Parallelizing Adaptive Multilevel Monte Carlo Methods for European Option Pricing (Project Proposal)

Kyle Lee (kylel), Matt Wei (mwei2)

November 2025

1 Project Page URL

`https://kyleleesea.github.io/CUDA-AdaptiveMultiLevelMonteCarlo/`

2 Summary

We will design a parallel algorithm for an Adaptive Multilevel Monte Carlo method for European option payoff estimation using CUDA.

3 Background

Options are a financial instrument with value derived from an underlying asset such as a stock. Options allow the holder the right to buy (call option) or sell (put option) at a prespecified price. In the case of a call option this means that if the underlying asset increased in price above the prespecified price (called the strike price) then the owner profits the difference.[1] There are multiple variations of options with different payoff schemes, European options allow the holder to exercise the option on a specific date. For example if I had a European call option for Apple stock dated 1 year from now with a strike price of $280 and 1 year from now the price of Apple stock rose to $300 then if I chose to exercise the option I'd profit $20.

Traders utilize options to profit from the volatility of stocks. To effectively utilize options traders use models to predict the payoff of an option. These include parametric methods (Black-Scholes, Partial Differential Equations, Monte Carlo

[1] https://www.investopedia.com/terms/o/option.asp (Options: Types, Spreads, Risk Metrics)

Methods, ...) and nonparametric methods (Regression, DNNs, ...).[2] Monte Carlo methods are a parametric method where we generate some number of samples each using independent random variables to simulate volatility and then combining these samples to make a prediction. For European stock option payoff calculation, high-level pseudocode may look like (note, this is highly abstracted):

Algorithm 1 Sequential Simple Monte Carlo

```
function MonteCarlo(numSamples, granularity, optionParams)
    payoffSum ← 0
    for $i = 1$ to numSamples do
        payoffSum ← payoffSum + GenerateSample(granularity, optionParams)
    end for
    return payoffSum / numSamples
end function
function GenerateSample(granularity, optionParams)
    sample ← optionParams.initial
    for $i = 1$ to granularity do
        $Z$ ← SampleNormal
        sample ← sample + TakeTimestep(optionParams, $Z$)
    end for
    return sample
end function
```

TakeTimestep is included as a black box for now but Euler-Mauryama discretization and Milstein discretization could both be utilized to implement this function.[3]

In the real world Adaptive Multilevel Monte Carlo methods are used rather than the simple abstraction shown above. Adaptive refers to the number of samples and granularity being computed by the algorithm to minimize an objective function and the algorithm only ends when convergence is achieved. Multilevel is a scheme to reduce the time complexity to reach convergence by having different levels with different granularities. This enables us to have more samples with coarse granularity (fewer timesteps) and fewer samples with fine granularity (more timesteps). We plan to implement and parallelize an adaptive multilevel approach.

[2] https://link.springer.com/article/10.1007/s11831-025-10343-3 (From Traditional to Computationally Efficient Scientific Computing Algorithms in Option Pricing: Current Progresses with Future Directions)

[3] https://arxiv.org/pdf/1505.00965 (An Introduction to Multilevel Monte Carlo for Option Valuation)

4 The Challenge

Effectively parallelizing Adaptive Multilevel Monte Carlo on GPUs is still an open research problem because work balancing is challenging. At the start of an iteration, we know how many samples at each level we need and the number of timesteps each sample needs. We then need to effectively map this workload to our available compute resources.

Additionally, the scale of the computation is massive with coarse samples taken on the order of billions and fine samples having timesteps taken up to 2^{40} per sample. Lastly, for Multilevel Monte Carlo to converge samples between fine and coarse levels need to be coupled, introducing dependency and increasing the challenge. So this project presents the following challenges and learning opportunities:

- Work is quantified in two dimensions: Number of samples and timesteps. Timesteps for a given sample should have nearby locality for us to reduce them together. Samples for a given level should have nearby locality to compute an average. How do we effectively map the work to our thread blocks to maximize locality?

- The number of total timesteps per iteration likely exceeds the total memory of the GPU. This will require us to figure out how to effectively batch our work and transfer intermediate results while minimizing expensive memory accesses.

- Different levels exhibit different patterns of number of samples to number of timesteps per sample. This will give us an opportunity to explore how to optimize each level and therefore explore how to parallelize different computational patterns. We'll explore tradeoffs between utilizing atomics vs. locks vs. increasing the number of batches.

- Each timestep requires us to sample from a normal distribution and generating randomness can be expensive. Therefore we'll learn how to effectively utilize libraries, such as cuRAND, to generate randomness in parallel.

