

Parallelization and Acceleration of Dynamic Option Pricing Models on GPU-CPU Heterogeneous Systems

Brian Wesley MUGANDA

Institute of Mathematical Sciences, Strathmore University, Ole Sangale Rd., Nairobi 00200, Kenya
E-mail: bmuganda@strathmore.edu

Bernard Shibwabo KASAMANI

School of Computing and Engineering Sciences, Strathmore University, Ole Sangale Rd., Nairobi
00200, Kenya
E-mail: bshibwabo@strathmore.edu

Abstract In this paper, stochastic global optimization algorithms, specifically, genetic algorithm and simulated annealing are used for the problem of calibrating the dynamic option pricing model under stochastic volatility to market prices by adopting a hybrid programming approach. The performance of this dynamic option pricing model under the obtained optimal parameters is also discussed. To enhance the model throughput and reduce latency, a heterogeneous hybrid programming approach on GPU was adopted which emphasized a data-parallel implementation of the dynamic option pricing model on a GPU-based system. Kernel offloading to the GPU of the compute-intensive segments of the pricing algorithms was done in OpenCL. The GPU approach was found to significantly reduce latency by an optimum of 541 times faster than a parallel implementation approach on the CPU, reducing the computation time from 46.24 minutes to 5.12 seconds.

Keywords parallelization; GPU computing; option pricing; GPU acceleration; stochastic volatility; hybrid programming

1 Introduction

The use of artificial intelligence in the financial services industry has the potential to transform the sector through greater efficiencies, cost reductions and better tools to draw intelligence from large datasets. Globally, the financial services industry increasingly requires access to scalable, accurate and reliable computing power due to increased competition, the volume of data, products and complexity in models. At the same time, the requirements by regulation authorities have also amplified the need for fast and reliable computing capacities for financial institutions by imposing higher standards for availability, system integrity and security. This has exalted pressure on the financial institutions to improve business efficiency by cutting development times and the costs for the introduction of new products and models^[1–3].

Over the past decades, significant progress has been made in the development of mathematical models for pricing and risk management, and the use of state-of-the-art workstations for their solutions. This increase in computing demands has been brought about by the tremendous

growth in both the scope and size of problems being addressed and the complexity of models being used. However, some of the more complex analytical models would exhaust the computer capabilities of the workstations, and analysts would then ideally resort to the use of clusters of workstations. By implementing massively parallel and distributed architecture, it is possible to achieve a great deal of flexibility, reliability and increase the performance-to-cost ratios for any institution. This is achieved by having a system of shared workstations that can solve a large complex task in a fraction of time than a single computer would, and thus eliminate the need to have constant upgrading to large computing hardware^[1,4,5].

Traditionally, large tasks that require constant re-computation could be accelerated by implementing parallel architectures which enable achieving the low latency requirements of computationally extensive and expensive problems which would be solved within microseconds. It is desirable to have complex optimization and forecasting techniques developed in addition to having the capacity to perform real-time calibrations of market valuation, risk sensitivities in addition to implementing stress tests, back tests and performance tests for all strikes and all maturities on an entire trading book in real-time. This would enable financial analysts to make timely, accurate investment and hedging decisions. This ever-increasing complexity and scope of problems being tackled in the area of computational finance has therefore brought about a need for efficient pricing architectures that are both powerful and flexible and which can accommodate complex models which capture implied volatility skews, stochastic volatility, stochastic interest rates and also jumps in the pricing of options. This would be an impossible task for financial risk managers without appropriate methodology, advanced analytical tools, and a first-class technological infrastructure^[1,4,6,7].

The growing use of powerful Graphics Processing Units (GPUs) has suggested the idea of developing algorithms to price financial options that are highly performing when executed on GPUs. Since GPUs are ubiquitous, they can be cheaper platforms for parallel computing. This can result in substantial savings in the computing times for financial products evaluation in comparison to those obtained when using the CPU. These savings could be used to speed up the decision making process in financial institutions such as banks, insurance companies and hedge funds since in these institutions departments usually evaluate a large number of contracts with varying parameter values and varying strike, maturity and volatility levels. This desirable timeliness feature of the decision making process in financial institutions is thus very well suited for parallelized, accelerated and/or distributed computing. The inherently parallel, multi-processor structure of the GPUs makes them especially attractive for parallelization in multiple problem solving contexts. GPUs usually exist in personal computers, supercomputers and mobile devices such as tablets and smartphones^[8,9].

GPUs have been found to offer a superior level of performance over CPUs, both in the achievable computational power as well as the memory bandwidth^[10,11]. Many financial measures are stochastic and require a high number of simulations. Option pricing, Value-at-Risk (VaR) calculations and hedging strategies require rapid adaptation with suitable techniques. Also, automated trading algorithms need fast execution to make gain from opportunities. Financial establishments using grid computing with CPU have pointed out excessive cost in hardware and electricity consumption. As it was in the case of Aon Benfield, a world's leading insurance

company which for a bond pricing service spent \$4 million in a grid architecture using CPUs and \$1.2 million in electricity a year^[12]. On the contrary, a GPU based pricing engine would only have cost \$144,000 and \$31,000 in electricity a year. For instance, for J.P. Morgan's Equity Derivatives Group, the equity derivative focused risk computation is performed on hybrid GPU-based systems, increasing performance by 40x compared to only CPU-based systems for the same electric power^[13].

