International Journal of Applied Sciences and Smart Technologies

Volume 2, Issue 1, pages 9–22

p-ISSN 2655-8564, e-ISSN 2685-9432

Pricing the Financial Heston Model Using Parallel Finite Difference Method on GPU CUDA

Pranowo

Department of Informatics Engineering, Faculty of Industrial Technology, Universitas Atma Jaya Yogyakarta, Indonesia
Corresponding Author: pranowo@uajy.ac.id

(Received 29-04-2019; Revised 17-10-2019; Accepted 17-10-2019)

Abstract

An option is a financial instrument in which two parties agree to exchange assets at a price or strike and the date or maturity is predetermined. Options can provide investors with information to set strategies so they can increase profits and reduce risk. Option prices need to be accurately evaluated according to reality and quickly so that the resulting value can be utilized at the best momentum. Valuation of option prices can use the Heston equation model which has advantages compared to other equation models because the assumption of volatility is not constant with time or stochastic volatility. The volatility that is not constant with time corresponds to reality because the underlying asset as a basis can experience fluctuations. The Heston equation has a disadvantage because it is a derivative equation that is difficult to solve. One way to solve derivative equations easily is to use a numerical solution to the finite difference method of non-uniform grids because the Heston equation can be assumed to be a parabolic equation. The numerical solution of the finite difference method can solve derivative equations flexibly and do not require matrix processing. But it requires a heavy and slow computing process because there are many elements of calculation and iteration. This study proposes a numerical solution to the finite difference method by using the Compute Unified Device Architecture (CUDA) parallel programming to solve the Heston equation model that applies the concept of stochastic volatility to get accurate and fast results. The results of this research proved 15.52 times faster in conducting parallel computing processes with error of 0.0016..

Keywords: option price, heston model, finite difference, parallel, GPU CUDA.

1 Introduction

Options provide investors with information to set strategies so they can increase profits and reduce risk. Valuation of option prices can be assessed using the Heston equation model that applies stochastic volatility, which means that something is determined randomly and may not be accurately predicted.

Research related to numerical solution of option price using Heston model been done previously which can be seen in Table 1. Researchers indicate that previous research was limited to solving Heston equations using numerical solutions for option prices and had not been implemented in parallel computing so that this study discussed the numerical solutions of finite methods difference non-uniform grids to solve Heston equations in parallel computing.

The computational process of the finite difference method of non-uniform grids will increase as the number of grids increases. Heavy computing process is an obstacle in using many grids to improve the accuracy of results. At first, the computer had only one Central Processing Unit (CPU) or called the uniprocessor architecture for computational processing. Today computers evolve into multicore architectures that support processing in parallel. Parallel processing can be done with parallel programming, namely programming that focuses on solving problems simultaneously using fully using the computational power of computer architecture [1]. These problems can be solved by computational processing using parallel programming that utilizes the Graphics Processor Unit (GPU) with the Compute Unified Device Architecture (CUDA)

Programming Model. GPU consists of a set of CPU’s that perform computational processes in parallel so that it can work on many computational processes simultaneously. CUDA Programming Model is an application programming model that utilizes GPU as the core computational process. The solution to the numerical problem of derivative equations in the Heston model using the finite difference method that utilizes CUDA Programming Model-based parallel programming is expected to determine accurate option values with fast computational performance.

Table 1. Previous research

Research	Purpose
Diamond–Cell Finite Volume Scheme for the Heston Model [7]	Propose a new numerical scheme to solve partial equations that appear in the Heston stochastic volatility model.
Stability of central finite difference schemes for the Heston PDE [8]	Measuring stability limits is useful for time discretization methods in numerical solutions of Heston partial differential equations that stand out from mathematical finance.
Pricing European Options with Proportional Transaction Costs and Stochastic Volatility Using a Penalty Approach and a Finite Volume Scheme [9]	Establishing European standard option pricing values based on proportional and stochastic volatility transaction cost using the penalty approach method and finite volume scheme.
Numerical methods to solve PDE models for pricing business companies in different regimes and implementation in GPUs [10]	Solving the problem of corporate valuation models using a numerical approach to the finite difference method developed with parallelization using GPU technology.
Pricing of early-exercise Asian options under Lévy processes based on Fourier cosine expansions [11]	Set prices for Asian options with initial training features based on two-dimensional integration and backward recursion from Fourier coefficients in several numerical techniques implemented on the GPU.