5 Resources

We'll reference existing research on parallelizing Monte Carlo methods to understand the existing literature on the topic and identify areas to improve. These resources include:

- A massively parallel implementation of multilevel Monte Carlo for finite element models, Mathematics and Computers in Simulation, Santiago Badia, Jerrad Hampton, Javier Principe. 2023.
- On the implementation of multilevel Monte Carlo simulation of the stochastic volatility and interest rate model using multi-GPU clusters, Harold A. Lay, Zane Colgin, Viktor Reshniak and Abdul Q. M. Khaliq. May 2, 2018.
- Efficient Asian option pricing with CUDA, A. Yuzhanin, I. Gankevich, E. Stepanov and V. Korkhov. 2015.
- Pricing American Options with Least Squares Monte Carlo on GPUs, Massimiliano Fatica and Everett Phillips. November 18, 2013.

For the Adaptive Multilevel Monte Carlo implementation we will work off Mike Giles' existing C++ code. Upon an initial look, we will still need to do additional work to streamline it into our specific European Option call pricing case.[4]. Mike Giles is the originator of the Multilevel Monte Carlo paper so we consider his work as an authoritative starting point. Giles also has an existing CUDA implementation which we will research to identify alternative approaches but we will not utilize his existing CUDA code in our implementation or copy any of it directly.

6 Goals and Deliverables

50% Goals:

- Implement sequential Adaptive Multilevel Monte Carlo for European stock option pricing using the Milstein discretization
- Gather existing papers on the subject and conduct a literature review
- Develop a test suite reflecting diverse epsilon levels (affecting how long convergence takes) and real world stock prices

75% Goals:

- Benchmark the sequential implementation utilizing the test suite described above
- Parallelize coarse workloads (levels with many samples but few timesteps per sample) on CUDA

[4]https://people.maths.ox.ac.uk/ gilesm/mlmc/ (”Multilevel Monte Carlo software ”)

- Benchmark the parallelized coarse workloads against the sequential implementation

100% Goals:

- Parallelize fine grained workloads (levels with many timesteps but few samples) on CUDA
- Optimize and refine the parallel implementation
- Benchmark the different parallel implementations and refinements
- Write the final paper and poster

125% Goals:

- Chain together MPI and CUDA to make the approach multi-node

150% Goals:

- Optimize algorithm to beat the Lay paper in both efficiency and performance.

In terms of performance, the 2018 paper by Lay, et al. achieved a $24.38\times$ speedup using "a five node cluster with each node consisting of two NVIDIA Tesla K40 GPUs, Intel QDR InfiniBand 40 Gb/s interconnect, 10 GigE data network, and 64 GB of RAM per node."[5].

Our 100% performance goal is a $\geq 12\times$ speedup over the sequential implementation with code run on GHC or PSC with a single GPU node. If we achieve this goal, then we will have significantly beat the Lay paper in terms of efficiency since we'd be using one GPU vs. them using $2 \cdot 5 = 10$ GPUs.

Our 150% performance goal is to be both more efficient (use less total compute) and performant (higher speedup) in comparison to the Lay paper. This would be $\geq 25\times$ speedup over the sequential implementation with code run on < 10 GPUs.

At the poster session we'll show performance and efficiency graphs our of approaches benchmarked against the sequential approach. We can also do a live demo showing the performance and calculated payoff for a user provided stock option.

[5]https://doi.org/10.1515/mcma-2018-2025

7 Platform Choice

We will utilize C++ and CUDA to implement our program. We will debug on GHC and benchmark on PSC. We utilize CUDA because the algorithm exhibits significantly taxing workloads and GPUs are well suited to handle the large amount of work. Additionally, we'll need to generate large numbers of normal distribution samples which we can do efficiently on GPUs using cuRAND or thrust. We're also very interested in exploring GPU programming more, which also motivates this choice.

We'll debug on GHC to avoid using up too much PSC course allowance. We benchmark on PSC so that if we get to our 125% goal we'll be benchmarking against the same hardware when expanding to multiple nodes.

8 Schedule

Nov 10 - Nov 16

- Decide on project and write project proposal (completed!)
- Begin gathering papers and literature to review

Nov 17 - Nov 23

- Gain deeper understanding of Adaptive Multilevel Monte Carlo by reading papers
- Write sequential algorithm
- Review papers on existing parallel designs and identify alternative approaches
- Develop benchmarking and test suite

Nov 24 - Nov 30

- Benchmark the sequential implementation
- Parallelize coarse grained workloads on CUDA
- Benchmark the parallelized approach

Dec 1 - Dec 8

- Parallelize fine grained workloads on CUDA
- Benchmark the parallelized approach
- Write final paper and poster
- If time permits work on 125% and 150% goals

9 Work Split

We're aiming to do a 60 (Kyle)/40 (Matt) workload split since Kyle has more bandwidth near the end of the semester. We'll both work together on the literature and paper review to both understand the underlying algorithm and existing parallel designs. We'll both ideate and pair program to create the initial parallelization approaches. We'll both run experiments and collaborate via pair programming, whiteboarding, and research to further optimize our approach. For the final paper, poster, and milestone report we'll divide the sections to write 60/40.