Among the works done on GPU acceleration in option pricing, most have focused on numerical implementation on GPU based on lattice methods such as in Dai, et al.^[10], Ganesan, et al.^[14], Solomon, et al.^[15], Zhang, et al.^[16], Suo, et al.^[17]; or based on Monte Carlo simulation such as in Michael and Françoise^[18], Podlozhnyuk and Mark^[19], Abbas-Turki and Bernard^[20], Trainor and Crookes^[21], Grauer-Gray, et al.^[22] and Yu, et al.^[23]; or Fourier methods and partial differential approximations such as in Dang^[24], Dang, Christara and Jackson^[25], Surkov^[4] and COS approach on GPU such as in Zhang and Oosterlee^[26]. These implementations were done for a particular option pricing model and option type such as American, Asian, Basket, European or multi-asset options. They found that running time experiments on the GPU implementation achieved significant speedup compared with the CPU-only case; and that with the growing the number of time steps for the problem, the CPU time increased much more sharply than the GPU time.

The complexities of dynamic option pricing models under stochastic volatility and the large size of the computing problem would require the designing of a much more efficient pricing system using alternative approaches and adoption of better suited model calibration algorithms. By implementing a parallel GPU accelerated architecture for such a pricing system that makes use of the dynamic option pricing model and uses stochastic optimization for the model calibration process, then a great deal of efficiency, flexibility, reliability and increased performance-to-cost ratios in the implementation of these models can be achieved. This paper thereby makes use of stochastic optimization algorithms – Genetic Algorithm and Simulated Annealing – to calibrate the developed dynamic models to market prices and consequently implements the dynamic option pricing model on a hybrid parallel GPU architecture for acceleration of option price evaluation. The performance of these models under the optimal parameters is evaluated as well as the computational performance of the parallel GPU architectural deployment. The objectives of this paper were therefore: (i) to examine the use of stochastic optimization algorithms in the calibration of the dynamic option pricing models under stochastic volatility and (ii) to evaluate the performance of the dynamic pricing models on the parallel GPU architecture.

This paper is therefore arranged as follows: Section 2.1 presents the model calibration approach adopted, Section 2.2 presents the stochastic optimization methods, and Section 3 presents the results of implementation of the calibration algorithms for the dynamic option pricing models and GPU acceleration results. Section 4 concludes the paper.

2 Methodology

2.1 Dynamic Pricing Model Calibration

The calibration process is the determining of the set of model parameters that would match the market prices of a set of options. It is ideally an inverse problem to the option pricing

problem. When the market prices are used to calibrate the models, more complex exotic options can be priced using the models and risk sensitivities and hedge ratios can be obtained accurately. Calibration of the model is a crucial process and a price to pay with more complex model is the increased complexity of the calibration process. The industry standard approach is to minimize the difference between the observed prices and the model prices^[27].

The problem of calibration the dynamic models is formulated as an optimization problem with the aim of minimizing the objective function $G(\theta)$ between the model prices and the market prices for a set of the traded options.

$$\inf_{\theta} G(\theta), \quad G(\theta) = \sum_{i=1}^N w_i |C_i^{\theta}(t, S_t, T_i, K_i) - C_i^*(T_i, K_i)|^2, \quad (1)$$

where N denotes the number of observations of options used for parameter estimation, w_i is an equal weight for ATM options, $C_i^*(T_i, K_i)$ is the market price of the call options observed at time t . $C_i^{\theta}(t, S_t, T_i, K_i)$ denotes the model prices computed using the vector θ of the model parameters from the dynamic option pricing under square-root stochastic volatility model, where the set of parameters is $\theta = (v_0, \kappa, \alpha, \theta)$ for the dynamic model under square-root stochastic volatility.

The model prices $C_i^{\theta}(t, S_t, T_i, K_i)$ are obtained by the formula by Muganda, Kyriakou and Kasamani^[28] as:

$$C_i^{\theta}(t, S_t, T_i, K_i) = S_u \Sigma_{u,T} h(\Phi(d_1), F_1(\hat{v}; a_1, b_1)) - K e^{-r(T-u)} h(\Phi(d_2), F_2(\hat{v}; a_2, b_2)), \quad (2)$$

where

$$\begin{aligned} d_1 &= \frac{\ln \frac{S_u}{K} + (r + \frac{1}{2}\hat{v})(T-u)}{\sqrt{(1-\rho^2)\hat{v}(T-u)}}, \\ d_2 &= \frac{\ln \frac{S_u}{K} + (r - \frac{1}{2}\hat{v})(T-u)}{\sqrt{(1-\rho^2)\hat{v}(T-u)}}, \\ \Sigma_{u,T} &= e^{-0.5\rho^2\hat{v}(T-u)}, \end{aligned}$$