2 Theory

2.1. Option Price

An option is a financial instrument in which two parties agree to exchange assets at a price or strike and the date or maturity is predetermined [2]. By paying in advance, known as price or premium from options, contract holders have the right, but not the obligation, to buy or sell assets at the time of maturity [3]. For example, the European option model has rules that can only be exercised at maturity.

The value of the option is based on the derivative value of the underlying asset, so the option is derivative. Based on this, the option contract is one of "derivative security" [4]. The underlying asset value has a property proportional to the value of the call option and the property is inversely proportional to the value of puts option. The value of up option calls if the value of the underlying asset rises and vice versa. The value puts down if the underlying asset rises and vice versa.

2.2. *Finite Difference for Heston PDE*

The finite difference method has the idea of discretizing domains with several grid points and using the finite difference to estimate derivatives at these grids [5]. The Finite Difference method assumes that the model grids can be structured or unstructured. The finite difference method is a technique to get numerical estimates from PDE.

To be able to implement finite difference to solve Heston PDE, it is necessary to discretize grids for the stock price and variance variables and discretize grids for maturity. This research uses non-uniform grids to discretize grids. Non-uniform grids have irregular grid distances between the two variables used. Non-uniform grids can be refined at certain points so that accurate price valuations can be produced with accurate prices using a few grid points.

The variables used to form grids are S, v, and t. It is necessary to determine the maximum value and the minimum value of S, v, and t as the value limit. The maximum value is denoted as S_{max}, v_{max}, and t_{max}. The values of S_{max} and $V_{max}$are obtained based on the calculated option case, while $t_{max} = \tau$ based on the maturity time. The minimum value is denoted as S_{min}, v_{min}, and t_{min}. The minimum value will always be set to $S_{min} = v_{min} = t_{min} = 0$ as the lower limit [6].

The grid size is set with $N_s + 1$ point for the stock price, $N_v + 1$ point for volatility, and $N_T + 1$ point for maturity. The width of non-uniform grids for $N_s + 1$ stock price is arranged by equation

$$S_i = K + csinh(\xi_i), i = 0,1, \dots, N_s \tag{1}$$

The width of non-uniform grids for $N_v + 1$ volatility is arranged by equation

$$v\_i = dsinh(j\Delta\eta), j = 0,1, \dots, N\_V \tag{2}$$

The width of non-uniform grids for $N_t + 1$ volatility is arranged by equation

$$t_i = i \times dt, n = 0,1, \dots, N_T \tag{3}$$

This Heston PDE model estimates the point values in the interior and boundary sections separately. The interior part (S_i, v_i) is estimated by using first-order derivatives with a central difference. The boundary section is governed by certain conditions.

The boundary section has several conditions that need to be initialized, i.e. the conditions at maturity, S $= S_{min}$ S $= S_{max}$, V $= V_{max}$, and V $= V_{min}$.

The boundary conditions at maturity, t $= 0$, the value of the call option is the intrinsic value (payoff) so that the equation is obtained.

$$U(S_i, v_j, 0) = max(0, S_i - K) \tag{4}$$

with a limit $i = 0, 1, \dots, N_s$ and $j = 0, 1, \dots, N_v$.

The boundary condition when S $= S_{min} = 0$, the call option becomes useless. Because that equation is obtained

$$\mathrm{U}(0, v_j, t_n) = 0 \tag{5}$$

with a limit of $n = 0, 1, \dots, N_T$ and $j = 0, 1, \dots, N_v$.

The boundary condition when S $= S_{max}$, the equation used is

$$U(S_{max}, v_j, t_n) = S_{max} \tag{6}$$

with a limit of $n = 0, 1, \dots, N_T$ and $j = 0, 1, \dots, N_v$.

The boundary condition when V $= V_{max}$, the equation used is

$$U(S_i, v_{max}, t_n) = S \tag{7}$$

with a limit of $n = 0, 1, \dots, N_T$ and $i = 0, 1, \dots, N_s$.

