlo para a previsão do preço de ações, além de apresentar os trabalhos relacionados.
- O Capítulo 3 descreve a arquitetura de uma GPU, bem como o modelo de programação utilizado nesse tipo de arquitetura.
- O Capítulo 4 descreve a implementação desenvolvida e a paralelização para GPUs.
- O Capítulo 5 apresenta os resultados obtidos.
- Por fim, no Capítulo 6 são apresentadas as considerações finais e as sugestões de trabalhos futuros.

2 PREVISÃO DO VALOR FUTURO DE AÇÕES

O mercado financeiro de ações possibilita que as empresas disponibilizem partes do seu capital, denominadas ações, para investidores que buscam adquirir participação nessas organizações (NETO, 2018). O mercado de ações reflete as particularidades da economia de cada país, sendo influenciado tanto por fatores internos quanto pelo cenário mundial (REIS; MEURER; SILVA, 2008). No contexto brasileiro, esse mercado é estruturado principalmente pela B3,[1] a bolsa de valores que concentra a negociação de ações de empresas de diversos setores, incluindo tecnologia, agronegócio e energia. A bolsa de valores se trata de um espaço essencial para a captação de recursos por parte das empresas para os mais diversos objetivos, como, por exemplo, o desenvolvimento de projetos de longo prazo (NYASHA; ODHIAMBO, 2013).

Ao mesmo tempo, o mercado financeiro consiste em uma oportunidade para os investidores que desejam obter retornos financeiros por meio da valorização de ações ou pelo recebimento de dividendos (NETO, 2018). A valorização das ações ocorre quando o mercado, de forma consensual, reconhece que determinada empresa possui um valor de mercado maior que o atual. Com isso, cada fração do seu capital, representada pelas ações, passa a ter um valor mais elevado (LEE; MYERS; SWAMINATHAN, 1999). Além disso, o lucro de uma empresa pode ser repassado aos acionistas de diversas formas, sendo que as principais são os dividendos e os Juros sobre capital próprio (JCP). Os dividendos são uma parte do lucro líquido da empresa e a sua distribuição para os acionistas é livre de impostos no Brasil. O JCP é uma outra forma de remuneração a qual pode ser considerada como uma despesa dedutível para a empresa, reduzindo assim o valor do imposto de renda e da contribuição social devidos sobre o lucro líquido. Por outro lado, para o acionista, um determinado percentual é tributado na fonte (SANTOS, 2007).

Nos últimos anos houve um crescimento expressivo no número de investidores nas bolsas de valores de diversos países (World Economic Forum, 2025). Esse fenômeno também é observado no Brasil, onde, por exemplo, no quarto trimestre de 2024, o número de pessoas físicas que investem em ações aumentou 6% em relação ao mesmo período do ano anterior (B3, 2024). Assim, cresceu também o interesse pelo desenvolvimento de novos métodos e ferramentas que auxiliem esses investidores na tomada de decisão e na previsão dos valores futuros das ações (DUARTE; GONZáLEZ; CRUZ, 2020; KUMAR; SARANGI; VERMA, 2022).

2.1 MÉTODOS PARA PREVISÃO DO VALOR FUTURO DE AÇÕES

Por ser uma área de grande interesse para o posicionamento das carteiras dos investidores, diversos métodos foram desenvolvidos objetivando prever o comportamento futuro dos preços das ações. Segundo Vuong *et al.* (2024a) esses métodos podem ser divididos em:

[1] <https://www.b3.com.br/>

- Métodos estatísticos: baseiam-se em dados históricos para a previsão do valor futuro das ações. Uma abordagem bastante utilizada é o modelo ARIMA, o qual determina o valor futuro de uma série temporal (neste caso, o preço da ação) através de uma combinação linear dos valores passados e dos erros em previsão anteriores (KYPER, 2011; ADEBIYI; ADEWUMI; AYO, 2014; PRAKHAR *et al.*, 2022). Segundo Vuong *et al.* (2024b), embora modelos como o ARIMA apresentem bons resultados em previsões de curto prazo, esses métodos tendem a perder a eficiência em períodos mais longos, devido principalmente à limitações em capturar as tendências não lineares do mercado de ações.

- Métodos baseados em aprendizado de máquina: esses métodos conseguem capturar características não lineares no comportamento dos preços das ações, sendo que os métodos mais empregados são as redes neurais artificiais (ANN do inglês *Artificial Neural Networks*) e as redes neurais recorrentes (RNN do inglês *Recurrent Neural Network*). Segundo Wang e Guo (2020) e Chen *et al.* (2021), essas técnicas podem apresentar uma baixa capacidade de generalização, não conseguindo tratar adequadamente à alta variabilidade do mercado acionário.

- Métodos baseados na análise de sentimentos: utilizam-se de técnicas de mineração de dados e processamento de linguagem natural para analisar notícias e informações em redes sociais. A partir dessas informações buscam prever as oscilações e o valor futuro das ações (PADMANAYANA; VARSHA; K, 2021; KABBANI; USTA, 2022). Esses métodos tornam-se atrativos visto que os preços das ações são diretamente afetados por notícias sobre o mercado (NOFSINGER, 2005; BOLLEN; MAO; ZENG, 2011; BUJARI; FURINI; LAINA, 2017). No entanto, essas abordagens enfrentam desafios, como a obtenção de dados de qualidade (PADMANAYANA; VARSHA; K, 2021; KABBANI; USTA, 2022) e a possibilidade de manipulações nas fontes de dados (BOLLEN; MAO; ZENG, 2011).

Os métodos de previsão do preço futuro de ações buscam fornecer estimativas sobre o comportamento e o valor de uma ação. No entanto, uma característica fundamental do mercado de ações é sua natureza incerta e a dificuldade em prever um valor único (REDDY; CLINTON, 2016; XIANG; VELU; ZYGIARIS, 2021). Para abordar essa incerteza e explorar uma gama de resultados possíveis, a Simulação de Monte Carlo (SMC) emerge como uma técnica complementar. Em vez de buscar uma única previsão pontual, a SMC utiliza uma amostragem aleatória para gerar um elevado número de trajetórias possíveis para o valor de uma ação (JAMES D. WHITESIDE, 2008; YANG; ALDOUS, 2015; ESTEMBER; MARAñA, 2016). Assim, essa abordagem permite analisar a distribuição dos valores possíveis, auxiliando na avaliação de riscos e na tomada de decisão (SONONO; MASHELE, 2015; BRăTIAN *et al.*, 2022).

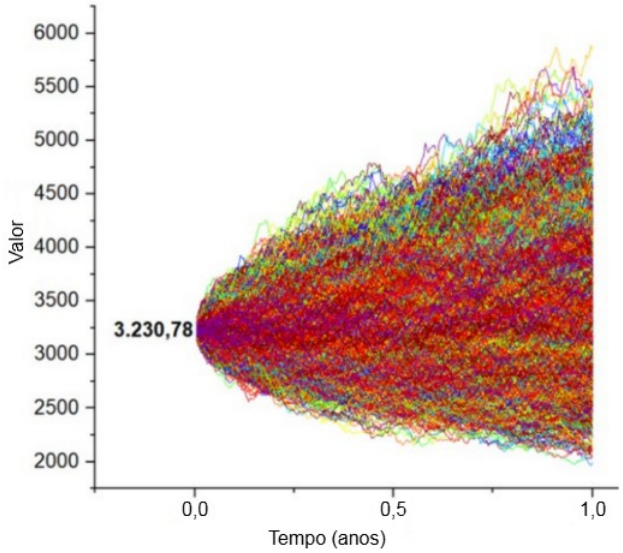
2.2 SIMULAÇÃO DE MONTE CARLO

A SMC pode ser definida como um método que se baseia na amostragem aleatória e na análise estatística para estimar os resultados (SIDDIAUI; PATIL, 2018). Essa capacidade de lidar com a incerteza através da repetição permite que a SMC seja vista como uma abordagem sistemática para a análise de múltiplos cenários hipotéticos. Desta forma, a SMC possibilita analisar inúmeras variações no comportamento do valor futuro de uma ação (ALRABADI; ALJARAYESH, 2015; SIDDIAUI; PATIL, 2018; MILOš; DIGKOGLOU, 2022).

Por exemplo, considere o objetivo de estimar o preço de uma ação em um intervalo de tempo escolhido. Com base em dados históricos, é possível calcular seu retorno médio e sua volatilidade. Contudo, o mercado é influenciado por inúmeros fatores aleatórios que tornam uma previsão única inadequada. A SMC aborda essa incerteza ao simular milhares de trajetórias de preço. Cada simulação parte do preço atual e projeta um caminho futuro, aplicando o retorno médio histórico acrescido de uma variação aleatória a cada dia. Ao final do processo, em vez de um único valor, obtém-se uma distribuição de probabilidades dos preços futuros, permitindo determinar intervalos de confiança e avaliar os riscos associados aos melhores e piores cenários.

A Figura 1 ilustra a aplicação da SMC para a previsão de possíveis cenários para a evolução dos preços de uma ação. Cada linha colorida representa a simulação de um cenário hipotético, demonstrando as possíveis flutuações no preço de uma ação. Inicialmente, todas as simulações partem de um valor comum e, a partir daí, evoluem de acordo com os coeficientes aleatórios introduzidos. Desta forma, observa-se que a SMC é capaz de modelar a incerteza dos preços, fornecendo um conjunto de possíveis trajetórias ao invés de uma previsão única. Observa-se ainda que, à medida que o tempo avança, a dispersão entre os cenários se amplia, evidenciando o aumento da incerteza e a maior dificuldade em realizar previsões de longo prazo.

Figura 1 – Simulação de diversos cenários com Monte Carlo



Fonte: Adaptado de Brătian *et al.* (2022)

Com a execução de uma SMC, torna-se possível realizar uma análise estatística dos resultados, estimando a probabilidade de diferentes cenários de preços e definindo intervalos de confiança para os valores futuros (YANG; ALDOUS, 2015; BRăTIAN *et al.*, 2022). Por exemplo, ao observar as diversas trajetórias apresentadas na Figura 1, pode-se quantificar a chance de o preço da ação ultrapassar um determinado patamar ou permanecer dentro de uma faixa específica. Este tipo de informação, oferece uma perspectiva quantitativa sobre os riscos e oportunidades futuras, auxiliando na tomada de decisão (ALRABADI; ALJARAYESH, 2015; SIDDIAUI; PATIL, 2018; MILOš; DIGKOGLOU, 2022).

No contexto da precificação de ações, a SMC é amplamente utilizada em conjunto com modelos de processos estocásticos [2], ou seja, que consideram a aleatoriedade de determinados parâmetros. Dentre esses, o mais utilizado e conhecido é o modelo GBM (*Geometric Brownian Motion*) (SONONO; MASHELE, 2015). Para sua aplicação do GBM na precificação de ativos, é necessário ajustar determinados parâmetros do modelo com base em dados históricos das ações (RATHGEBER; STADLER; STöCKL, 2015; REDDY; CLINTON, 2016).

2.2.1 Movimento Browniano Geométrico

O GBM é um modelo que procura descrever como os valores de uma ação variam em função do tempo. Esse é descrito como um processo estocástico de tempo contínuo, que modela a evolução do preço de uma ação considerando uma tendência de retorno médio e um componente aleatório de volatilidade. Ambos os componentes são estimados a partir dos dados históricos do ativo. O retorno médio esperado é, frequentemente, calculado a partir da média dos retornos históricos de uma ação, enquanto o componente aleatório de volatilidade é obtido a partir do desvio padrão desses retornos históricos (JAMES D. WHITESIDE, 2008; BRăTIAN *et al.*, 2022). Assim, o modelo utiliza o desempenho passado da ação para projetar as possíveis trajetórias de preço (JAMES D. WHITESIDE, 2008; BRăTIAN *et al.*, 2022).

O modelo GBM descreve que a variação infinitesimal (uma mudança infinitamente pequena) do preço de uma ação (dS_t), em um tempo t, pode ser obtida a partir da Equação 2.1. Nesta, μ representa o retorno médio esperado; S_t é o preço da ação no tempo t; dt corresponde ao incremento infinitesimal de tempo; σ é o componente aleatório de volatilidade; e dB_t é a fonte de aleatoriedade do modelo, que segue um movimento browniano que busca capturar as flutuações imprevisíveis.

$$dS_t = \mu S_t \, dt + \sigma S_t \, dB_t \tag{2.1}$$

A Equação 2.1 representa a variação do preço em tempo contínuo. Porém, as simulações computacionais exigem uma expressão explícita para valor de S_t, que neste caso só pode ser

[2] Modelos matemáticos que descrevem sistemas ou fenômenos que evoluem ao longo do tempo de forma aleatória

obtida por meio do cálculo estocástico, devido à presença do termo aleatório dB_t (OKSENDAL, 2013). Uma abordagem comum é analisar a evolução do logaritmo do preço ($\ln(S_t)$), aplicando o Lema de Itô (OKSENDAL, 2013), uma ferramenta do cálculo estocástico que generaliza as regras de diferenciação para processos aleatórios. A aplicação do Lema de Itô a $\ln(S_t)$, seguida da integração ao longo do tempo, resulta na Equação 2.2, que descreve a variação do logaritmo do preço entre o instante inicial ($t = 0$) e o tempo t (SONONO; MASHELE, 2015; YANG; ALDOUS, 2015; BRăTIAN *et al.*, 2022).

$$\ln S_t - \ln S_0 = \left(\mu - \frac{1}{2}\sigma^2\right) t + \sigma B_t \tag{2.2}$$

Aplicando a função exponencial em ambos os lados da equação, elimina-se o logaritmo obtendo-se a Equação 2.3 (SONONO; MASHELE, 2015; ESTEMBER; MARAñA, 2016; BRăTIAN *et al.*, 2022).

$$S_t = S_0 \exp\left[\left(\mu - \frac{\sigma^2}{2}\right) t + \sigma B_t\right] \tag{2.3}$$

Ao aplicar a Equação 2.3 a um intervalo de tempo finito e discreto, de t a $t + \Delta t$, e considerando que o incremento browniano $(B_{t+\Delta t} - B_t)$ pode ser representado como $\varepsilon\sqrt{\Delta t}$, onde $\varepsilon \sim \mathcal{N}(0, 1)$ é uma variável aleatória com uma distribuição normal padrão, obtém-se a Equação 2.4. Essa é a forma discreta do GBM, sendo frequentemente utilizada nas SMC (SONONO; MASHELE, 2015; REDDY; CLINTON, 2016; XIANG; VELU; ZYGIARIS, 2021; BRăTIAN *et al.*, 2022).

$$S_{t+\Delta t} = S_t \exp\left[\left(\mu - \frac{\sigma^2}{2}\right) \Delta t + \sigma\varepsilon\sqrt{\Delta t}\right] \tag{2.4}$$

A determinação dos parâmetros μ e σ é realizada a partir dos valores históricos das ações. O parâmetro μ geralmente é calculado através da média dos retornos logarítmicos diários (ou em outra frequência). Ou seja, inicialmente é calculado o retorno logarítmico $r_i = \ln(S_i/S_{i-1})$ para todos os dias i e, posteriormente, o valor de μ é obtido a partir da média desses retornos (JAMES D. WHITESIDE, 2008; PARUNGROJRAT; KIDSOM, 2019; BRăTIAN *et al.*, 2022). O parâmetro de volatilidade σ corresponde ao desvio padrão dos retornos logarítmicos r_i (PARUNGROJRAT; KIDSOM, 2019; XIANG; VELU; ZYGIARIS, 2021). Em ambos os casos, se os retornos forem calculados em uma frequência menor que a anual (diária ou mensal), é comum anualizar esses valores, multiplicando-os pelo número de observações correspondentes a um ano (por exemplo, 252 dias de negociação) (JAMES D. WHITESIDE, 2008).