S_u is the current stock price, r is the risk-free interest rate, K is the strike price, \hat{v} is the estimate of the variance, h is the conditional copula h-function with respect to the second argument with dependence parameter φ , ρ denotes the correlation parameter, and Φ is the standard normal cumulative distribution function. $F_1(\hat{v}; a_1, b_1)$ and $F_2(\hat{v}; a_2, b_2)$ are the cumulative Gamma distribution functions with arguments for the scale and shape of the distribution as $a_1 = \frac{\tilde{m}_f}{b_1}$ and $b_1 = \frac{s_f^2}{\tilde{m}_f}$, $a_2 = \frac{m_f}{b_2}$ and $b_2 = \frac{s_f^2}{m_f}$ and that:

$$\begin{aligned} m_f &= v_u \left[\frac{1 - e^{-\alpha(T-u)}}{\alpha(T-u)} \right] + \beta \left[\frac{e^{-\alpha(T-u)} + \alpha(T-u) - 1}{\alpha(T-u)} \right], \\ \tilde{m}_f &= v_u \left[\frac{1 - e^{-\tilde{\alpha}(T-u)}}{\tilde{\alpha}(T-u)} \right] + \tilde{\beta} \left[\frac{e^{-\tilde{\alpha}(T-u)} + \tilde{\alpha}(T-u) - 1}{\tilde{\alpha}(T-u)} \right], \\ &\text{with } \tilde{\alpha} = \alpha - \gamma^2 \text{ and } \tilde{\beta} = \frac{\alpha\beta}{\alpha - \gamma^2}, \end{aligned}$$

$$s_f^2 = \gamma^2 v_u \left[\frac{1 - 2\alpha(T-u)e^{-\alpha(T-u)} - e^{-2\alpha(T-u)}}{\alpha^3(T-u)^2} \right] + \gamma^2 \beta \left[\frac{e^{-2\alpha(T-u)} + 2\alpha(T-u) + 4(\alpha(T-u) + 1)e^{-\alpha(T-u)} - 5}{2\alpha^3(T-u)^2} \right],$$

α , β and γ represent the mean-reversion speed, long-run mean and volatility of volatility rate for the stochastic volatility square-root process.

This function $G(\theta)$ in Equation (1) is not of any particular structure or form and is not necessarily convex such that it could be having more than one local minimum or both local and global minima, and it is not clear whether a gradient based algorithm would achieve a unique minimum, or whether if a local optimizer such as the nonlinear least square optimization is used, the solution would not be stuck at a local minimum. Therefore, due to this and the possible sensitivity of this function to the initial parameters, the use of stochastic optimization algorithms is suggested. The stochastic global optimization algorithms considered were the genetic algorithm and the simulated annealing algorithm^[27,29].

2.2 Stochastic Optimization Algorithms

Optimization schemes can be broadly categorized into two groups — Local and global schemes. Local optimization schemes, such as gradient based methods tend to start from their initial parameter estimates and then choose new parameter estimates such that the value of the objective function always moves towards an optimum value. The schemes will continue until a stationary point is found, and as such, tend to locate local optimums rather than global ones due to the choice of initial parameters. Global optimization schemes, on the other hand, are much less sensitive to initial parameter estimates. Many of these methods fall into the category of stochastic optimization schemes since they generate and use random variables to assist in locating an optimum. Stochastic optimization schemes use random variates to generate and accept new parameter values. This helps the algorithms from being stuck in a region of a local optimum rather than being at the global optimum. This increased flexibility however comes at a computational cost. The stochastic optimization schemes applied to the calibration of the dynamic stochastic volatility model are the Genetic Algorithm and the Simulated Annealing Algorithm^[27,29].

2.2.1 Genetic Algorithm

The Genetic Algorithm is inspired by evolutionary biology, hence the procedure starts with a population of solutions with the objective function being a fitness function which is to be maximized and the iterations are the generations. The new candidate solutions which are called children are created by crossover, which refers to a mixing of the existing solutions, and mutation which refers to the randomly changing of component solutions. A selection among parents and children solutions takes place at the end of each generation. The survival of a solution may be stochastic with the probability of survival proportional to a solution's fitness^[29]. The genetic algorithm is illustrated in Figure 1.

```

1: Randomly generate initial population  $P$  of solutions
2: while stopping criteria not met do
3:   Select  $P' \subseteq P$ , initialize  $P'' = \emptyset$  ▷ Set of children
4:   for  $i = 1$  to  $n$  do
5:     Randomly select individuals  $x^a$  and  $x^b$  from  $P'$ 
6:     Apply crossover to  $x^a$  and  $x^b$  to produce  $x^{\text{child}}$ 
7:     Randomly mutate child  $x^{\text{child}}$ 
8:      $P'' = P'' \cup x^{\text{child}}$ 
9:   end for
10:   $P = \text{survive}(P', P'')$ 
11: end while
12: return best solution