The boundary conditions when $V = V_{min} = 0$, the equation used is 2, namely $\frac{\partial U}{\partial S}$ which is solved using the central difference and $\frac{\partial U}{\partial v}$ which are resolved using forward difference. The equation formed is

$$\frac{\partial U}{\partial S}(S_i, 0, t_n) = \frac{U^n_{i+1,0} - U^n_{i-1,0}}{S_{i+1} - S_{i-1}}, \qquad \frac{\partial U}{\partial v}(S_i, 0, t_n) = \frac{U^n_{i,1} - U^n_{i,0}}{v_1} \tag{8}$$

Explicit schemes will be used as a technique to solve the finite difference. The equation used to obtain the elements $U^{n+1}_{i,j}$ is

$$U^{n+1}_{i,j} = U^n_{i,j} + dt\left[\frac{1}{2}v_j S_i^2 \frac{\partial^2}{\partial S^2} + \frac{1}{2}\sigma^2 v_j \frac{\partial^2}{\partial V^2} + \rho\sigma v_j S_i \frac{\partial^2}{\partial v \partial S} + (r-q)S_i \frac{\partial}{\partial S} + \kappa(\theta - v_j)\frac{\partial}{\partial v} - r\right] U^n_{i,j} \tag{9}$$

2.3. *Compute Unified Device Architecture (CUDA) Programming Structure*

The CUDA programming model can execute applications on heterogeneous computing systems by only annotating code with a set of extensions to the C programming language. NVIDIA can be used to allocate the right host memory (CPU) and device (GPU) so that applications can be optimized and maximize the use of hardware [1]. The structure of the CUDA application process can be seen in Figure 1.