2.3 TRABALHOS RELACIONADOS

No trabalho de Arnaut-Berilo, Zaimović e Nedzmija Turbo-Merdan (2017) foi realizada uma comparação da SMC com o modelo ARIMA para a previsão dos preços futuros de 5 ações do índice DJIA (*Dow Jones Industrial Average*). As ações avaliadas foram das empresas Pfizer, Coca-Cola, General Electric, Exxon e 3M. A SMC foi implementada com base no modelo GBM, sendo realizadas 1000 simulações em diferentes cenários para estimar os preços. Para o desenvolvimento dos modelos foram utilizados os preços históricos de março de 2006 até março de 2016. Os modelos foram avaliados considerando 21 dias de negociação, no período de 10 março a 10 de abril de 2016. Os períodos de avaliação foram divididos em janelas de tempo de 5, 10, 15 e 21 dias. Os resultados mostraram que o método ARIMA apresentou uma ligeira vantagem em períodos de tempo mais curtos (5 dias). No entanto, a SMC demonstrou superioridade em todas as 5 ações para períodos mais longos (21 dias).

No trabalho desenvolvido por Singh (2021) foi realizada uma avaliação da SMC para prever os preços futuros de 15 ações selecionadas do NSE (*National Stock Exchange of India*). Essas ações estavam divididas em três setores: bancário, hoteleiro, mídia e entretenimento. Para estimar a distribuição dos preços futuros, foi utilizada uma forma simplificada do modelo GBM e foram realizadas simulações com 1000 cenários para cada uma das ações. O estudo utilizou dados históricos, porém não foram especificadas as datas. Os modelos foram avaliados considerando a distribuição dos preços calculados no 30º dia. Nas conclusões, os autores destacaram que as previsões utilizando SMC apresentaram um bom desempenho, representando uma boa alternativa para antecipar o comportamento dos preços das ações.

No trabalho de Xiang, Velu e Zygiaris (2021) foi realizada a previsão do valor futuro de dois índices de mercado. Mais especificamente foram realizadas SMC para a previsão dos índices KLCI da Malásia (*Kuala Lumpur Composite Index*) e S&P 500 (*Standard & Poor's 500*) dos Estados Unidos. As simulações foram realizadas utilizando o modelo GBM com 1000 cenários para cada um dos índices. Os parâmetros do modelo foram determinados utilizando os preços dos 5 anos anteriores à data de início da simulação, que foi feita a partir de 16 de dezembro de 2020. Os autores concluíram que as simulações para o S&P 500 indicaram uma tendência de alta, superando o KLCI, que demonstrou uma trajetória lateral (onde o preço seria mantido) ou de baixa. Posteriormente, observou-se que o índice americano de fato valorizou-se acima de 15%, enquanto o índice da Malásia demonstrou uma queda superior a 10%.

No trabalho de Miloš e Digkoglou (2022), a SMC é utilizada para uma avaliação dos preços futuros das ações da companhia Apple. O estudo descreve a utilização de modelos estocásticos, mas não especifica se foi utilizado o GBM ou outro modelo. Neste trabalho, foram realizadas simulações de 1000 cenários e os parâmetros do modelo foram ajustados utilizando os valores diários das ações da Apple durante um período de seis anos, compreendido entre 2017 e 2022. As projeções consideraram um horizonte de 120 dias, e os resultados foram avaliados considerando as probabilidades de diferentes níveis de retorno. Os resultados apresentaram

uma probabilidade de $55,64\%$ para um retorno superior a 10% e de $12,69\%$ para perdas acima de 10%. Neste período, os preços apresentaram um aumento de $10,74\%$, confirmando assim as projeções apresentadas no artigo.

No trabalho desenvolvido por Siddiaui e Patil (2018) é apresentado um sistema de avaliação de ações para orientar investimentos de longo prazo. O trabalho combina SMC com um Fluxo de Caixa Descontado (FCD) para estimar o valor futuro de ações, priorizando empresas com baixo endividamento. Para o desenvolvimento do modelo, foram utilizados dados históricos de cinco anos, no período de 2011 a 2015. O período de avaliação foi realizado de março de 2016 a março de 2017. Os resultados apresentaram uma comparação entre o desempenho de ações recomendadas e não recomendadas. Esses resultados indicaram que as ações recomendadas apresentaram um crescimento médio de 20,52%, significativamente superior ao das demais ações, que foi de 8,27%.

2.4 DEFINIÇÃO DO MODELO

A Quadro 1 apresenta um resumo dos trabalhos relacionados. Neste pode ser observado que a SMC é uma ferramenta versátil, com aplicabilidade em diversos contextos, desde a previsão de preços de curto e longo prazo para ações individuais (ARNAUT-BERILO; ZAIMOVIć; Nedzmija Turbo-Merdan, 2017; SINGH, 2021; MILOš; DIGKOGLOU, 2022), até a previsão de índices de mercado (XIANG; VELU; ZYGIARIS, 2021).

Quadro 1 – Sistematização dos trabalhos relacionados

Referência	**Ativos Analisados**	**Modelo**	**Tempo Histórico**	**Tempo Previsto**
Arnaut-Berilo, Zaimović e Nedzmija Turbo-Merdan (2017)	5 ações do índice DJIA: Pfizer, Coca-Cola, General Electric, Exxon e 3M	ARIMA vs. SMC	10-30 anos	21 dias
Siddiaui e Patil (2018)	+60 ações de baixo endividamento recomendadas por corretoras indianas	SMC e Fluxo de Caixa Descontado	5 anos	3 a 5 anos
Xiang, Velu e Zygiaris (2021)	Índices S&P 500 (EUA) e KLCI (Malásia)	SMC com GBM	5 a 10 anos	1000 dias
Singh (2021)	15 ações da NSE (setores de bancos, hotelaria e mídia)	SMC	1 ano	30 dias
Miloš e Digkoglou (2022)	Ações da Apple (AAPL)	SMC	6 anos	120 dias

Fonte: O Autor (2025).

Observa-se ainda no Quadro 1 uma ampla utilização do GBM e a realização de simulações em um elevado número de cenários, frequentemente 1.000 ou mais. Desta forma, neste trabalho também foram realizadas simulações utilizando um modelo GBM. Além disso, foram

realizadas um número elevado de simulações por ativo, a fim de analisar como a quantidade de cenários influencia na estimativa dos preços. Para o desenvolvimento, foram utilizados os dados históricos de um período de quatro anos, compreendendo de janeiro de 2021 a dezembro de 2024, de forma a excluir o período de alta volatilidade durante a pandemia de COVID-19. A avaliação foi realizada considerando um horizonte de 120 dias, compreendendo o intervalo de janeiro a junho de 2025.

A realização de um grande número de simulações, necessária para aprimorar a precisão das estimativas, gera um alto custo computacional (SEVRANI, 2011; KOçAK, 2020; POUDEL, 2022). Desta forma, nesse trabalho foi desenvolvida uma implementação paralela da SMC para GPUs, com o objetivo de acelerar a simulação e possibilitar a utilização de um número maior de cenários. A SMC é por natureza paralelizável, visto que as simulações para cada cenário podem ser realizadas de forma independente. Essa característica de independência torna os métodos de Monte Carlo adequados para implementação em arquiteturas paralelas (CASTILLO *et al.*, 2014). Optou-se pela utilização de GPUs, visto que essas possuem a capacidade de executar milhares de operações de ponto flutuante simultaneamente (ANDERSON *et al.*, 2012). Além disso, com a popularização dos jogos eletrônicos, as GPUs tornaram-se mais acessíveis e hoje estão presentes em diversos computadores pessoais (MOURNING *et al.*, 2010; COOK, 2013).

3 UNIDADES DE PROCESSAMENTO GRÁFICO

Em 1965, Gordon Earle Moore, cofundador da Intel, previu que o número de transistores em *chips* eletrônicos dobraria a cada dois anos sem um aumento significativo de custo, uma tendência que se confirmou ao longo do tempo e ficou conhecida como Lei de Moore (MOORE, 2006). Esse avanço foi impulsionado pela capacidade de miniaturização dos transistores, permitindo que os fabricantes de microprocessadores aumentassem a densidade desses componentes por unidade de área e, consequentemente, o poder de processamento (BURG; AUSUBEL, 2021).

No entanto, nos últimos anos, os fabricantes de processadores têm enfrentado desafios cada vez maiores para aumentar o poder computacional, devido principalmente a limitações físicas. A miniaturização dos transistores, embora tenha impulsionado avanços significativos, também gerou problemas, como a dificuldade de dissipação de calor e a crescente dificuldade na comunicação entre os componentes (BROCK, 2006). Diante dessas limitações, a indústria passou a investir no desenvolvimento de arquiteturas paralelas, que se caracterizam pela utilização de múltiplos processadores trabalhando de forma cooperativa para aumentar o desempenho computacional (STALLINGS, 2017).

3.1 TAXONOMIA DE FLYNN

Uma das primeiras e mais conhecidas formas de classificar as arquiteturas de computadores foi proposta por Flynn (1966). Essa ficou conhecida como taxonomia de Flynn, e organiza os computadores, sejam eles paralelos ou sequenciais, com base no número de fluxos de instruções e do conjunto de dados processados. A partir dessas características, os sistemas computacionais foram agrupados em quatro categorias:

- *Single Instruction Single Data* (SISD): modelo no qual um único fluxo de instruções opera sobre um único conjunto de dados. Essa é uma arquitetura puramente sequencial, onde as operações são realizadas uma de cada vez. Computadores equipados com processadores de um único núcleo (*single core*) pertencem a essa categoria.
- *Single Instruction Multiple Data* (SIMD): arquitetura caracterizada por um único fluxo de instrução que é executado sobre múltiplos dados. A arquitetura SIMD foca no paralelismo em nível de dados, mas mantém uma estrutura fundamentalmente sequencial. Esta abordagem é encontrada em muitos processadores vetoriais e matriciais, sendo as GPUs um exemplo recente dessa categoria.

- *Multiple Instruction Single Data* (MISD): essa arquitetura é definida pela execução de múltiplas instruções sobre um único conjunto de dados. Não há consenso sobre exemplos práticos desta arquitetura, sendo que muitos especialistas a consideram como uma categoria teórica.

- *Multiple Instruction Multiple Data* (MIMD): caracteriza-se pela execução simultânea de múltiplos fluxos de instruções sobre múltiplos fluxos de dados. Exemplos dessa categoria incluem, os processadores modernos com múltiplos núcleos (*multicore*), computadores multiprocessados e *clusters* de computadores[1].

Os sistemas paralelos também podem ser categorizados com base na organização da memória e na comunicação entre os núcleos de processamento, sendo geralmente divididos em arquiteturas de memória compartilhada e memória distribuída. Arquiteturas com memória compartilhada permitem que os múltiplos núcleos acessem e manipulem uma mesma região de memória, facilitando a comunicação e sincronização entre os processos. Já na memória distribuída, cada núcleo possui sua própria memória, e a comunicação ocorre por meio de troca de mensagens, através de uma rede de computadores (IBBETT, 1982; OWENS *et al.*, 2008).

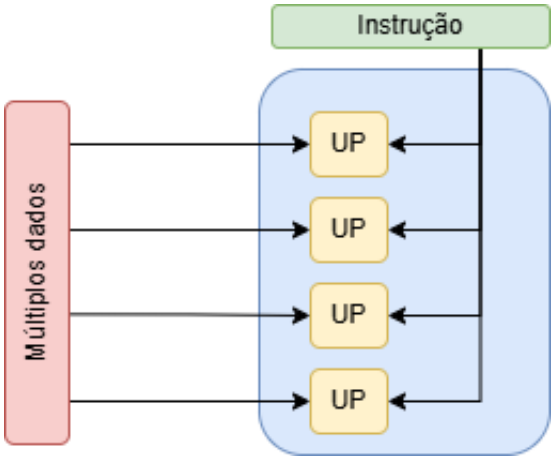
3.1.1 Modelo SIMD

O modelo SIMD foi projetado para explorar o paralelismo em nível de dados, aplicando uma mesma instrução a múltiplos dados, embora o fluxo de controle permaneça sequencial. Isso se diferencia do modelo tradicional (SISD). Por exemplo, ao somar dois vetores de n elementos, o modelo SISD realiza n operações de leitura (uma para cada elemento), n somas e n operações de escrita. Já no modelo SIMD, todos os dados são lidos em uma única operação, a instrução de soma é aplicada a todos os elementos simultaneamente, e os resultados são gravados em uma única operação de escrita. Esse processamento em blocos acelera significativamente aplicações que manipulam de forma uniforme grandes volumes de dados (IBBETT, 1982; AMIRI; SHAHBAHRAMSI, 2020).

Conforme pode ser observado na Figura 2, um sistema SIMD é gerenciado por uma única unidade de controle (UC), a qual é responsável por buscar, decodificar e despachar as instruções para um conjunto de unidades de processamento (UP). Assim, cada uma das UPs executa uma mesma instrução sobre um conjunto diferente de dados. A comunicação e a sincronização entre as múltiplas unidades de processamento, bem como o acesso aos dados, são frequentemente realizadas por meio de uma região de memória compartilhada.

[1] Grupos de computadores interconectados que trabalham em conjunto, dividindo tarefas para melhorar desempenho

Figura 2 – Arquitetura SIMD



Fonte: O Autor (2025)

Dentre os processadores SIMD, destacam-se duas categorias: os processadores vetoriais (*vector processors*) e os processadores matriciais (*array processors*) (BASU, 2016). Os processadores vetoriais são mais versáteis, pois conseguem realizar operações tanto com vetores quanto com valores escalares. Exemplos dessa categoria incluem supercomputadores como o *CRAY-1*, *CDC STAR-100* e *CDC CYBER 205*(HWANG; SU; NI, 1981; IBBETT, 1982). Já os processadores matriciais dependem de uma unidade de controle externa ou de um processador hospedeiro para executar operações sobre escalares, sendo o supercomputador *Illiac IV* um dos precursores dessa tecnologia (HWANG; SU; NI, 1981; IBBETT, 1982).