```

Figure 1 Genetic algorithm

2.2.2 Simulated Annealing Algorithm

The Simulated Annealing Algorithm starts with a random solution x^c and creates a new solution x^n by adding a small perturbation to x^c . If the new solution is better than the current one ($\Delta < 0$), it is accepted and replaces x^c . In the case, x^n is worse, the algorithm does not reject it right away but applies a stochastic acceptance criterion, thus there would still be a chance that the new solution will be accepted, albeit only with a certain probability. This probability is a decreasing function of both the order of magnitude of deterioration and the time the algorithm has already run. This time factor is controlled by the temperature parameter T which is reduced over time; hence, impairments in the objective function become less likely to be accepted and, eventually, the algorithm turns into a standard local search. The algorithm stops after a predefined number of iterations R_{\max} as illustrated in Figure 2^[29].

```

1: Set  $R_{\max}$  and  $T$ 
2: Randomly generate current solution  $x^c$ 
3: for  $r = 1$  to  $R_{\max}$  do
4:   while stopping criterion not met do
5:     Generate  $x^n \in \mathcal{N}(x^c)$  ▷ Neighbor to current solution
6:     Compute  $\Delta = f(x^n) - f(x^c)$  and generate  $u$  (uniform random variable)
7:     if  $\Delta < 0$  or  $e^{-\Delta/T} > u$  then
8:        $x^c = x^n$ 
9:     end if
10:  end while
11: end for

```

Figure 2 Simulated annealing algorithm

2.3 GPU Acceleration

Unlike CPU that works at really high frequency to achieve high speed, GPUs have a parallel architecture composed of numerous streaming multiprocessors that work at a lower frequency allowing for lower power consumption and faster speed if the algorithm is parallelizable. A GPU

has multiple streaming multiprocessors (SM) that contain memory registers for threads to use, several memory caches (shared memory, constant cache, texture memory, L1 cache), thread schedulers. Each core also consists of an Arithmetic logic unit (ALU) that handles integer and single precision calculations and a Floating-point unit (FPU) that handles double precision calculations. The number of multiprocessor cores depends on microarchitecture generation; they may be different for NVIDIA, Intel and AMD graphic cards^[30–32].

In the GPU architecture, which is optimized for parallel tasks and has significantly more compute resources; we can leverage on the high throughput and high performance per watt that the CPU can provide. The GPU could accelerate applications running on the CPU by offloading some of the compute-intensive and time-consuming portions of the code. The rest of the application still runs on the CPU. From a user's perspective, the application runs faster because it is using the massively parallel processing power of the GPU to boost performance^[30–32].

GPU acceleration enables use of highly optimized function to perform 2X-10X faster than the CPU-only parallelization alternatives. GPU acceleration within Python can be performed by use of acceleration libraries adopting NVIDIA CUDA or OpenCL implementation. OpenCL (Open Computing Language) is the standard framework for writing portable applications that work on cross-platform CPU, GPUs, mobile devices and embedded platforms^[33,34]. The OpenCL standard is supported by various vendors, including — AMD, ARM, Intel IBM, NVIDIA, Qualcomm, while CUDA is only supported by NVIDIA.

2.4 Parallelization Execution Strategy on OpenCL

In the development of the parallel algorithms on the GPU, we make use of PyOpenCL which enables the use of Python workflow in the deployment. Particularly, OpenCL is used to parallelize and accelerate the solution of dynamic option pricing models. These pricing algorithms are written into native OpenCL code by dividing the task of computing the option prices using the dynamic models to be executed in parallel on the GPU through the kernel which we assigned work-items based on the number of contracts from 85 to 10,000 which represented an increase of computed prices from 66,300 to 7,800,000 respectively. Data movements were minimized by efficiently managing the memory access patterns such that the data transfers were only at termination of parallelization by the work groups to the global memory spaces. Figure 3 below illustrates the implemented parallelization strategy used in the GPU parallelization of the dynamic option pricing models.

The kernel executed the OpenCL GPU device within a well-defined context managed by the host which executes the host program. This context defines the environment within which kernels execute. It included the following resources:

1. Devices: OpenCL GPU Device which was 'Intel(R) HD Graphics 630' on 'Intel(R) OpenCL' on the host 'Intel(R) Core (TM) i7-8705G CPU @ 3.10 GHz'. Graphics Base Frequency 350 MHz, Graphics Max Dynamic Frequency 1.10 GHz, Graphics Video Max Memory 64 GB, Max Resolution (HDMI) 4096 x 2160 @30Hz.
2. CPU Host: 'Intel(R) Core (TM) i7-8705G CPU @ 3.10 GHz'. with 4 total Cores, 8 total threads; Max Turbo Frequency, 4.10 GHz, Processor Base Frequency 3.10 GHz, Cache 8 MB, Memory Types DDR4-2400, OS Windows 10.