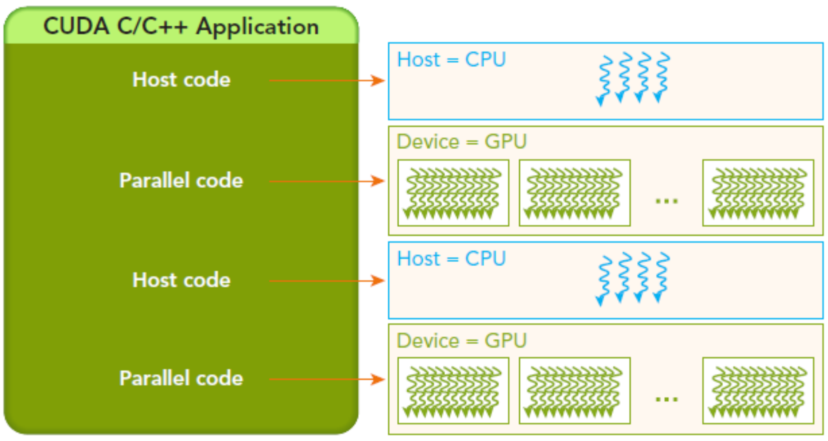


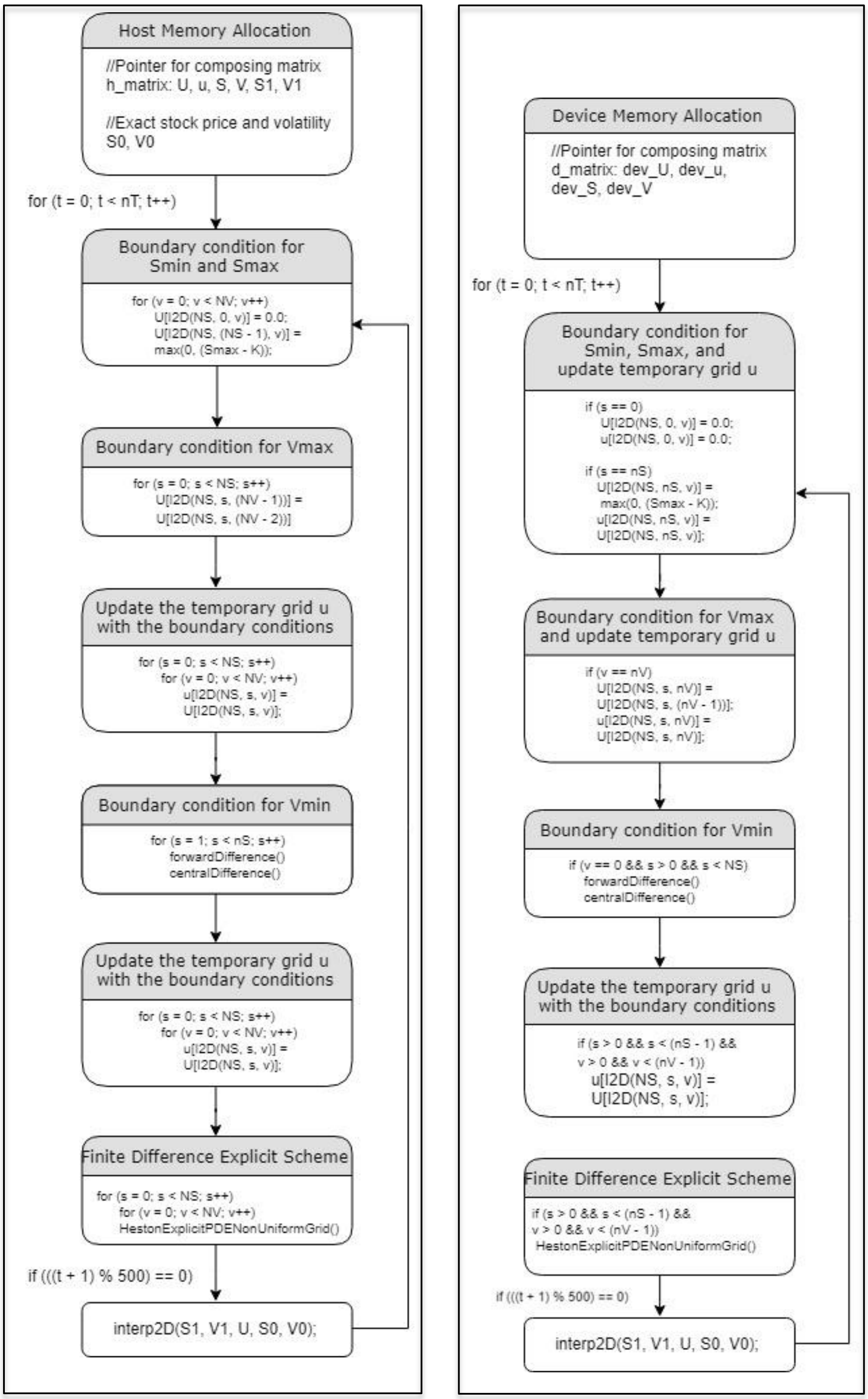
Figure 1. CUDA programming structure

CUDA which consists of serial code is run on the host, while parallel code is run on the GPU device. Host code is written in ANSI C and Device code is written using CUDA C. All code can be placed in a single source file or can use multiple source files to build the desired application or library. Codes that have been created for hosts and devices can be run using NVIDIA C Compiler (NVCC).

3 Algorithm

Parallel programming is a programming algorithm that forms a program that is capable of working on several processes in parallel utilizing multiple processors. In programming, CUDA uses SIMT (Single Instruction, Multiple Threads) execution models that are similar to SIMD (Single Instruction, Multiple Data) execution models for general data parallel programming [10]. The CUDA code execution unit, the kernel, executes simultaneously a set of threads in each block freely. Each thread will run one processor simultaneously on the same but different data instructions. Figure 2 describes the CPU and GPU algorithms.

Based on the flowchart above, it can be seen that the GPU algorithm can simplify the CPU algorithm so that it is not complex, where simple processes such as temporary grid updates, u, can be done simultaneously with boundary initialization. Therefore the GPU algorithm is simpler and not much repetitive. Repetition is only done to do time iterations. The 2-D matrix used is changed to $1-D$, because GPUs have different matrix index concepts. The CPU index is a row of x columns, $U\ [i \dots Ns]\ [i \dots Nv]$, and an index on the GPU in the form of columns x rows, $U\ [v \dots Nv]\ [i \dots Ns]$. GPU uses column x row index because it adjusts the hierarchy of blocks and threads. Changing the index to $1-D$ makes the matrix index can be adjusted according to the indexing formula $((ni\ *\ 1)\ +1)$ in order to meet the concept of matrix CPU and GPU. In addition, memory allocation is only done in the matrix pointer U, u, S, V to allocate memory pointers on the device, GPU and copy values from the host, CPU to device, GPU. Another parameter that is not a pointer can be used directly across hosts and devices.