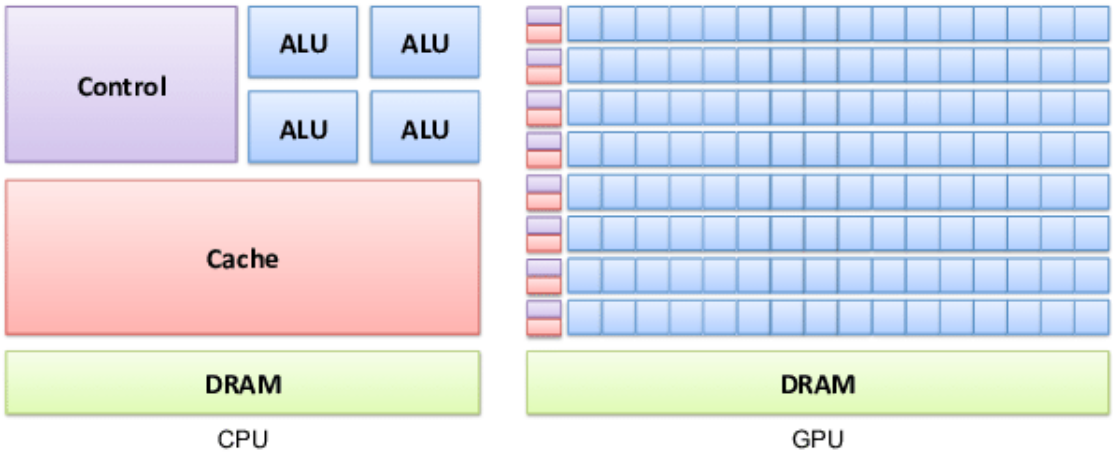
Nesse contexto, as GPUs, surgem como a principal evolução dos processadores matriciais e da arquitetura SIMD (AMIRI; SHAHBAHRAMSI, 2020). Devido a sua capacidade de processamento massivamente paralelo essa arquitetura mostrou-se extremamente eficiente para uma vasta gama de aplicações (OWENS *et al.*, 2008), sendo particularmente vantajosa para algoritmos que manipulam um alto volume de dados e que possuem um grau de paralelismo, tais como simulações científicas e de engenharia, análise e mineração de dados e algoritmos de aprendizado de máquina (OWENS *et al.*, 2008; KIRK; HWU, 2017; CHUNDURU, 2024).

3.2 UNIDADES DE PROCESSAMENTO GRÁFICO

A crescente demanda por poder de processamento em diversas áreas da ciência e engenharia tem impulsionado a busca por arquiteturas computacionais mais eficientes. Desta forma, as GPUs emergiram como uma alternativa às tradicionais CPUs para tarefas que exigem alto grau de paralelismo (ALERSTAM; SVENSSON; ANDERSSON-ENGELS, 2008; ANDERSON *et al.*, 2012; CASTILLO *et al.*, 2014). Originalmente concebidas para acelerar a renderização de gráficos em tempo real, as GPUs evoluíram para arquiteturas altamente paralelas e programáveis (OWENS *et al.*, 2008).

Essa capacidade de processamento massiva é possibilitada pela presença de um elevado número de núcleos de processamento. Conforme pode ser observado na Figura 3, uma GPU possui uma parcela maior da sua área física dedicada às unidades de computação (ALU do inglês *Arithmetic Logic Unit*) em comparação com uma CPU. As CPUs, por outro lado, possuem menos núcleos, porém mais avançados, com uma quantidade maior de memória *cache* e unidades de controle mais complexas e otimizadas para executar um único fluxo de instruções (OWENS *et al.*, 2008; SANDERS; KANDROT, 2010). Em contraste, as GPUs possuem centenas ou milhares de núcleos mais simples, projetados para executar uma mesma instrução em múltiplos fluxos de dados. Essa estrutura com um número maior de núcleos, torna as GPUs ideais para tarefas que podem ser divididas em operações idênticas e independentes (OWENS *et al.*, 2008; SANDERS; KANDROT, 2010).

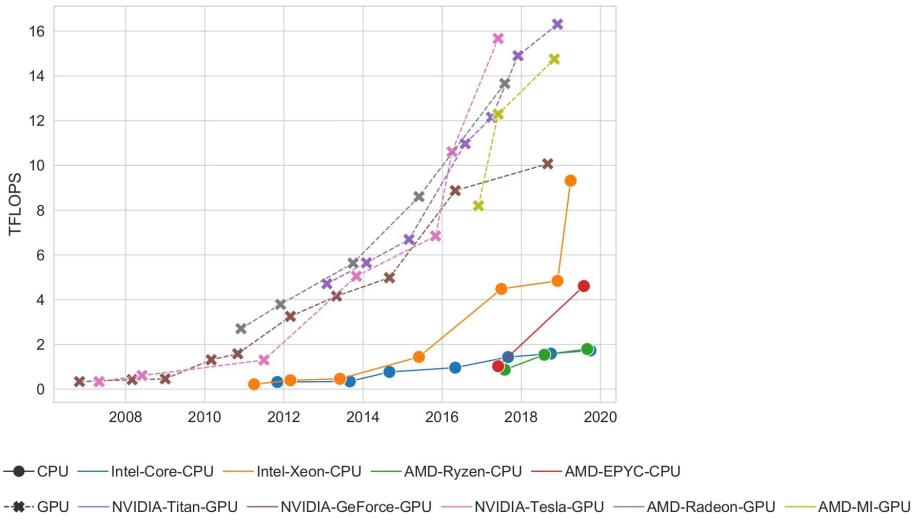
Figura 3 – Arquiteturas CPUs vs. GPUs



Fonte: Retirado de Lounis *et al.* (2015)

A Figura 4 presenta uma comparação de desempenho entre CPUs e GPUs. Como pode ser observado, a diferença de desempenho tem se intensificado ao longo dos anos, demonstrando uma crescente vantagem no uso de GPUs em processamentos intensivos. Além disso, as GPUs oferecem maior eficiência energética, reduzindo o consumo de energia, custos com refrigeração e a necessidade de espaço físico. Desta forma, essas tornaram-se uma boa alternativa, possibilitando a substituição de grandes *clusters* de CPUs por sistemas mais compactos e econômicos (OWENS *et al.*, 2008).

Figura 4 – Capacidade de processamento de CPUs vs. GPUs

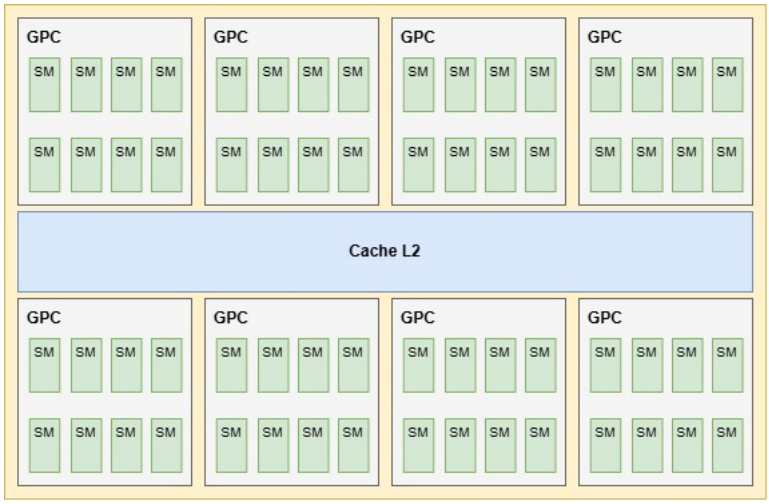


Fonte: Adaptado de Sun *et al.* (2019)

3.2.1 Arquitetura de uma GPU

De modo geral, a arquitetura de uma GPU é caracterizada pela presença de múltiplas unidades de processamento paralelo, cada uma composta por diversos núcleos de execução. No caso da NVIDIA, esses processadores são denominados processadores de fluxo (SMs do inglês, *Streaming Multiprocessor*). Cada um desses SM funciona como uma unidade de processamento independente, agrupando diversos núcleos de execução, chamados de CUDA *cores*. Na Figura 5 observa-se uma representação da arquitetura da NVIDIA Ada Lovelace. Nela, os diversos SMs são agrupados em *clusters* de processamento gráfico (GPC do inglês *Graphics Processing Cluster*), os quais se comunicam por meio de uma memória *cache* compartilhada.

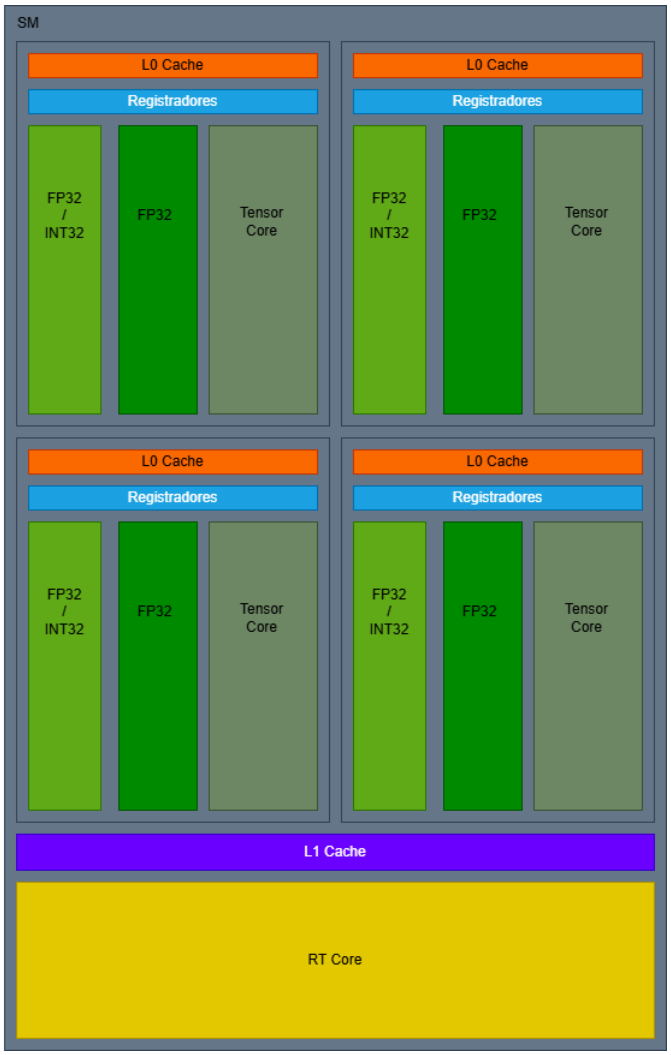
Figura 5 – Arquitetura Ada Lovelace



Fonte: Adaptado de NVIDIA (2022)

Os SMs (Figura 6) são as unidades responsáveis pelo processamento paralelo nas GPUs. Cada SM é projetado para gerenciar e executar múltiplas *threads*. No contexto de GPUs, uma *thread* pode ser entendida como a unidade mais básica de execução, ou seja, uma sequência de instruções que é processada por um dos núcleos (SANDERS; KANDROT, 2010; KIRK; HWU, 2017). Para lidar com esse processamento, os SMs integram diversos tipos de processadores, sendo que na arquitetura Ada Lovelace, destacam-se: os núcleos CUDA e os núcleos especializados. Os núcleos CUDA são as unidades de processamento fundamentais, sendo responsáveis pela execução de operações de ponto flutuante e inteiros. Os núcleos especializados incluem os núcleos RT, dedicados à aceleração de *Ray Tracing* gráfico, e os núcleos *Tensor*, otimizados para a execução de operações matriciais (NVIDIA, 2022).

Figura 6 – SM na Arquitetura Ada Lovelace

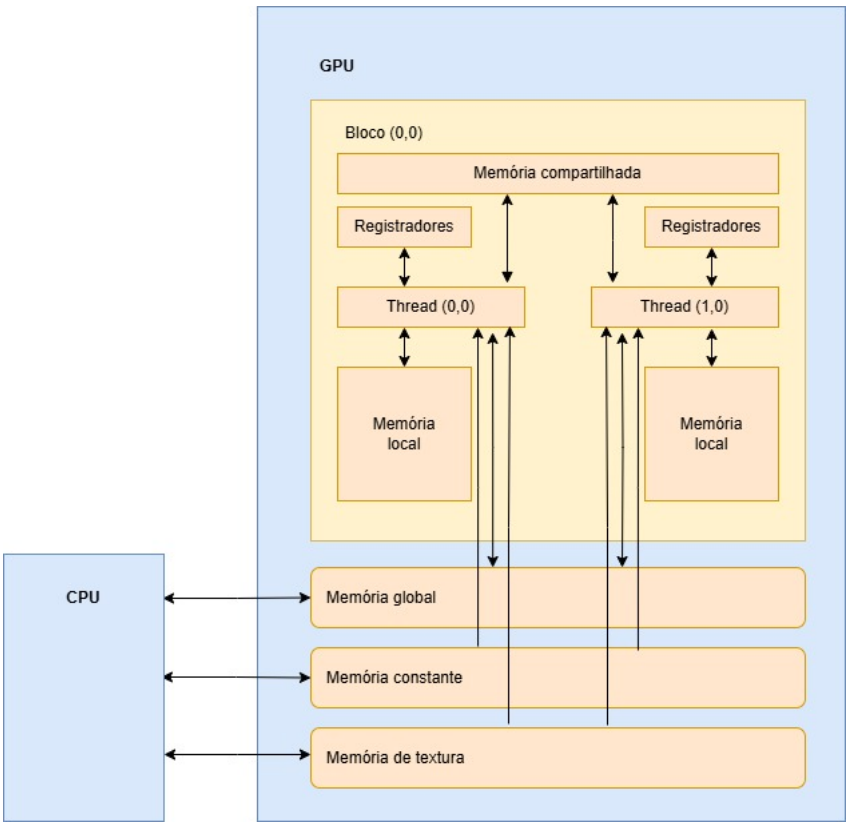


Fonte: Adaptado de NVIDIA (2022)

3.2.1.1 Hierarquia de memória

Em uma GPU, existem diferentes níveis de memória, cada um com diferentes escopos, tamanhos, tempos de acesso e características de uso (STALLINGS, 2017). A utilização correta desses recursos podem provocar um aumento significativo de desempenho (OWENS *et al.*, 2008). Como pode ser observado na Figura 7, os principais níveis de memória são os registradores, a memória local, memória compartilhada, memória global, memória constante e memória de textura.

Figura 7 – Hierarquia de memória de uma GPU



Fonte: Adaptado de Lai *et al.* (2019)

Os níveis de memória de uma GPU podem ser descritos como:

- Registradores: são as memórias mais rápidas, estando localizadas dentro de cada núcleo CUDA. Cada núcleo possui acesso exclusivo a seu próprio conjunto de registradores, que são utilizados para armazenar variáveis locais e dados temporários.
- Memória local: é acessível por uma única *thread*. Ela funciona como uma extensão dos registradores, sendo utilizada para armazenar variáveis que excedem o espaço disponível nos registradores.
- Memória compartilhada: encontra-se localizada dentro de cada SM. Essa é uma memória de baixa latência que pode ser acessada por todas as *threads* que executam em um mesmo

SM. Ela é fundamental para a cooperação entre as *threads*, funcionando como uma *cache* e reduzindo a latência de acesso à memória global.