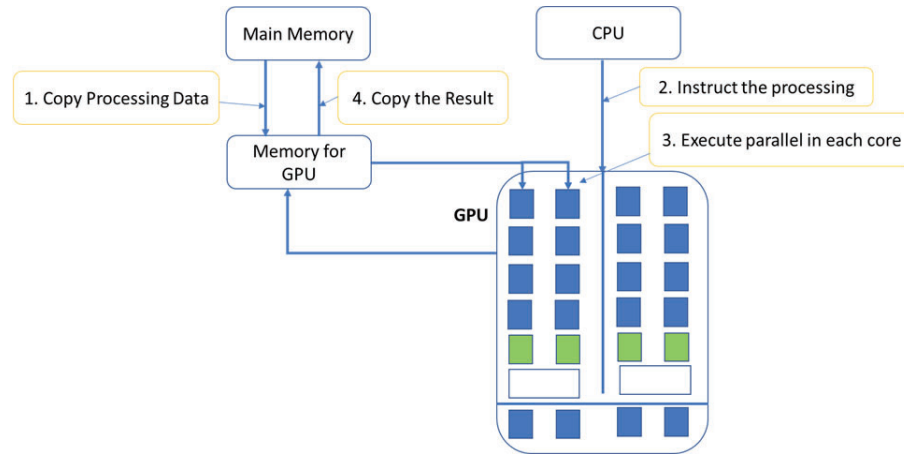


Figure 3 Parallel execution model implemented

The model was also implemented for the parallelization on the CPU by defining the platform device to be ‘Intel(R) Core (TM) i7-8705G CPU @ 3.10 GHz’. This was possible since OpenCL provides a hardware-agnostic programming model, allowing code to run on a variety of CPU architectures as well as other types of accelerators such as GPUs. The performance of the parallel GPU acceleration was compared to the parallel CPU implementation. Further, a hybrid programming model approach was used for the stochastic optimization algorithms, Genetic Algorithm and Simulated Annealing, which were embedded in the prototype from *R* but within python using the python wrapper rpy2 which was used to interface the *R*-based stochastic optimization algorithms within the Python application development environment before the parallel execution strategy was adopted for the dynamic option pricing models.

3 Results and Discussion

3.1 Calibration Results of Stochastic Optimization Algorithms for the Dynamic Option Pricing Model

The data used in this paper was of daily closing prices for the S&P 500 Index European Call Options of maturity of 3 Years with expiry of 18th December 2019 and 85 Strikes price levels from 100 to 3400 USD. The cross-sectional calibration of the models was done at time-to-maturity of 3 years. The evaluation of the performance was performed on the remainder of the option data series. The interest rate of 1.8559% used in the evaluation was obtained as the aggregate of US Treasury Bill rates. Single parameter copula specifications were used in the calibration of the dynamic option pricing model.

We compared the results obtained from calibration using the simulated annealing and genetic algorithms. These results are summarized in Table 1 next. In Table 2, the parameters obtained from this optimization process of the dynamic option pricing model to market prices were used in the evaluation of the pricing performance. A hybrid programming model approach was used where the optimization algorithms were embedded in the system from *R* but within Python using the python wrapper rpy2 to interface the *R*-based stochastic optimization

algorithms within the Python application development environment.

Table 1 Dynamic Models Calibrations with Simulated Annealing and Genetic Algorithm

	Parameter	Gaussian	Clayton	Gumbel	Frank	Joe
Simulated Annealing	v_0	0.01000	0.01000	0.01000	0.01000	0.01000
	κ	0.000087	0.000045	0.000055	0.00015	0.000054
	α	0.01502	0.01500	0.01506	0.01578	0.01500
	δ	0.72331	0.20002	0.26039	0.20000	0.02000
	θ_1	-0.31947	0.20013	1.000002	5.00012	6.0000
		Value=10592.91 Counts=17196 Time=8.025 mins	Value=11318.53 Counts=10893 Time=12.3540 mins	Value=11119.4 Counts=10794 Time=7.066 mins	Value=13803.71 Counts=10365 Time=4.3414 mins	Value=41362.97 Counts=10629 Time=4.9399 mins
Genetic Algorithm	v_0	0.01000	0.01012	0.01025	0.01022	0.01069
	κ	0.00223	0.01425	0.02868	0.00086	0.05343
	α	0.01623	0.01509	0.01809	0.01568	0.01562
	δ	0.69081	0.20713	0.26218	0.21698	0.21342
	θ_1	-0.34499	0.21578	1.03997	5.05838	6.05391
		Value=-10609.82 Count=603 Time=20.3731 mins	Value=-11612.22 Counts=813 Time=24.5789 mins	Value=-11910.54 Counts=441 Time=16.3676 mins	Value=-14296.27 Counts=659 Time=22.8067 mins	Value=-47650.44 Counts=398 Time=14.9203 mins

Notes: Table 1 reports the optimal parameters obtained from the calibration of the dynamic option pricing model with single parameter copula specifications. The calibration was performed at 3 year time-to-maturity for ATM options. The optimal parameter values, iterations and obtained fitness function values of the genetic algorithm and simulated annealing algorithm are presented for the dynamic option pricing model under various copula specifications.

The optimal parameters obtained by the Genetic Algorithm and Simulated Annealing calibrations were then used in the pricing of ITM 3-Year S&P Index European Options. The results are shown in Table 2 next. The Dynamic Option Pricing Model was evaluated under these optimal parameters for ITM options. The RMSE obtained were then compared under various copula specifications (Gaussian, Clayton, Gumbel, Frank and Joe) with the RMSE values obtained by the Heston model. Under the optimal parameters obtained by the Genetic Algorithms, the performance improvement was found to be higher than under the optimal parameters obtained by Simulated Annealing for ITM options.