- Memória global: é a memória principal da GPU, sendo caracterizada por uma capacidade de armazenamento maior e uma alta latência de acesso. Essa memória é compartilhada por todas as *threads* de todos os SMs, podendo também ser acessada pela CPU para a transferência de dados entre o *host*[2] e a GPU.
- Memória Constante: também se encontra localizada na memória global. A memória constante é otimizada para armazenar dados somente leitura, podendo ser acessada por todas as *threads* de todos os SMs.
- Memória de textura: também está localizada na memória global, sendo otimizadas para o acesso de dados em formatos 1D, 2D ou 3D.

3.3 MODELO DE PROGRAMAÇÃO CUDA

A consolidação das GPUs impulsionou a criação de ambientes de programação que simplificam a interação com o *hardware* e facilitem o seu uso em aplicações de propósito geral (KIRK; HWU, 2017). Nesse cenário, a NVIDIA desenvolveu a CUDA (*Compute Unified Device Architecture*), uma plataforma de software e um modelo de programação paralela para as suas GPUs (SANDERS; KANDROT, 2010). O objetivo da CUDA é reduzir a curva de aprendizado para a programação em GPUs, oferecendo extensões às linguagens C e C++ que permitem a criação de código executável nos múltiplos núcleos da GPU (SANDERS; KANDROT, 2010).

O paradigma da CUDA baseia-se em um modelo de computação heterogêneo, onde a CPU e a GPU colaboram para a execução de uma aplicação. Nele, o programador desenvolve um único código-fonte, incluindo as rotinas que serão executadas na CPU (o *host*), bem como as funções paralelas destinadas à GPU (o *device*). O compilador da NVIDIA, conhecido como NVCC[3], é responsável por compilar e gerar o código para cada uma das arquiteturas (SANDERS; KANDROT, 2010).

A execução de uma aplicação CUDA começa no *host*, que é responsável pela alocação da memória na GPU, bem como pela transferência de dados entre o *host* e o dispositivo (*device*). Para isso, a interface de programação da CUDA oferece um conjunto de funções específicas. A função `cudaMalloc()` é responsável por alocar um bloco de memória na memória global do *device*. Já a função `cudaMemcpy()` realiza a transferência de dados entre a memória do *host* e a do *device*, podendo operar em qualquer sentido. Por fim, a função `cudaFree()` é responsável por liberar a memória alocada no *device* (SANDERS; KANDROT, 2010). Na Algoritmo 1 tem-se exemplo de utilização das funções `cudaMalloc()` e `cudaMemcpy()` em um pseudo-código para soma de dois vetores.

[2] Dispositivo hospedeiro do programa, normalmente representado pela CPU
[3] Disponível em: <https://docs.nvidia.com/cuda/cuda-compiler-driver-nvcc/>

Algoritmo 1 – Exemplo de alocação de memória em CUDA

```
1    int a[N], b[N], c[N];
2    int *dev_a, *dev_b, *dev_c;
3
4    cudaMalloc((void**)&dev_a, N * sizeof(int));
5    cudaMalloc((void**)&dev_b, N * sizeof(int));
6    cudaMalloc((void**)&dev_c, N * sizeof(int));
7
8    cudaMemcpy(dev_a, a, N * sizeof(int), cudaMemcpyHostToDevice));
9    cudaMemcpy(dev_b, b, N * sizeof(int), cudaMemcpyHostToDevice));
```

Fonte: O Autor (2025)

Após a preparação dos dados, as funções paralelas são executadas na GPU através da chamada de funções especiais, denominadas *kernels*. Como pode ser observado no Algoritmo 2, um *kernel* é identificado pelo qualificador `__global__` antes da definição da função, e sua invocação é realizada pelo *host* através da seguinte sintaxe <<< ... >>>, que define a configuração de execução no *device*. Uma vez que o *kernel* foi concluído, o controle da execução retorna ao *host*, que pode então transferir os resultados de volta à sua memória principal para prosseguir com a execução do restante do programa (SANDERS; KANDROT, 2010).

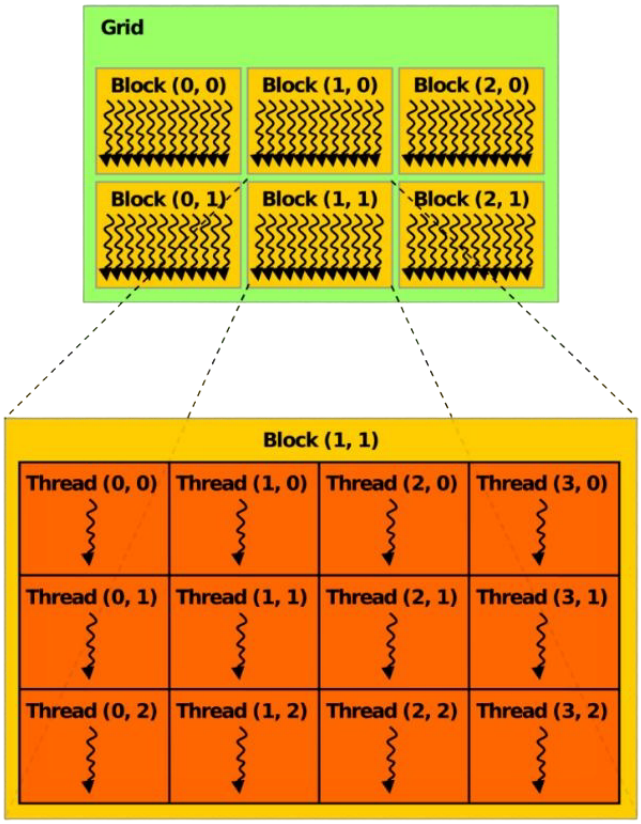
Algoritmo 2 – Exemplo de código paralelo em CUDA para somar vetores

```
1  __global__ void add( int *a, int *b, int *c ) {
2      int tid = blockIdx.x; // busca o dado que a thread deve processar
3      if (tid < N)
4      c[tid] = a[tid] + b[tid];
5  }
6
7  int main(void) {
8     add <<< NBLOCKS, NTHREADS >>>( dev_a, dev_b, dev_c );
9  }
```

Fonte: O Autor (2025)

Conforme pode ser observado na Figura 8, a plataforma CUDA organiza as *threads* de um *kernel* em uma hierarquia de dois níveis: blocos e *grid*. No nível inferior, as *threads* são agrupadas em blocos (*thread blocks*), os quais podem cooperar, sincronizando operações com barreiras e compartilhando dados através da memória compartilhada do SM. Os blocos são executados de forma independente e em qualquer ordem, permitindo assim a execução do código em GPUs com diferentes capacidades de processamento (SANDERS; KANDROT, 2010). Já no nível superior, os blocos são organizados em um *grid*, que engloba todas as *threads* executadas pelo *kernel*. No Algoritmo 2 tem-se um exemplo da chamada de um *kernel* (linha 8), onde `NBLOCKS` representa o número de blocos de *threads* que formarão o *grid*, e `NTHREADS` especifica quantas *threads* farão parte de cada bloco.

Figura 8 – Hierarquia de blocos e *grids*



Fonte: NVIDIA Corporation (2025)

Para que cada *thread* processe um dado diferente, a CUDA fornece variáveis embutidas para que cada *thread* identifique a posição do dado a ser processado. As principais são `threadIdx` (índice da *thread* no bloco) e `blockIdx` (índice do bloco no *grid*). Com elas e com a variável `blockDim` (tamanho do bloco), é possível calcular um índice global único para cada *thread*, que corresponde ao dado que será processado pela mesma (Algoritmo 3). Essa estrutura permite que um mesmo *kernel* seja executado por milhares de *threads* simultaneamente, cada uma operando sobre um dado distinto.

Algoritmo 3 – Exemplo de kernel com identificação de uma thread

```
1 __global__ void add( int *a, int *b, int *c ) {
2     int tid = threadIdx.x + blockIdx.x * blockDim.x;
3     if (tid < N)
4         c[tid] = a[tid] + b[tid];
5 }
```

Fonte: Adaptado de (SANDERS; KANDROT, 2010)

Além do qualificador __global__ que define funções do tipo *kernel*, a CUDA também disponibiliza outros dois qualificadores: __host__ que define funções que só poderão ser executadas pelo *host*, e __device__ que define funções que executarão em GPU, mas só poderão ser chamadas por funções __global__ ou outras funções do tipo __device__.

Ademais, a CUDA também disponibiliza qualificadores que definem em qual memória as variáveis serão alocadas. Essas memórias apresentam diferentes velocidades e capacidades, de modo que a escolha adequada desses recursos constitui um fator importante para o desempenho final dos programas. O qualificador __device__ é utilizado para alocar uma variável na memória global da GPU, sendo que essa variável é acessível a todas as *threads* do *kernel*. O qualificador __constant__ é utilizado para alocar uma variável na memória constante, que é um espaço de memória de apenas leitura para a GPU, mas que pode ser modificada pelo *host*. Uma variável alocada na memória constante é acessível a todas as *threads* do *kernel*. Por fim, o qualificador __shared__ é utilizado para alocar uma variável na memória compartilhada de um SM. Neste caso, a variável só pode ser declarada no escopo de uma função *kernel*. Essa memória é compartilhada e acessível por todas as *threads* de um mesmo bloco (SANDERS; KANDROT, 2010; COOK, 2013). No Algoritmo 4 tem-se exemplos de utilização desses qualificadores.

Algoritmo 4 – Exemplo de alocação de memória para variáveis

```
1 __constant__ int const_data[256];
2 __device__ int global_data[256];
3
4 __global__ void exemplo_de_kernel() {
5     __shared__ int shared_data[256];
6 }
```

Fonte: O Autor (2025)

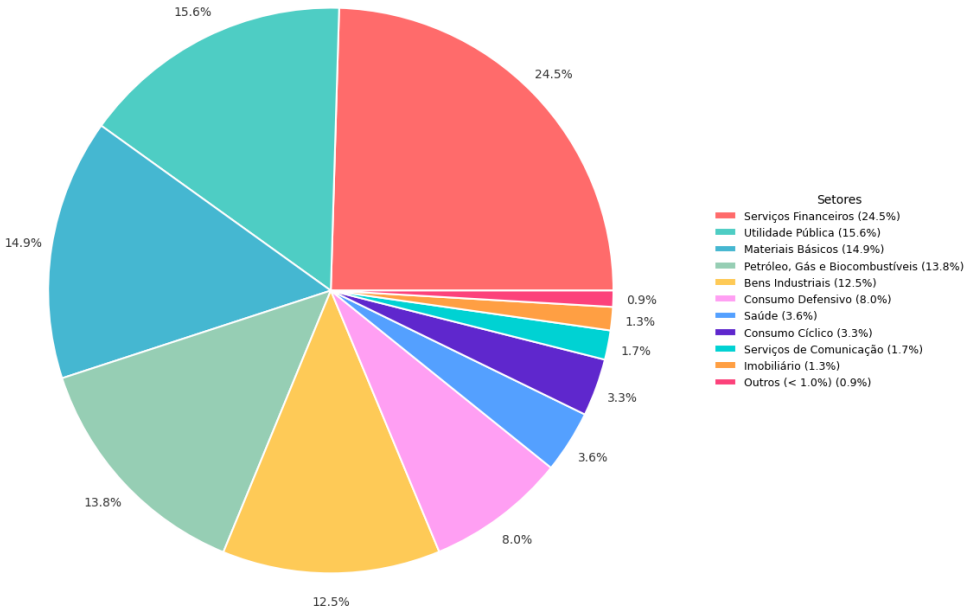
4 IMPLEMENTAÇÃO DESENVOLVIDA

Este capítulo apresenta a implementação desenvolvida. Inicialmente, é apresentado como foram definidas as ações utilizadas para os testes. Em seguida, é descrito o processo de coleta dos dados históricos e o cálculo das constantes do GBM. Na sequência, é apresentada a versão sequencial do método de SMC e, posteriormente, a implementação paralela em GPU CUDA.

4.1 DEFINIÇÃO DAS AÇÕES

Para a seleção das ações foi considerada a composição do índice Bovespa, que reúne 87 empresas divididas em 11 setores. A participação de cada setor no Ibovespa pode ser observada na Figura 9. Inicialmente, foram selecionadas as maiores empresas dentro dos cinco principais setores: Itaúsa S.A. - ITSA4 (Serviços Financeiros); Eletrobras - ELET3 (Utilidade Pública); Vale S.A. - VALE3 (Materiais Básicos); Petróleo Brasileiro S.A. - PETR4 (Petróleo, Gás e Biocombustíveis) e WEG S.A. - WEGE3 (Bens Industriais). Além disso, foi realizada uma avaliação do próprio índice Ibovespa, uma vez que sua composição diversificada permite captar dinâmicas de mercado mais amplas. Isso se deve ao fato de que cada setor exibe tendências de volatilidade distintas, enquanto o índice, ao agregar todos eles, tende a apresentar uma volatilidade geral menor.

Figura 9 – Composição do Ibovespa (maio de 2025)

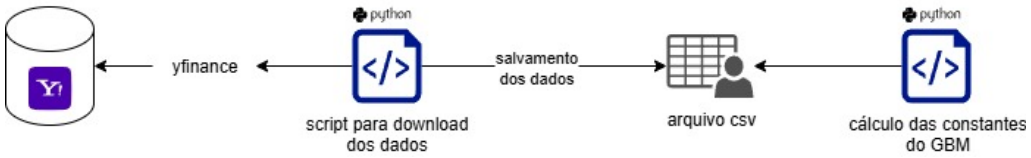


Fonte: O Autor (2025)

4.2 COLETA DOS DADOS HISTÓRICOS

Os históricos dos valores das ações foram coletados por meio da biblioteca yfinance[1], implementada em Python, que fornece acesso aos preços históricos de ações da plataforma *Yahoo finance*[2]. Neste caso, foi utilizado o preço diário ajustado, ou seja, o preço já adicionando fatores externos que podem gerar renda, como dividendos e JCP. Esses dados foram pré-processados e armazenados em um arquivo no formato CSV (*Comma-Separated Values*), de modo a facilitar sua leitura por um outro programa desenvolvido em Python, que é responsável pelo cálculo das constantes necessárias ao modelo de GBM (Figura 10).

Figura 10 – Visualização dos passos para se obter as constantes do GBM



Fonte: O Autor (2025)

O método GBM utiliza duas constantes que são calculadas a partir dos dados históricos de um ativo: o termo μ, que representa o retorno histórico anualizado, e o termo σ, que representa a volatilidade anualizada do ativo. O parâmetro μ é obtido pela média dos retornos logarítmicos diários multiplicada pelo número de dias de negociação em um ano (geralmente 252). Já a volatilidade σ é calculada como o desvio padrão dos retornos logarítmicos diários (anualizado pela multiplicação pela raiz quadrada do número de dias de negociação).

O Algoritmo 5 apresenta o cálculo das constantes μ e σ. Na linha 1 calculam-se os retornos logarítmicos diários utilizando a função log da biblioteca *NumPy*. Como entrada, são utilizados os preços dos ativos armazenados na coluna df['price'] de um *DataFrame* da biblioteca *pandas*. Os valores resultantes são então armazenados na coluna df['returns'] do mesmo *DataFrame*. A linha 2 determina o retorno histórico anualizado μ, que é calculado obtendo-se a média dos retornos logarítmicos (df['returns'].mean()) e multiplicando-a pelo número de dias de negociação. Na linha 3 tem-se o cálculo da volatilidade histórica anualizada σ, que é obtido calculando-se o desvio padrão dos retornos logarítmicos (df['returns'].std()) e, em seguida, anualizando o resultado através da multiplicação pela raiz quadrada do número de dias de negociação no ano (np.sqrt(252)). Por fim, na linha 4 tem-se o calcula do termo *drift* ajustado, que corresponde ao componente $\left(\mu - \frac{\sigma^2}{2}\right)$ da equação de simulação. Este ajuste é armazenado na variável drift e é o valor efetivamente utilizado no cálculo para simular a tendência do ativo.

[1] Disponível em: <https://ranaroussi.github.io/yfinance/>
[2] Disponível em: <https://finance.yahoo.com/>

Algoritmo 5 – Cálculo das constantes GBM

```
1     df['returns'] = np.log(df['price']).diff()
2     historical_return = df['returns'].mean() * 252
3     historical_volatility = df['returns'].std() * np.sqrt(252)
4     drift = historical_return - 0.5 * historical_volatility**2
```

Fonte: O Autor (2025)

4.3 IMPLEMENTAÇÃO SEQUENCIAL

A implementação sequencial foi desenvolvida na linguagem de programação C, padrão ANSI. Nesta, cada trajetória de preço é simulada de forma sequencial, onde uma simulação só se inicia após a conclusão da anterior. A estrutura do código foi criada considerando duas funções: a função generate_normal_random(), responsável pela geração de números aleatórios com um distribuição normal; e a função monte_carlo_gbm() que é responsável pela execução da SMC.

O Algoritmo 6 apresentada o código da função generate_normal_random(). Essa implementa a transformada de Box-Muller (SCOTT, 2011; SHUKUR *et al.*, 2024), que converte dois números aleatórios de distribuição uniforme em um número aleatório com distribuição normal padrão.

Algoritmo 6 – Geração de números aleatórios com distribuição normal

```
1  double generate_normal_random() {
2      double u1 = ((double)rand() / (RAND_MAX)) * 0.99999 + 0.00001;
3      double u2 = ((double)rand() / (RAND_MAX)) * 0.99999 + 0.00001;
4
5      double z = sqrt(-2.0 * log(u1)) * cos(2.0 * M_PI * u2);
6      return z;
7  }
```

Fonte: O Autor (2025)

O Algoritmo 7 apresenta o código da função principal, monte_carlo_gbm(), responsável pela execução da SMC. Essa utiliza dois laços, onde o laço externo (linha 14) é quem realiza a execução das simulações para cada cenário gerado pelo método de Monte Carlo. O laço interno (linha 15) percorre cada passo de tempo, calculando o preço da ação em cada instante. O valor da ação é calculado através da equação do GBM (Equação 2.4), conforme pode ser observado na linha 17. A variável aleatória (epsilon) introduz o componente estocástico a cada passo de tempo da simulação, sendo gerada a partir da função generate_normal_random() (linha 16).

Algoritmo 7 – Execução da SMC

```
1  double** monte_carlo_gbm(double S0, double mu, double sigma, double T,
2      double dt, int num_simulations) {
3      int num_steps = (int)(T / dt);
4
5      double** paths = (double**)malloc((num_steps + 1) * sizeof(double*));
6      for (int i = 0; i <= num_steps; i++) {
7          paths[i] = (double*)malloc(num_simulations * sizeof(double));
8      }
9
10     for (int i = 0; i < num_simulations; i++) {
11         paths[0][i] = S0;
12     }
13
14     for (int i = 0; i < num_simulations; i++) {
15         for (int t = 1; t <= num_steps; t++) {
16             double epsilon = generate_normal_random();
17             paths[t][i] = paths[t - 1][i] * exp((mu - 0.5 * sigma * sigma)
18             * dt + sigma * sqrt(dt) * epsilon);
19         }
20     }
21     return paths;
22 }
```

Fonte: O Autor (2025)

4.4 IMPLEMENTAÇÃO PARALELA EM GPU

Para o desenvolvimento da implementação paralela foi utilizada a plataforma CUDA versão 12.6. Na versão paralelizada para GPU, cada simulação é uma *thread* independente, possibilitando a execução simultânea de milhares de trajetórias. O implementação pode ser dividida em 3 partes principais a transferêncais dos dados para a memória da GPU, execução das simulações e coleta dos resultados.

A etapa de transferência dos dados para a memória da GPU consiste em alocar a memória necessária para os resultados da simulação, buscar os preços iniciais e copiar esses dados para a memória da GPU. O Algoritmo 8 apresenta esse processo, em que, na linha 1, é calculado, na variável paths_size, o tamanho necessário para armazenar os resultados das simulações, sendo num_steps o número de dias de simulação e num_simulations o total de simulações. Em seguida, na linha 5, é alocada na memória da GPU a variável d_paths, que armazenará os resultados das simulações. Depois, na linha 7, é alocado o vetor h_initial_prices , que é inicializado com $S0$, o valor inicial da ação a ser simulada (linha $S0$). Por fim, na linha 14, a função cudaMemcpy é utilizada para copiar esses valores para a variável d_paths, alocada na memória da GPU.

Algoritmo 8 – Transferência dos dados para a memória da GPU

```
1     size_t paths_size = (long long)(num_steps + 1) *
2     num_simulations * sizeof(double);
3
4     double *d_paths;
5     cudaMalloc((void **)&d_paths, paths_size);
6
7     double *h_initial_prices = (double *)malloc(
8         num_simulations * sizeof(double)
9     );
10    for (int i = 0; i < num_simulations; i++)
11    {
12        h_initial_prices[i] = S0;
13    }
14    cudaMemcpy(
15        d_paths, h_initial_prices,
16        num_simulations * sizeof(double), cudaMemcpyHostToDevice
17    );
```

Fonte: O Autor (2025)

Após a preparação dos dados, tem-se o início da execução das simulações da SMC na GPU. O Algoritmo 9 apresenta o código do *kernel* monte_carlo_full_path_kernel , responsável pela execução das simulações na GPU. Nele, cada *thread* é responsável por calcular uma trajetória completa, reduzindo a necessidade de sincronização entre as *threads*. Na linha 7 cada *thread* recebe um identificador único (variável i), que a associa a uma simulação específica. Após, cada *thread* inicializa um gerador de números aleatórios próprio através da função curand_init (linha 11). Isso possibilita que cada *thread* tenha uma sequência de números aleatórios única e independente. Em seguida, na linha 15, é executado o laço que, para cada instante de tempo, calcula o preço da ação por meio da equação do GBM, armazenando o resultado na variável current_price . Nessa etapa, utiliza-se a função curand_normal_double para gerar números aleatórios com distribuição normal (linha 16). Os resultados são então armazenados no vetor linearizado paths, localizado na memória global da GPU (linha 20).

Algoritmo 9 – Kernel CUDA para a execução da SMC

```
1  __global__
2  void monte_carlo_full_path_kernel(
3      double* paths, int num_simulations, int num_steps,
4      double mu, double sigma, double dt,
5      double sqrt_dt, unsigned long long seed
6  ) {
7      int i = blockIdx.x * blockDim.x + threadIdx.x;
8
9      if (i < num_simulations) {
10         curandState local_state;
```

```
11          curand_init(seed, i, 0, &local_state);
12
13          double current_price = paths[i];
14
15          for (int t = 1; t <= num_steps; t++) {
16              double epsilon = curand_normal_double(&local_state);
17              current_price = current_price * exp(
18                  (mu - 0.5 * sigma * sigma) * dt + sigma * sqrt_dt * epsilon
19              );
20              paths[(long long)t * num_simulations + i] = current_price;
21          }
22      }
23  }
```

Fonte: O Autor (2025)

O Algoritmo 10 apresenta a chamada do *kernel* responsável por realizar as simulações. Esse recebe como parâmetros: o vetor d_paths, previamente alocado e inicializado com os valores iniciais e transferido para a memória da GPU; o número de simulações (num_simulations); o número de passos (ou dias) (num_steps); as constantes μ (mu) e σ (sigma); as variáveis dt e sqrt_dt que representam, respectivamente, o incremento do passo temporal (neste caso, $1/252$, correspondente a um dia útil em um ano) e sua raiz quadrada; e por fim, a variável seed, obtida via função time da linguagem C, para inicialização do gerador de números aleatórios no *kernel*. A execução do *kernel* é configurada a partir do número de *threads* por bloco, definido na variável threads_per_block (com valor 256), e do número de blocos por *grid*, armazenado na variável blocks_per_grid, calculado pela expressão $(\text{num\_simulations} + \text{threads\_per\_block} - 1)/\text{threads\_per\_block}$. Esses valores foram determinados empiricamente por meio de testes iterativos, selecionando-se a combinação que resultou na melhor otimização do tempo de processamento.

Algoritmo 10 – Chamada do *kernel* para a execução das simulações

```
1       monte_carlo_full_path_kernel<<<blocks_per_grid, threads_per_block>>>(
2           d_paths, num_simulations, num_steps, mu, sigma, dt, sqrt_dt, seed
3       );
```

Fonte: O Autor (2025)

O Algoritmo 11 apresenta a função calculate_and_print_confidence_band , responsável pela exibição dos resultados da simulação. Esta recebe como parâmetros o dia específico a ser analisado (day), uma referência ao vetor d_paths contendo todos os resultados das simulações, o número de simulações (num_simulations) e o número de passos (variável num_steps). Na linha 7, aloca-se um vetor na memória do *host* (h_prices_at_day) destinado a armazenar os resultados das simulações correspondentes ao dia especificado. A transferência dos valores para a memória do *host* é realizada por meio da função cudaMemcpy, com a diretiva cudaMemcpyDeviceToHost. Em

seguida, na linha 14, esses valores são ordenados utilizando a função qsort da linguagem C, o que possibilita o cálculo das faixas de confiança. As variáveis lower_index e upper_index (definidas nas linhas 21 e 22) são utilizadas para delimitar a faixa desejada (no exemplo faixa de confiança de 75%).

Algoritmo 11 – Função para calcular as faixas de confiança

```
1  void calculate_and_print_confidence_band(
2      int day,
3      const double* d_paths,
4      int num_simulations, int num_steps
5  ) {
6      size_t daily_prices_size = num_simulations * sizeof(double);
7      double* h_prices_at_day = (double*)malloc(daily_prices_size);
8      const double* d_source_ptr = d_paths +
9          (long long)day * num_simulations;
10     cudaMemcpy(
11         h_prices_at_day,
12         d_source_ptr, daily_prices_size, cudaMemcpyDeviceToHost
13     );
14     qsort(
15         h_prices_at_day,
16         num_simulations,
17         sizeof(double),
18         compare_doubles
19     );
20
21     int lower_index = (int)(num_simulations * 0.125);
22     int upper_index = (int)(num_simulations * 0.875);
23
24     double lower_bound = h_prices_at_day[lower_index];
25     double upper_bound = h_prices_at_day[upper_index];
26
27     printf("\%.4f;\%.4f;", lower_bound, upper_bound);
28 }
```

Fonte: O Autor (2025)

5 TESTES E RESULTADOS OBTIDOS

Neste capítulo são apresentados os principais resultados obtidos. Em um primeiro momento, é realizada uma análise do desempenho do modelo, avaliando a qualidade das previsões por meio da comparação entre os valores previstos e os valores reais dos ativos. Por fim, é realizada uma avaliação do desempenho da implementação paralela em GPU.

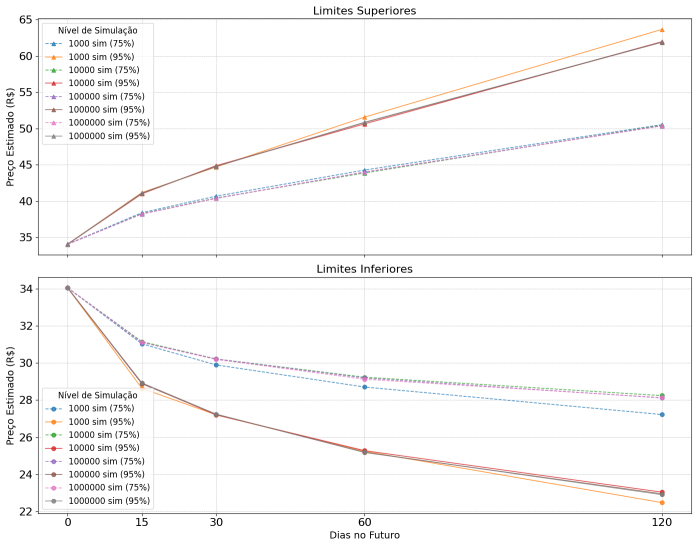
5.1 AVALIAÇÃO DO MODELO

A avaliação do modelo foi realizada com base em um período de 120 dias já transcorrido, de janeiro a junho de 2025. Os valores gerados pelo modelo foram comparados com os preços ajustados de fechamento dos ativos. Uma vez que a SMC gera múltiplos resultados, um para cada simulação, a análise realizada foi feita por meio da construção de um intervalo de confiança, conforme descrito em Brătian *et al.* (2022). Neste caso, foram estipuladas duas faixas de confiança 95% e 75% do total de simulações feitas. Por exemplo, 75% dos preços gerados em todas as simulações situam-se dentro deste intervalo de preços.

5.1.1 Análise do Número de Simulações

Conforme Jabbour e Liu (2005), o aumento no número de simulações tende a melhorar a precisão dos resultados. Na Figura 11 é apresentado uma comparação entre as trajetórias das ações da empresa Petrobras (PETR4), utilizando 1.000, 10.000, 100.000 e 1.000.000.