The Genetic Algorithm parameters had an aggregate of 25.16% performance improvement whereas the Simulated Annealing optimal parameters lead to an aggregate of 24.06% performance improvement across copula specification. The highest performance improvement was 34.04% under optimal Genetic Algorithm parameter for the Joe copula specification over the Heston Model. These optimal parameters were as follows $v_0 = 0.0107$, $\kappa = 0.0534$, $\alpha = 0.0156$, $\delta = 0.2134$ and $\theta = 6.0539$. The lowest performance improvement under optimal parameter was under the clayton copula specification at 9.34%. Generally, for ITM option pricing by use of optimal parameters obtained by the stochastic optimization algorithms yields a significant

performance improvement (Aggregates of 25.16% for GA and 24.06% for SA) over the Heston Model. Cross-copula performance variation is highest for the implementations using Genetic Algorithms optimal parameters at 0.0808 performance standard deviation and lower for the implementations using Simulated Annealing optimal parameters at 0.071548 performance standard deviation. The Genetic Algorithm optimal parameters yield the best performance for all copula specifications.

Table 2 Performance Evaluation by RMSE and RMSE Percentage Improvement of Dynamic Pricing Model under optimal parameters obtained by Genetic Algorithm and Simulated Annealing Algorithm

		Gaussian	Clayton	Gumbel	Frank	Joe	Heston
Genetic Algorithm	ITM	56.48	49.27	48.59	46.87	42.30	64.13
		11.92%	23.19%	24.25%	26.92%	34.04%	-
Simulated Annealing	ITM	56.50	66.14	47.41	45.32	42.75	64.13
		11.90%	9.34%	26.07%	29.33%	33.34%	-

Notes: Performance evaluation by RMSE of the Dynamic Option Pricing Model under optimal parameters by Genetic Algorithm and Simulated Annealing for ITM 3 Year S&P 500 Index call options. The performance improvement over the Heston under the various copula specifications was also computed and presented.

3.2 Parallel Implementation of Dynamic Option Pricing Model on GPU

In a bid to accelerate this process of evaluation, a heterogeneous GPU-based architecture was adopted. A variety of speed and pricing back-tests were performed within this architecture for ITM options across maturity levels and strike levels. In development of this system, the dynamic option pricing models were implemented in C-based kernel in OpenCL for GPU acceleration. The OpenCL device used was ‘Intel(R) HD Graphics 630’ on ‘Intel(R) OpenCL’ on the host ‘Intel(R) Core (TM) i7-8705G CPU @ 3.10 GHz’. The results were then compared to the efficiency within a parallel CPU-based implementation also on OpenCL using 4 cores and 8 threads. This system architecture allowed for efficient development of the application in multilanguage and multicore heterogeneous architectural paradigm that would ensure optimal optimization algorithm performance as well as pricing model performance.

The number of contracts and observations was increased from 85 to 10,000 representing 66,300 and 7,800,000 price observations respectively. The time-to-maturities in the evaluation was set to 3 years. These results are presented in Table 3 which compares the speeds of the parallel GPU implementation and the parallel CPU-based implementation.

From the GPU implementation results of the dynamic option pricing model, we find that the GPU implementation performance is at least 163 times faster than the parallel implementation on the CPU. As the number of observations and the number of contracts under observation is increased from 85 contracts to 1000 contracts, the GPU implementation performance peaks at 541 times faster than that of the parallel implementation on the CPU, this reduces the implementation time from 46.24 minutes to 5.12 seconds. When with a maximum number of contracts under evaluation being set to 10,000 contracts the implementation time on the GPU

Table 3 Evaluation of Parallel Implementation on GPU and on CPU of the Dynamic Option Pricing Model

Number of Contracts	Obs	Implementation with Time-to-maturity of 3 Years	Dynamic Option Pricing Model Computational Speed	Acceleration factor on GPU vs. CPU
85	66,300	Parallel on CPU	3.7811 mins	-
		Parallel on GPU	1.3905 secs	x163
100	78,000	Parallel on CPU	4.9540 mins	-
		Parallel on GPU	1.7919 secs	x166
300	234,000	Parallel on CPU	6.6807 mins	-
		Parallel on GPU	2.2205 secs	x181
500	390,000	Parallel on CPU	22.908970 mins	-
		Parallel on GPU	3.19855 secs	x431
1000	780,000	Parallel on CPU	46.25455 mins	-
		Parallel on GPU	5.13269 secs	x541
2000	1,560,000	Parallel on CPU	1.12187 hours	-
		Parallel on GPU	9.41050 secs	x428
5000	3,900,000	Parallel on CPU	1.7510 hours	-
		Parallel on GPU	22.7924 secs	x277
10000	7,800,000	Parallel on CPU	5.97684 hours	-
		Parallel on GPU	85.17690 secs	x252

Notes: Dynamic Option Pricing Model Parallel Implementation on GPU vs Parallel Implementation on CPU Performance Comparison on European Options with 3 Year time-to-maturity. The aggregate performance of the parallel implementation on GPU was 273 times faster across number of contracts. The number of options contracts analyzed was increased from 85 to 10,000 throughout this increment of number of contracts the speed of parallel implementation on GPU increase to a maximum of 541 time faster for 1000 contracts where the acceleration factor decreased steadily albeit still being high at 252 times faster for 10,000 option contracts.