Figura 11 – Avaliação do Número de Simulações - PETR4



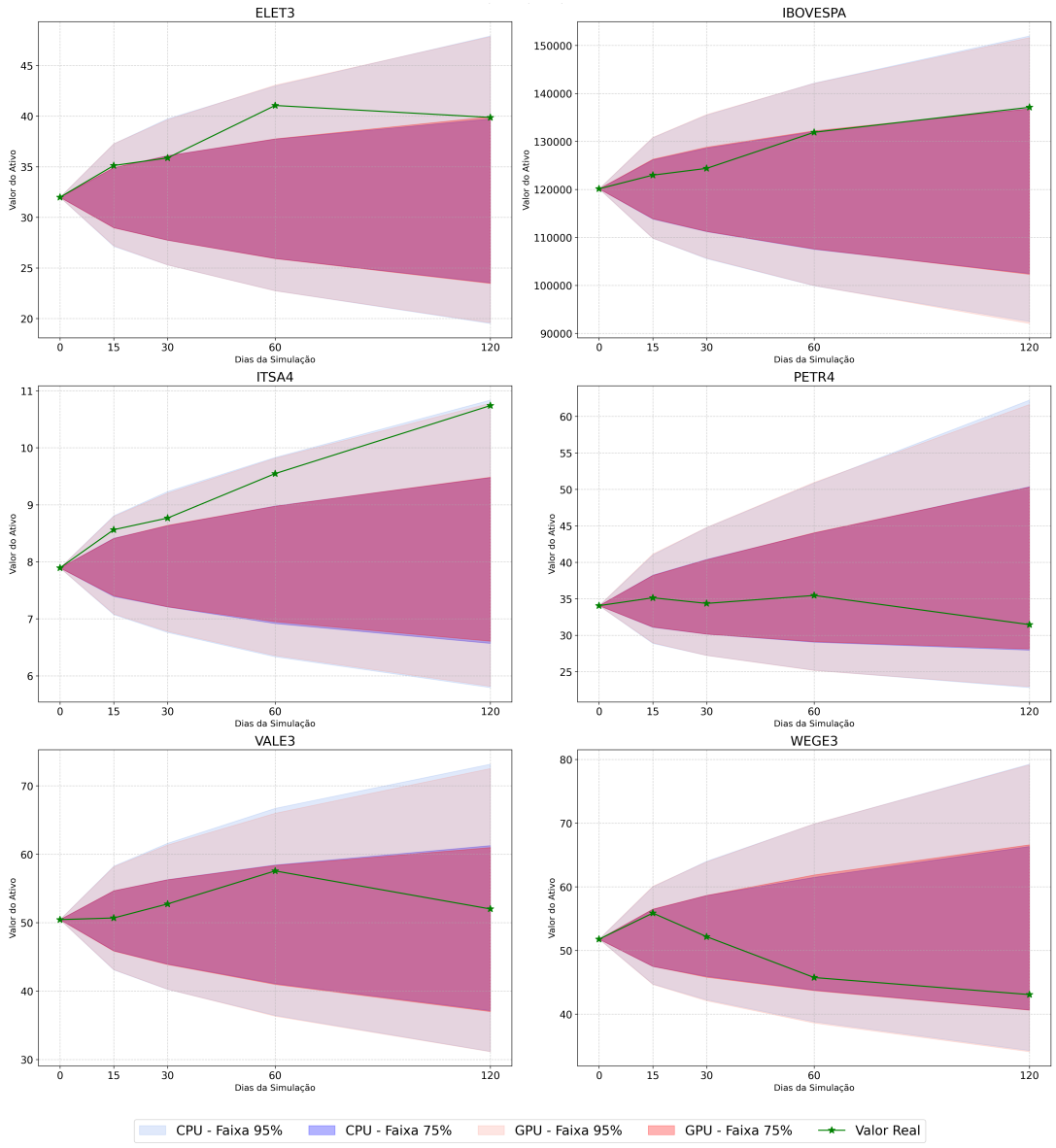
Fonte: O Autor (2025)

Observa-se que, a partir de 100.000 simulações, as variações reduzem-se a décimos de centavo, indicando que esse número é suficiente para garantir a precisão da análise. Assim, adotou-se esse valor para os demais testes.

5.1.2 Avaliação das Predições

A Figura 12 mostra as previsões obtidas para cada ativo, apresentando a variação do preço em função do número de dias. A faixa em rosa escuro representa o intervalo de confiança de 75%, ou seja, a região onde se encontram os resultados de 75% das simulações. Já a faixa rosa claro corresponde ao intervalo de confiança de 95%. A linha verde representa o valor real da ação naquele período.

Figura 12 – Resultados por ação



Fonte: O Autor (2025)

Observa-se que em todos os cenários, o valor real dos ativos esteve contido na faixa de confiança de 95%. Para a faixa de confiança de 75%, verificou-se que os ativos ELET3 e ITSA4, caracterizados por elevada volatilidade no período analisado, ficaram fora da faixa prevista. Observa-se ainda que, à medida que o prazo de predição avança, ocorre um "efeito de cone", em que os limites das faixas tornam-se mais distantes entre si, refletindo a maior dispersão das predições inerente ao modelo GBM.

5.2 AVALIAÇÃO DA IMPLEMENTAÇÃO PARALELA EM GPU

Para a realização dos testes da implementação paralela em GPU foi utilizada uma *NVIDIA RTX 4060*, que é baseada na arquitetura Ada Lovelace. As suas especificações incluem uma memória DRAM de 8 *Gigabyte*s (GB) e um total de 3072 núcleos CUDA, 96 núcleos *Tensor* e 24 núcleos *RT* [1]. Esses núcleos são organizados em 24 SMs, sendo que cada um possui um cache L1 local de 128 KB e um cache L2 compartilhado com capacidade de 24 MBs. Para cálculos com ponto flutuante, a placa possui um desempenho estimado de 15,11 TFLOP/s.

As configurações do computador hospedeiro incluem um processador *Intel Core i5-10400F*, com 6 núcleos físicos e 12 *threads*, que possuem uma frequência base de 2,9 GHz e uma frequência máxima de 4,3 GHz (*max turbo boost*) [2] , 16 GBs de memória RAM DDR4, operando a 2666 MHz. Para armazenamento, esse computador possui um disco rígido SSD NVMe (*Non-Volatile Memory Express*) KC3000 com 1 TB. O sistema operacional utilizado será o Ubuntu por meio da ferramenta WSL (*Windows Subsystem for Linux*), versão 24.04.1, com o *kernel 5.15.146.1-microsoft-standard-WSL2*.

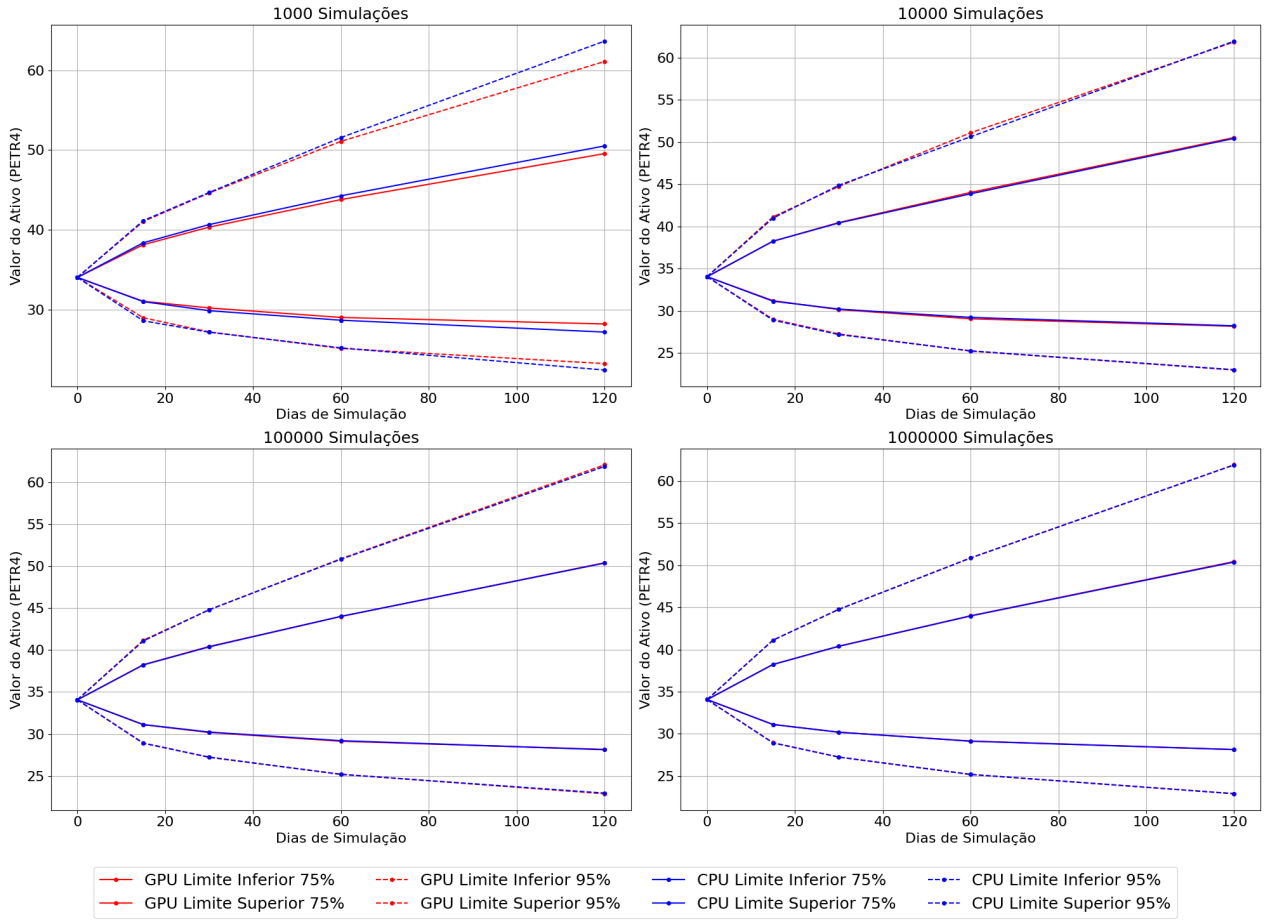
5.2.1 Comparação dos Resultados Numéricos

Na Figura 13, são apresentados gráficos que ilustram a média dos resultados obtidos para o ativo PETR4, tanto na execução sequencial em CPU (azul) quanto em GPU (vermelho), em função do número de simulações. No caso de 1.000 simulações, observa-se uma variação maior nos resultados, atribuída não a influência de erros introduzidos pela paralelização, mas à baixa precisão decorrente do reduzido número de simulações. Para os demais casos, nota-se a convergência dos resultados em ambas as abordagens, o que evidencia que o emprego de paralelismo não compromete a qualidade das soluções obtidas.

[1] Disponível em: <https://www.nvidia.com/pt-br/geforce/graphics-cards/40-series/rtx-4060-4060ti/>

[2] Disponível em: <https://www.intel.com.br/content/www/br/pt/products/sku/199278/intel-core-i510400f-processor-12m-cache-up-to-4-30-ghz/specifications.html>

Figura 13 – Comparação implementações sequencial e paralela em GPU para o ativo PETR4



Fonte: O Autor (2025)

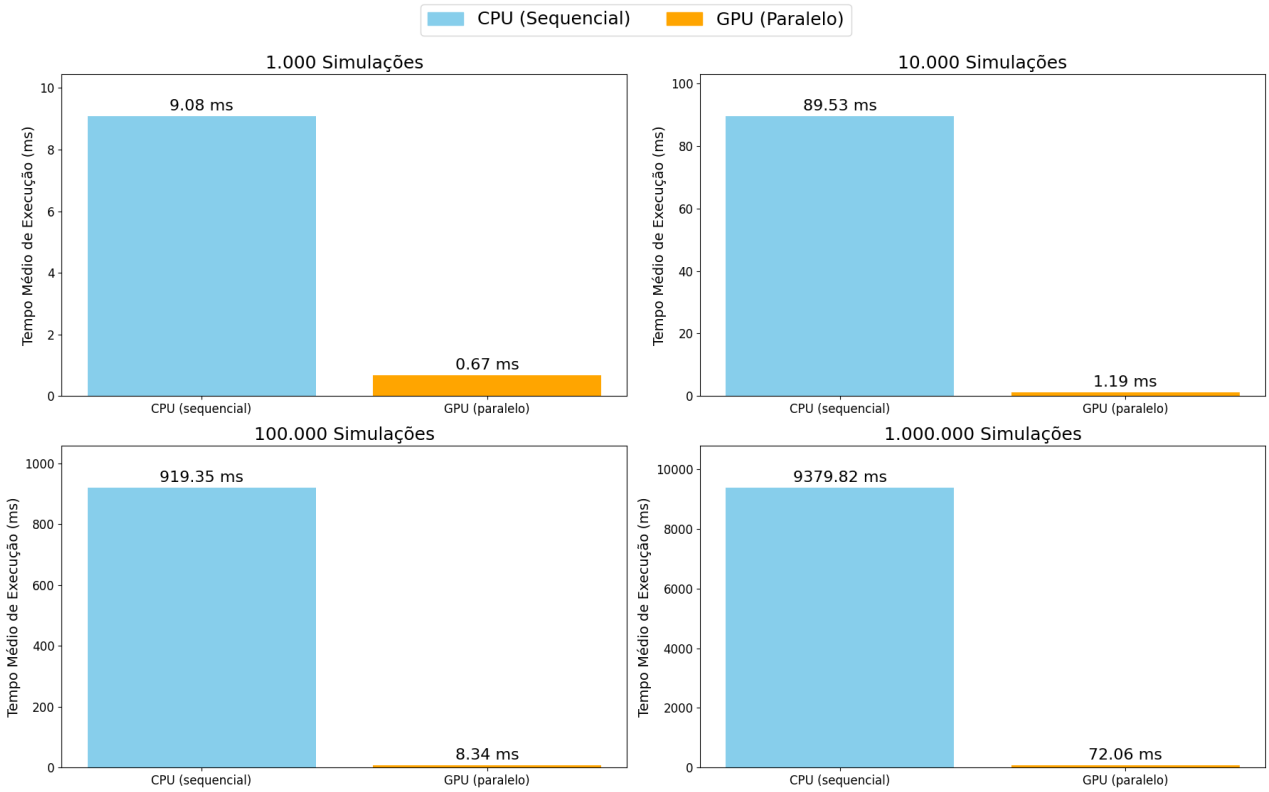
5.3 AVALIAÇÃO DE DESEMPENHO

A avaliação das implementações foi realizada após a inicialização do sistema. Foram conduzidos testes com 1.000, 10.000, 100.000 e $1.000.000$ simulações. Para cada teste, foram realizadas 3 execuções, a partir das quais foram calculados a média dos tempos de execução e o desvio padrão. A partir das médias dos tempos de execução, foi calculado o *speedup*. Esse consiste em uma métrica que indica o ganho de desempenho de uma implementação paralela em relação à sequencial. O cálculo do *speedup* é feito através da Equação 5.1, que é a razão entre o tempo de processamento sequencial ($T_{\text{sequencial}}$) na CPU e o tempo de processamento paralelo na GPU (T_{paralelo}).

$$S = \frac{T_{\text{sequencial}}}{T_{\text{paralelo}}} \tag{5.1}$$

A Figura 14 mostra a comparação entre os tempos médios de execução das implementações sequencial e paralela em GPU. Observa-se um ganho significativo de desempenho na execução paralela em relação à sequencial. A vantagem de desempenho se torna mais expressiva com o aumento do número de simulações, visto que a natureza da SMC permite a execução independente de cada simulação por uma *thread*, o que favorece um uso mais eficiente dos núcleos disponíveis.

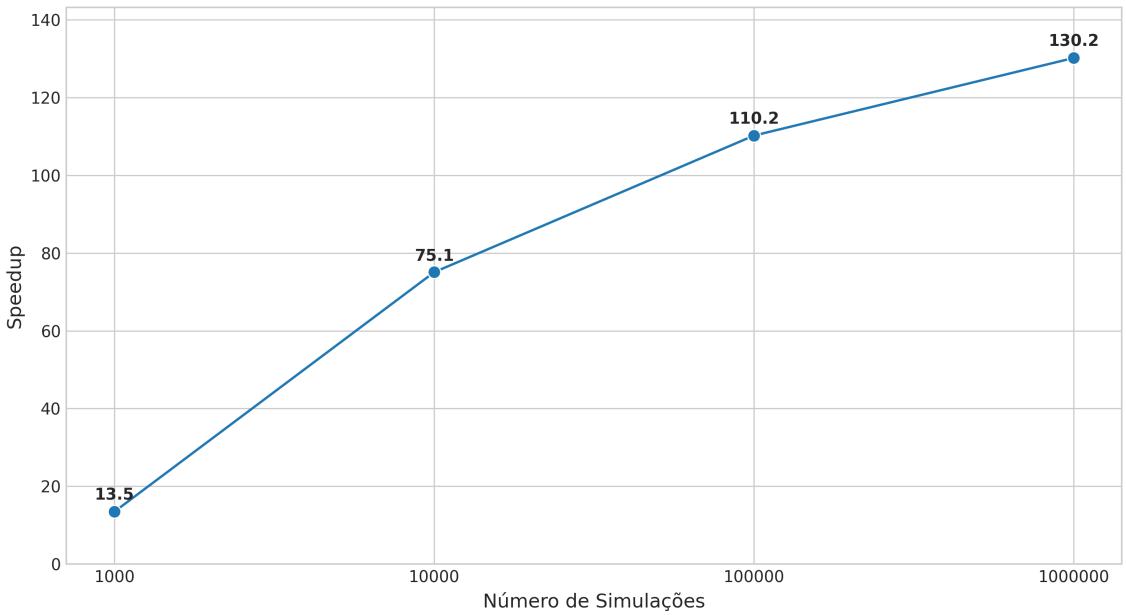
Figura 14 – Tempo de execução da implementação sequencial e a paralela em GPU



Fonte: O Autor (2025)

A Figura 15 apresenta um gráfico do *speedup* em função do número de simulações (em escala logarítmica). Observa-se que o ganho de desempenho aumenta significativamente com o número de simulações. O *speedup* apresentou um crescimento, passando de $13{,}5\times$ com 1.000 simulações para $130{,}2\times$ com $1.000.000$ simulações. Isso é esperado visto que o custo computacional da transferência de dados se torna proporcionalmente menor em relação ao processamento total, podendo ser diluído pelo número elevado de simulações. Por outro lado, observa-se uma desaceleração no ganho de *speedup* conforme o número de simulações aumenta. Isso ocorre porque, ao se atingir a ocupação total dos núcleos da GPU, as *threads* excedentes precisam aguardar para serem executadas.

Figura 15 – *Speedup* em função do número de simulações



Fonte: O Autor (2025)

6 CONSIDERAÇÕES FINAIS

Este trabalho apresentou como principal objetivo o desenvolvimento e a avaliação de uma versão paralela em GPU da SMC para a previsão do preço de ações, utilizando o modelo do GBM. Para desenvolvimento da versão paralela em GPU foi utilizada a plataforma CUDA, pois é o modelo de programação disponibilizado pela NVIDIA, empresa que atualmente lidera o mercado de placas de vídeo e também é a fabricante da GPU que foi utilizada como base no desenvolvimento deste trabalho.

Para a avaliação do modelo, foram selecionadas as ações de maior relevância de cinco dos principais setores do Ibovespa (ITSA4, ELET3, VALE3, PETR4 e WEGE3), além do próprio índice. Essa escolha permitiu analisar o comportamento do modelo em ativos com diferentes características de volatilidade. Os testes de convergência indicaram que, a partir de 100.000 simulações, os resultados atingem um ponto de equilíbrio entre precisão e custo computacional. Este valor, adotado para as análises subsequentes, é superior ao encontrado na literatura que frequentemente utilizam apenas 1.000 simulações para suas análises.

A implementação da SMC com um modelo GBM apresentou resultados satisfatórios, com os valores reais dos ativos permanecendo dentro do intervalo de confiança de 95% em todos os cenários. No entanto, para o intervalo mais restrito de 75%, observou-se que ativos com maior volatilidade no período analisado de 6 meses, como ELET3 e ITSA4, ficaram fora da faixa prevista, evidenciando uma limitação do modelo em cenários de maior instabilidade. Adicionalmente, confirmou-se que a incerteza da previsão, representada pela amplitude dos intervalos de confiança, aumenta para previsões mais longas, uma característica intrínseca ao modelo.

A implementação paralela em GPU mostrou um bom desempenho, apresentando uma redução significativa no tempo de execução e mantendo a qualidade numérica dos resultados. A natureza paralelizável da SMC mostrou-se ideal para a arquitetura das GPUs, permitindo que cada trajetória de preço seja calculada de forma independente. O ganho de desempenho (*speedup*) mostrou-se mais relevante com o aumento do número de simulações, atingindo um valor de $130,2$x para $1.000.000$ de simulações. Isso ocorre, pois o custo computacional associado à transferência de dados entre o *host* e o *device* torna-se proporcionalmente menor em relação ao processamento total. No entanto, o ganho de *speedup* apresenta uma desaceleração à medida que o número de simulações aumenta, devido à ocupação total dos núcleos disponíveis na GPU, o que leva as *threads* excedentes a aguardarem sua execução.

6.1 TRABALHOS FUTUROS

Como sugestão para trabalhos futuros, propõe-se:

- Uma análise comparativa de desempenho utilizando outras plataformas de computação paralela, como o OpenCL (*Open Computing Language*) (NVIDIA, 2012; MEMETI *et al.*, 2017).
- A implementação e avaliação do modelo *Variance Gamma* (MADAN; SENETA, 1990; HOYYI DEDI ROSADI, 2021), que oferece maior flexibilidade para modelar as assimetrias frequentemente observadas nos retornos de ativos financeiros.
- A expansão da aplicação para a análise de risco de portfólios, utilizando os milhares de cenários gerados pela SMC. Essa abordagem permitiria uma avaliação mais completa dos riscos de mercado e exigiria a modelagem da correlação entre os diferentes ativos da carteira (DESAI; LELE; VIENS, 2003; PEDERSEN, 2014).
- A criação de um modelo híbrido que combine dados sentimentais com modelos estatísticos (KALVA; NAGANJANEYULU, 2020; RUAN; JIANG, 2025). Neste sentido, sugere-se o desenvolvimento que incorpore a análise de sentimentos, extraída de notícias e redes sociais, para ajustar dinamicamente os parâmetros do modelo GBM. Isso permitiria que o modelo reagisse a eventos de mercado que não são capturados apenas pelos dados históricos de preços.
- O desenvolvimento de uma análise comparativa utilizando bibliotecas de alto nível para computação em GPU, como Numba em Python (NUMBA, 2015), a fim de avaliar tanto os ganhos de produtividade no desenvolvimento quanto o desempenho da solução em relação à implementação em linguagem C com CUDA.
- Uma investigação sobre a eficiência energética da solução proposta em comparação à execução em CPU, visto que a redução drástica no tempo de processamento proporcionada pelo paralelismo tende a diminuir o consumo total de energia para cargas de trabalho intensivas, tornando a abordagem em GPU frequentemente mais eficiente (HUANG; XIAO; FENG, 2009; MITTAL; VETTER, 2014).

REFERÊNCIAS

ADEBIYI, A.; ADEWUMI, A.; AYO, C. Stock price prediction using the arima model. In: **Proceedings - UKSim-AMSS 16th International Conference on Computer Modelling and Simulation, UKSim 2014**. [S.l.: s.n.], 2014.

ALERSTAM, E.; SVENSSON, T.; ANDERSSON-ENGELS, S. Parallel computing with graphics processing units for high-speed monte carlo simulation of photon migration. **Journal of Biomedical Optics**, SPIE-Intl Soc Optical Eng, v. 13, n. 6, p. 060504, 2008. ISSN 1083-3668. Disponível em: <http://dx.doi.org/10.1117/1.3041496>.

ALRABADI, D. W. H.; ALJARAYESH, N. I. A. Forecasting stock market returns via monte carlo simulation: The case of amman stock exchange. **Jordan Journal of Business Administration**, DAR Publishers / University of Jordan, v. 11, n. 3, p. 745–756, 2015.

AMIRI, H.; SHAHBAHRAMSI, A. Simd programming using intel vector extensions. **Journal of Parallel and Distributed Computing**, Elsevier, v. 135, p. 83–100, 2020.

ANDERSON, J. A. *et al.* Massively parallel monte carlo for many-particle simulations on gpus. **Journal of Computational Physics 254**, arXiv, 2012. Disponível em: <https://arxiv.org/abs/1211.1646>.

ARNAUT-BERILO, A.; ZAIMOVIć, A.; Nedzmija Turbo-Merdan. Effectiveness of monte carlo simulation and arima model in predicting stock prices. Unpublished, 2017. Disponível em: <http://rgdoi.net/10.13140/RG.2.2.14050.15046>.

B3. **Uma análise da evolução dos investidores na B3**. São Paulo: B3, 2024. Disponível em: <https://www.b3.com.br/pt_br/market-data-e-indices/servicos-de-dados/market-data/consultas/mercado-a-vista/perfil-pessoas-fisicas/perfil-pessoa-fisica/>. Acesso em: 21 maio 2025.

BASU, S. K. **Parallel and distributed computing**. Delhi, India: PHI Learning, 2016.

BOLLEN, J.; MAO, H.; ZENG, X. Twitter mood predicts the stock market. **Journal of Computational Science**, v. 2, n. 1, p. 1–8, 2011. ISSN 1877-7503. Disponível em: <https://www.sciencedirect.com/science/article/pii/S187775031100007X>.

BREALEY, R. A.; MYERS, S. C.; ALLEN, F. **Principles of corporate finance**. Thirteenth edition. New York, NY: McGraw-Hill Education, 2020. ISBN 9781260013900; 1260013901.

BROCK, D. C. (Ed.). **Understanding moore's law**. Philadelphia, PA: Chemical Heritage Foundation, 2006.

BRăTIAN, V. *et al.* Geometric brownian motion (gbm) of stock indexes and financial market uncertainty in the context of non-crisis and financial crisis scenarios. **Mathematics**, v. 10, n. 3, 2022. ISSN 2227-7390. Disponível em: <https://www.mdpi.com/2227-7390/10/3/309>.

BUJARI, A.; FURINI, M.; LAINA, N. On using cashtags to predict companies stock trends. In: **14th IEEE Annual Consumer Communications Networking Conference (CCNC)**. [S.l.: s.n.], 2017. p. 25–28.

BURG, D.; AUSUBEL, J. H. Moore's law revisited through intel chip density. **PLOS ONE**, Public Library of Science (PLoS), v. 16, n. 8, p. e0256245, ago. 2021. ISSN 1932-6203. Disponível em: <http://dx.doi.org/10.1371/journal.pone.0256245>.

CASTILLO, E. *et al.* Financial applications on multi-cpu and multi-gpu architectures. **The Journal of Supercomputing**, Springer Science and Business Media LLC, v. 71, n. 2, p. 729–739, nov. 2014. ISSN 1573-0484. Disponível em: <http://dx.doi.org/10.1007/s11227-014-1316-5>.

CHEN, W. *et al.* Mean–variance portfolio optimization using machine learning-based stock price prediction. **Applied soft computing**, Elsevier, v. 100, p. 106943, 2021.

CHUNDURU, A. Gpu parallel computing architectures: Unlocking the power of parallelism for high-performance applications. **International Journal of Scientific Research in Computer Science, Engineering and Information Technology**, v. 10, p. 390–396, 11 2024.

COOK, S. **A Short History of Supercomputing**. [S.l.]: Elsevier, 2013.

DESAI, R.; LELE, T.; VIENS, F. A monte-carlo method for portfolio optimization under partially observed stochastic volatility. In: **2003 IEEE International Conference on Computational Intelligence for Financial Engineering, 2003. Proceedings.** [S.l.: s.n.], 2003. p. 257–263.

DOAN, V. *et al.* Parallel pricing algorithms for multi-dimensional bermudan/american options using monte carlo methods. **Mathematics and Computers in Simulation**, v. 81, n. 3, p. 568–577, 2010. ISSN 0378-4754. The Sixth IMACS Seminar on Monte Carlo Methods Applied Scientific Computing VII. Forward Numerical Grid Generation, Approximation and Simulation. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0378475410002739>.

DUARTE, J. J.; GONZáLEZ, S. M.; CRUZ, J. C. Predicting stock price falls using news data: Evidence from the brazilian market. **Computational Economics**, Springer Science and Business Media LLC, v. 57, n. 1, p. 311–340, nov. 2020. ISSN 1572-9974. Disponível em: <http://dx.doi.org/10.1007/s10614-020-10060-y>.

ESTEMBER, R. D.; MARAñA, M. J. R. **Forecasting of Stock Prices Using Brownian Motion – Monte Carlo Simulation**. IEOM Society International, 2016. Disponível em: <https://index.ieomsociety.org/index.cfm/article/view/ID/3105>.

FLYNN, M. Very high-speed computing systems. **Proceedings of the IEEE**, v. 54, p. 1901 – 1909, 01 1966.

HOYYI DEDI ROSADI, A. A. **Journal of Mathematical and Computational Science**, SCIK Publishing Corporation, 2021. ISSN 1927-5307. Disponível em: <http://dx.doi.org/10.28919/jmcs/5469>.

HUANG, S.; XIAO, S.; FENG, W. On the energy efficiency of graphics processing units for scientific computing. In: **2009 IEEE International Symposium on Parallel amp; Distributed Processing**. IEEE, 2009. p. 1–8. Disponível em: <http://dx.doi.org/10.1109/IPDPS.2009.5160980>.

HWANG, K.; SU, S.-P.; NI, L. M. Vector computer architecture and processing techniques. **Advances in Computers**, Elsevier, v. 20, p. 115–197, 1981. ISSN 0065-2458. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0065245808604970>.