becomes 85.17 seconds from 5.97 hours. This indicates a 252 times faster evaluation speed for the GPU based implementation over the parallel approach. The aggregate performance of the parallel implementation on GPU was 273 times faster across number of contracts. As the number of option contracts under implementation is increased from 85 contracts to 1000 contracts the computation speeds are reduced from 3.78 minutes to 1.39 seconds and from 46.24 minutes to 5.13 seconds, this represents an acceleration of 163 times and 541 respectively. As the number of contracts are increased from 2,000 through to 10,000, the acceleration decreases gradually albeit still being high from 1.12 hours to 9.40 seconds and 5.97 hours to 85.17 seconds representing a computational speed acceleration 428 times and 252 times respectively.

An optimal peak in acceleration performance of 541 times is witnessed, and then a slow decline is seen in the acceleration as the number of contracts and observations are increased.

This could have resulted from the overhead of kernel launches as the parallelism is increased such that the number of launches in the compute GPU processing units also grow. This can also be explained by the fact that as the degree of parallelism increases, the available computing resources such as the CPU cores, GPU compute units and memory bandwidth may have become saturated, leading to diminishing returns and suboptimal acceleration.

4 Conclusion

This experimental evaluation of the implementation of the dynamic option pricing model on the GPU shows that the approach is efficient and scalable, which achieves almost a linear speedup across number of option contracts and observations. There were no differences in accuracy of the evaluation between the GPU implementation and those achieved by the parallel CPU approach. The highly parallel structures of GPUs make them more effective than the traditional CPU for the compute intensive segments of the application. These results favour considerably the use of GPU in acceleration of financial applications more so when model throughput and speed are desired for novel complex pricing models.

Calibration of the model was done by the Genetic Algorithm and Simulated Annealing and the optimal parameters obtained. The parameters were then used in the pricing of ITM 3-Year S&P 500 Index European call options. Under the optimal parameters obtained by the Genetic Algorithms, the performance improvement was found to be higher than under the optimal parameters obtained by Simulated Annealing for ITM options. The Genetic Algorithm parameters had an aggregate of 25.16% performance improvement whereas the Simulated Annealing optimal parameters lead to an aggregate of 24.06% performance improvement across copula specification over the Heston model. The highest performance improvement was 34.04% under optimal Genetic Algorithm parameter for the Joe copula specification over the Heston model.

The dynamic option pricing model was implemented on GPU using OpenCL where the pricing kernel function was offloaded to the GPU. The number of 85 contracts was gradually increased to 10,000 contracts representing 66,300 and 7,800,000 price observations with the time-to-maturity in the evaluation being 3 years. The parallel GPU implementation was at least 163 faster than the parallel CPU implementation. As the number of observations and the number of contracts under observation is increased from 85 contracts to 1000 contracts, the GPU implementation performance peaks at 541 times faster than that of the parallel implementation on the CPU, this reduces the implementation time from 46.24 minutes to 5.12 seconds. When with a maximum number of contracts under evaluation is set to 10,000 contracts the implementation time on the GPU becomes 85.17 seconds from 5.97 hours. This indicates a 252 times faster evaluation speed for the GPU based implementation over the CPU approach.

The highly parallel structure makes GPUs more effective than traditional CPUs in compute intensive and highly parallel applications. An increasing number of financial problems could be efficiently and accurately solved with GPU acceleration. A natural extension of the parallel GPU implementation would be a consideration of architectures that are as heterogeneous as possible with distributed infrastructures composed of CPU/GPU and Clouds, and since OpenCL enables use of a diverse range of accelerators including multi-core CPUs, GPUs, DSPs, FPGAs and dedicated hardware, this distributed infrastructure could be adopted. The

implementation on GPU architectures has the benefit of being less costly, but limited in terms of memory since the GPU does not offer much memory as the CPU. The efficient utilization of GPUs and alternative high-performance architectures and techniques in the pricing of financial derivatives using various numerical algorithms presents a research area worth further exploration whose benefits would be incredible in the pricing, hedging, trading and analytical and energy efficiencies and cost savings for the relevant financial institutions and their divisions.