IBBETT, R. N. Vector processes. In: ____. **The Architecture of High Performance Computers**. New York, NY: Springer New York, 1982. p. 126–164. ISBN 978-1-4757-6715-5. Disponível em: <https://doi.org/10.1007/978-1-4757-6715-5_7>.

JABBOUR, G. M.; LIU, Y.-K. Option pricing and monte carlo simulations. **Journal of Business & Economics Research**, Clute Institute, Littleton, CO, USA, v. 3, n. 9, p. 1–6, September 2005. ISSN 1542-4448. Disponível em: <https://doi.org/10.19030/jber.v3i9.2802>.

JAMES D. WHITESIDE, I. A practical application of monte carlo simulation in forecasting. In: **2008 AACE International Transactions**. Morgantown, WV: AACE International, 2008. p. EST.04.1–EST.04.12. Available via ProQuest Document ID: 208193177.

JUAN, G.; RAFAEL, A.; JUNIOR, A. **O Potencial das Graphics Processing Units**. [S.l.], 2012.

KABBANI, T.; USTA, F. E. **Predicting The Stock Trend Using News Sentiment Analysis and Technical Indicators in Spark**. arXiv, 2022. Disponível em: <https://arxiv.org/abs/2201.12283>.

KALVA, S.; NAGANJANEYULU, S. Stock price prediction based on finance related news using nlp, lasso and arimax. **i-manager's Journal on Software Engineering**, i-manager Publications, v. 14, n. 4, p. 11, 2020. ISSN 2230-7168. Disponível em: <http://dx.doi.org/10.26634/jse.14.4.17661>.

KIRK, D. B.; HWU, W.-m. W. Programming massively parallel processors: A hands-on approach. In: **Programming Massively Parallel Processors: A Hands-on Approach**. Third. [S.l.]: Morgan Kaufmann, 2017.

KIRKBY, D. R.; DELPY, D. T. Parallel operation of monte carlo simulations on a diverse network of computers. **Physics in Medicine and Biology**, IOP Publishing, v. 42, n. 6, p. 1203–1208, jun. 1997. ISSN 1361-6560. Disponível em: <http://dx.doi.org/10.1088/0031-9155/42/6/016>.

KOçAK, D. **A Method to Increase the Power of Monte Carlo Method: Increasing the Number of Iteration**. Pedagogical Research, 2020. Disponível em: <https://www.pedagogicalresearch.com/article/a-method-to-increase-the-power-of-monte-carlo-method-increasing-the-number-of-iteration-6258>.

KUMAR, D.; SARANGI, P. K.; VERMA, R. A systematic review of stock market prediction using machine learning and statistical techniques. **Materials Today: Proceedings**, v. 49, p. 3187–3191, 2022. ISSN 2214-7853. National Conference on Functional Materials: Emerging Technologies and Applications in Materials Science. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2214785320390337>.

KYPER, E. Arima modeling with intervention to forecast and analyze chinese stock prices. **International Journal of Engineering Business Management**, v. 3, 01 2011.

LAI, J. *et al.* A multi-gpu parallel algorithm in hypersonic flow computations. **Mathematical Problems in Engineering**, v. 2019, p. 1–15, 03 2019.

LEE, C. M. C.; MYERS, J.; SWAMINATHAN, B. What is the Intrinsic Value of the Dow? **The Journal of Finance**, v. 54, n. 5, p. 1693–1741, out. 1999. ISSN 0022-1082, 1540-6261. Disponível em: <https://onlinelibrary.wiley.com/doi/10.1111/0022-1082.00164>.

LEYVA-SUáREZ, E.; HERRERA, G. S.; de la Cruz, L. M. A parallel computing strategy for monte carlo simulation using groundwater models. **Geofísica Internacional**, v. 54, n. 3, p. 245–254, 2015. ISSN 0016-7169. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0016716915000239>.

LONDHE, D. *et al.* A survey on gpu system considering its performance on different applications. **Computer Science Engineering: An International Journal (CSEIJ)**, v. 3, p. 11–19, 08 2013.

LOUNIS, M. *et al.* Gpu-based parallel computing of energy consumption in wireless sensor networks. In: **2015 European Conference on Networks and Communications (EuCNC)**. [S.l.: s.n.], 2015. p. 290–295.

MADAN, D. B.; SENETA, E. The variance gamma (v.g.) model for share market returns. **The Journal of Business**, University of Chicago Press, v. 63, n. 4, p. 511, jan. 1990. ISSN 1537-5374. Disponível em: <http://dx.doi.org/10.1086/296519>.

MEMETI, S. *et al.* **Benchmarking OpenCL, OpenACC, OpenMP, and CUDA: programming productivity, performance, and energy consumption**. arXiv, 2017. Disponível em: <https://arxiv.org/abs/1704.05316>.

MILOš, I.; DIGKOGLOU, P. The volatility of stock market returns: Application of monte carlo simulation. **Economics of Sustainable Development**, v. 6, p. 17–30, 01 2022.

MITTAL, S.; VETTER, J. S. A survey of methods for analyzing and improving gpu energy efficiency. **ACM Computing Surveys**, Association for Computing Machinery (ACM), v. 47, n. 2, p. 1–23, ago. 2014. ISSN 1557-7341. Disponível em: <http://dx.doi.org/10.1145/2636342>.

MOORE, G. E. Cramming more components onto integrated circuits, reprinted from electronics, volume 38, number 8, april 19, 1965, pp.114 ff. **IEEE Solid-State Circuits Society Newsletter**, v. 11, n. 3, p. 33–35, 2006.

MOURNING, C. *et al.* Gpu acceleration of robust point matching. In: ____. **Advances in Visual Computing**. Springer Berlin Heidelberg, 2010. p. 417–426. ISBN 9783642172779. Disponível em: <http://dx.doi.org/10.1007/978-3-642-17277-9_43>.

NAGARAJAN, D.; PRABHAKARAN, M. Prediction of stock price movements using monte carlo simulation. **International Journal of Innovative Technology and Exploring Engineering**, v. 8, p. 2012–2016, 10 2019.

NAVARRO, C.; HITSCHFELD, N.; MATEU, L. A survey on parallel computing and its applications in data-parallel problems using gpu architectures. **Communications in Computational Physics**, v. 15, p. 285–329, 09 2013.

NETO, A. A. **Mercado Financeiro**. 14. ed. [S.l.]: Atlas, 2018. 513 p.

NOFSINGER, J. R. Social mood and financial economics. **Journal of Behavioral Finance**, Informa UK Limited, v. 6, n. 3, p. 144–160, set. 2005. ISSN 1542-7579. Disponível em: <http://dx.doi.org/10.1207/s15427579jpfm0603_4>.

NUMBA. **Numba: a LLVM-based Python JIT compiler**. GitHub, 2015. Disponível em: <https://numba.pydata.org/>.

NVIDIA. **OpenCL™ Programming Guide for the CUDA™ Architecture, Version 4.2**. [S.l.], 2012. Disponível em: <https://developer.download.nvidia.com/compute/DevZone/docs/html/OpenCL/doc/OpenCL_Programming_Guide.pdf>.

____. **NVIDIA Ada Lovelace GPU Architecture**. [S.l.], 2022. Whitepaper. Disponível em: <https://images.nvidia.com/aem-dam/Solutions/geforce/ada/nvidia-ada-gpu-architecture.pdf>.

NVIDIA Corporation. **CUDA C Programming Guide**. 2025. Acessado em 27 de junho de 2025. Disponível em: <https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html>.

NYASHA, S.; ODHIAMBO, N. The brazilian stock market development: A critical analysis of progress and prospects during the past 50 years. **Risk Governance and Control: Financial Markets Institutions**, v. 3, p. 7–15, 09 2013.

OKSENDAL, B. **Stochastic Differential Equations: An Introduction with Applications**. Springer Berlin Heidelberg, 2013. (Universitext). ISBN 9783662036204. Disponível em: <https://books.google.com.br/books?id=gizqCAAAQBAJ>.

OLBERT, L. Industry-specific stock valuation methods – a literature review. **Journal of Accounting Literature**, Emerald, v. 47, n. 5, p. 52–70, maio 2024. ISSN 2452-1469. Disponível em: <http://dx.doi.org/10.1108/JAL-04-2023-0065>.

OWENS, J. *et al.* Gpu computing. **Proceedings of the IEEE**, Institute of Electrical and Electronics Engineers (IEEE), v. 96, n. 5, p. 879–899, maio 2008. ISSN 1558-2256. Disponível em: <http://dx.doi.org/10.1109/JPROC.2008.917757>.

PADMANAYANA; VARSHA; K, B. Stock market prediction using twitter sentiment analysis. **International Journal of Scientific Research in Science and Technology**, Technoscience Academy, p. 265–270, jul. 2021. ISSN 2395-6011. Disponível em: <http://dx.doi.org/10.32628/CSEIT217475>.

PARUNGROJRAT, N.; KIDSOM, A. Stock price forecasting: Geometric brownian motion and monte carlo simulation techniques. **MUT Journal of Business Administration**, Mahanakorn University of Technology, v. 16, n. 1, p. 90–103, 2019.

PEDERSEN, M. E. H. Portfolio optimization and monte carlo simulation. **SSRN Electronic Journal**, Elsevier BV, 2014. ISSN 1556-5068. Disponível em: <http://dx.doi.org/10.2139/ssrn.2438121>.

POUDEL, R. **Monte Carlo Simulation**. [S.l.], 2022.

PRAKHAR, K. *et al.* Effective stock price prediction using time series forecasting. In: **2022 6th International Conference on Trends in Electronics and Informatics (ICOEI)**. [S.l.: s.n.], 2022. p. 1636–1640.

RAM, J. A review on gpu architectures and programming. In: **International journal of scientific research in science, engineering and technology**. [S.l.]: Technoscience Academy, 2023.

RATHGEBER, A.; STADLER, J.; STöCKL, S. Modeling share returns - an empirical study on the variance gamma model. **Journal of Economics and Finance**, v. 40, 02 2015.

RAYCHAUDHURI, S. Introduction to monte carlo simulation. In: **2008 Winter Simulation Conference**. [S.l.: s.n.], 2008. p. 91–100.

REDDY, D.; CLINTON, V. Simulating stock prices using geometric brownian motion: Evidence from australian companies. **Australasian Accounting, Business and Finance Journal**, v. 10, p. 23–47, 01 2016.

REIS, L.; MEURER, R.; SILVA, S. D. **Stock returns and foreign investment in Brazil**. [S.l.], 2008.

RUAN, L.; JIANG, H. Stock price prediction using finbert-enhanced sentiment with shap explainability and differential privacy. **Mathematics**, v. 13, n. 17, 2025. ISSN 2227-7390. Disponível em: <https://www.mdpi.com/2227-7390/13/17/2747>.

SANDERS, J.; KANDROT, E. **CUDA by example**. Boston, MA: Addison-Wesley Educational, 2010.

SANTOS, A. D. Quem está pagando juros sobre capital próprio no Brasil? **Revista Contabilidade & Finanças**, v. 18, n. spe, p. 33–44, jun. 2007. ISSN 1519-7077. Disponível em: <http://www.scielo.br/scielo.php?script=sci_arttext&pid=S1519-70772007000300004&lng=pt&tlng=pt>.

SCOTT, D. Box–muller transformation. **Wiley Interdisciplinary Reviews: Computational Statistics**, v. 3, 03 2011.

SEVRANI, K. **Empirical analysis of the parallelization of the Monte Carlo simulation and its usage in the field of Economy**. [S.l.], 2011.

SHAH, D.; ISAH, H.; ZULKERNINE, F. Stock market analysis: A review and taxonomy of prediction techniques. **International Journal of Financial Studies**, v. 7, n. 2, 2019. ISSN 2227-7072. Disponível em: <https://www.mdpi.com/2227-7072/7/2/26>.

SHUKUR, A. *et al.* Application of the box-muller transformation in generating normally distributed random variables: A numerical approach. v. 4, p. 32–43, 05 2024.

SIDDIAUI, S. S.; PATIL, V. A. Stock market valuation using monte carlo simulation. In: **2018 International Conference on Current Trends towards Converging Technologies (ICCTCT)**. [S.l.: s.n.], 2018. p. 1–7.

SINGH, T. R. **A Study on Monte Carlo Simulation for Stock Price Forecasting**. Tese (DBA Dissertation) — Swiss School of Business and Management, Geneva, November 2021.

SONONO, E.; MASHELE, H. Prediction of stock price movement using continuous time models. **Journal of Mathematical Finance**, v. 05, p. 178–191, 01 2015.

STALLINGS, W. **Arquitetura e organização de computadores**. S.l.: Editora Pearson, 2017. v. 10. ISBN 9788543020532.

SUN, Y. *et al.* **Summarizing CPU and GPU Design Trends with Product Data**. arXiv, 2019. Disponível em: <https://arxiv.org/abs/1911.11313>.

VUONG, P. H. *et al.* A bibliometric literature review of stock price forecasting: From statistical model to deep learning approach. **Science Progress**, v. 107, n. 1, p. 00368504241236557, 2024. PMID: 38490223. Disponível em: <https://doi.org/10.1177/00368504241236557>.

____. A bibliometric literature review of stock price forecasting: From statistical model to deep learning approach. **Science Progress**, SAGE Publications, v. 107, n. 1, jan. 2024. ISSN 2047-7163. Disponível em: <http://dx.doi.org/10.1177/00368504241236557>.

WANG, Y.; GUO, Y. Forecasting method of stock market volatility in time series data based on mixed model of arima and xgboost. **China Communications**, IEEE, v. 17, n. 3, p. 205–221, 2020.

WEIGEL, M. **Monte Carlo methods for massively parallel computers**. 2017. Disponível em: <https://arxiv.org/abs/1709.04394>.

World Economic Forum. **2024 Global Retail Investor Outlook**. [S.l.], 2025. Disponível em: <https://reports.weforum.org/docs/WEF_2024_Global_Retail_Investor_Outlook_2025.pdf>.

XIANG, J. N. P.; VELU, S.; ZYGIARIS, S. Monte carlo simulation prediction of stock prices. In: **2021 14th International Conference on Developments in eSystems Engineering (DeSE)**. [S.l.: s.n.], 2021. p. 212–216.

YANG, Z.; ALDOUS, D. Geometric brownian motion model in financial market. **University of California, Berkeley**, Citeseer, 2015.

APÊNDICE A – DECLARAÇÃO DE USO DE INTELIGÊNCIA ARTIFICIAL

Durante a preparação deste trabalho, o autor utilizou a ferramenta Gemini[1] versão 2.5 Pro para revisão ortográfica, ajustes de coesão textual e auxílio no desenvolvimento dos códigos em Python utilizados para geração de gráficos. Após o uso desta ferramenta, o autor revisou e editou o conteúdo e assume total responsabilidade pelo conteúdo da publicação.

[1] https://gemini.google.com/