References

- [1] Basermann A, Kohring G A, Neff C. Derivative pricing as a business grid application using NextGRID technology. *WIT Transactions on Information and Communication Technologies*, 2008, 41: 53–62.
- [2] Harris M. Grid computing adoption in the financial services sector. <https://www.theglobaltreasurer.com/2008/03/11/grid-computing-adoption-in-the-financial-services-sector> [2023-01-12].
- [3] Lee J. Access to finance for artificial intelligence regulation in the financial services industry. *European Business Organization Law Review*, 2020, 21(4): 731–757.
- [4] Surkov V. Parallel option pricing with Fourier space time-stepping method on graphics processing units. *Parallel Computing*, 2010, 36(7): 372–380.
- [5] Zeng J, Ling L, Zeng J, et al. Dynamic pricing model of service-oriented educational products based on revenue management. *Mathematical Problems in Engineering*, 2022.
- [6] Chorafas D N. Risk management technology in financial services: Risk control, stress testing, models, and IT systems and structures. Elsevier, 2011.
- [7] Kanninen J, Koskinen M. Distributed calibration of option pricing models with multiple contracts. Working Paper, Tampere University of Technology, Finland, 2017.
- [8] Fatone F L, Marco G, Francesca M, et al. Parallel option pricing on GPU: Barrier options and realized variance options. *The Journal of Supercomputing*, 2012, 62(3): 1480–1501.
- [9] Dou W, Shoushuai M. Performance evaluation of parallel re-computing algorithm in different data distribution modes. *Journal of Algorithms & Computational Technolog*, 2018, 12(1): 43–52.
- [10] Dai B, Ying P, Bin G. Parallel option pricing with BSDE method on GPU. 2010 Ninth International Conference on Grid and Cloud Computing. IEEE, 2010: 191–195.
- [11] Ma H. Development of a CPU-GPU heterogeneous platform based on a nonlinear parallel algorithm. *Nonlinear Engineering*, 2022, 11(1): 215–222.
- [12] Trading Risk. Secondary Cat Bond Pricing Falls 2% in Q1 - Aon Benfield Securities. London, 2012.
- [13] NVIDIA. NVIDIA Tesla GPUs Used by J.P. Morgan Run Risk Calculations in Minutes not Hours. New York, 2014.
- [14] Ganesan N, Roger C D, Jeremy B. Acceleration of binomial options pricing via parallelizing along time-axis on a GPU. *Proceedings of Symposium on Application Accelerators in High Performance Computing*, 2009.
- [15] Solomon S, Ruppa T K, Parimala T. Option pricing on the GPU. 2010 IEEE 12th International Conference on High Performance Computing and Communications (HPCC), 2010: 289–296.
- [16] Zhang N, Chi-Un L, Ka L M. Binomial American option pricing on CPU-GPU heterogenous system. *Engineering Letters*, 2012, 20(3): 279–285.
- [17] Suo S, Ruiming Z, Ryan A, et al. GPU option pricing. *Proceedings of the 8th Workshop on High Performance Computational Finance*, 2015: 1–6.
- [18] Michael B, Françoise B. Towards parallel and distributed computing on GPU for American basket option pricing. 4th IEEE International Conference on Cloud Computing Technology and Science Proceedings, IEEE, 2012: 723–728.
- [19] Podlozhnyuk V, Mark H. Monte Carlo option pricing. NVIDIA, 2013.
- [20] Abbas-Turki L A, Bernard L. American options pricing on multi-core graphic cards. 2009 International Conference on Business Intelligence and Financial Engineering, 2009: 307–311.
- [21] Trainor S, Crookes D. GPU acceleration of a basket option pricing engine. In *World Congress on Engineering*, 2013.
- [22] Grauer-Gray S, William K, Robert S. Accelerating financial applications on the GPU. *Proceedings of the 6th Workshop on General Purpose Processor Using Graphics Processing Units*, 2013: 127–136.

- [23] Yu J, Hiroki S, Ryusuke M, et al. GPU implementation of efficient pedestrian detection based on MCMC. SCIS & ISIS, 2010: 1624–1629.
- [24] Dang D M. Modeling multi-factor financial derivatives by a Partial Differential Equation approach with efficient implementation on Graphics Processing Units. Doctoral Dissertation, University of Toronto, 2012.
- [25] Dang D M, Christina C, Jackson K R. An efficient graphics processing unit-based parallel algorithm for pricing multi-asset American options. *Concurrency and Computation: Practice and Experience*, 2012, 24(8): 849–866.
- [26] Zhang B, Oosterlee C W. Option pricing with cos method on graphics processing units. 2009 IEEE International Symposium on Parallel & Distributed Processing, IEEE, 2009: 1–8.
- [27] Mrázek M, Jan P. Calibration and simulation of Heston model. *Open Mathematics*, 2017, 15(1): 679–704.
- [28] Muganda B W, Kyriakou I, Kasamani B K. Modelling asymmetric stochastic volatility and option pricing: A conditional copula approach. *Scientific African*, 2023.
- [29] Gilli M, Dietmar M, Enrico S. Numerical methods and optimization in finance. Academic Press, 2019.
- [30] Bruintjes T M. Design of a fused multiply-add floating-point and integer datapath. Master's Thesis, University of Twente, 2011.
- [31] Zhang Y, Lu P, Bin L, et al. Architecture comparisons between Nvidia and ATI GPUs: Computation parallelism and data communications. 2011 IEEE International Symposium on Workload Characterization (IISWC), IEEE, 2011: 205–215.
- [32] Forsell M, Sara N, Jussi R, et al. Performance and programmability comparison of the thick control flow architecture and current multicore processors. *The Journal of Supercomputing*, 2022, 78(3): 3152–3183.
- [33] Di Pierro M. Portable parallel programs with Python and OpenCL. *Computing in Science & Engineering*, 2014, 16(1): 34–40.
- [34] Gorobets A, Pavel B. Heterogeneous CPU+ GPU parallelization for high-accuracy scale-resolving simulations of compressible turbulent flows on hybrid supercomputers. *Computer Physics Communications*, 2022 February, 271: 108231.