Host Memory Allocation
Boundary condition for Smin and Smax
Boundary condition for Vmax
Update the temporary grid u with the boundary conditions
Boundary condition for Vmin
Update the temporary grid u with the boundary conditions
Finite Difference Explicit Scheme
Device Memory Allocation
Boundary condition for Smin, Smax, and update temporary grid u
Boundary condition for Vmax and update temporary grid u
Boundary condition for Vmin
Update the temporary grid u with the boundary conditions
Finite Difference Explicit Scheme

```
//Pointer for composing matrix
h_matrix: U, u, S, V, S1, V1

//Exact stock price and volatility
S0, V0

for (t = 0; t < nT; t++)

for (v = 0; v < NV; v++)
    U[I2D(NS, 0, v)] = 0.0;
    U[I2D(NS, (NS - 1), v)] =
    max(0, (Smax - K));

for (s = 0; s < NS; s++)
    U[I2D(NS, s, (NV - 1))] =
    U[I2D(NS, s, (NV - 2))]

for (s = 0; s < NS; s++)
    for (v = 0; v < NV; v++)
        u[I2D(NS, s, v)] =
        U[I2D(NS, s, v)];

for (s = 1; s < nS; s++)
    forwardDifference()
    centralDifference()

for (s = 0; s < NS; s++)
    for (v = 0; v < NV; v++)
        u[I2D(NS, s, v)] =
        U[I2D(NS, s, v)];

for (s = 0; s < NS; s++)
    for (v = 0; v < NV; v++)
    HestonExplicitPDENonUniformGrid()

if (((t + 1) % 500) == 0)

interp2D(S1, V1, U, S0, V0);
```

```
//Pointer for composing matrix
d_matrix: dev_U, dev_u,
dev_S, dev_V

for (t = 0; t < nT; t++)

if (s == 0)
    U[I2D(NS, 0, v)] = 0.0;
    u[I2D(NS, 0, v)] = 0.0;

if (s == nS)
    U[I2D(NS, nS, v)] =
    max(0, (Smax - K));
    u[I2D(NS, nS, v)] =
    U[I2D(NS, nS, v)];

if (v == nV)
    U[I2D(NS, s, nV)] =
    U[I2D(NS, s, (nV - 1))];
    u[I2D(NS, s, nV)] =
    U[I2D(NS, s, nV)];

if (v == 0 && s > 0 && s < NS)
    forwardDifference()
    centralDifference()

if (s > 0 && s < (nS - 1) &&
v > 0 && v < (nV - 1))
    u[I2D(NS, s, v)] =
    U[I2D(NS, s, v)];

if (s > 0 && s < (nS - 1) &&
v > 0 && v < (nV - 1))
 HestonExplicitPDENonUniformGrid()

if (((t + 1) % 500) == 0)

interp2D(S1, V1, U, S0, V0);
```

Figure 2. Flowchart of CPU (left) and GPU (right) algorithms

Finite difference non-uniform grids numerical solutions require complex and many computational processes, so that an increase in the number of relevant grids can be used to measure computational performance. This study conducted a numerical experiment by increasing the number of phased grids to see the difference between the performance of the GPU algorithm and the CPU algorithm. Experiments have been carried out on stand-alone computers with Intel Core i7 $8700K$ which has 6 cores with 3.7 GHz clock, 16 GB RAM, and Nvidia Geforce GTX 1080 Ti GPU which has 3584 processors and 11117 MB GDDR5X. The CUDA version installed is 9.0.

4 Results and Discussions

The implementation of the GPU algorithm to solve the equations of the Heston model using the finite difference method non-uniform grids needs to be verified. Verification is done by conducting numerical experiments to see the convergence of numerical finite difference non-uniform grids with exact values, along with the increase in the number of grid points for stock prices and volatility. The parameter used for this numerical experiment is $S_{max} = 200$; $V_{max} = 0.5$; $T_{max} = 0.15$; $K = 100$; $r = 0.02$; $q = 0.5$; $\sigma = 0.3$ and $\rho = -0.9$.

The combination of the number of grid points for the stock price and the volatility used varies. The size of the grids is formed by following the condition that finer grids approach the strike price K and around the point $v_0 = 0$. The number of grids for the stock price, N_s, has a range of values from 80 to 190, with 10 increases. The number of grids for the stock price, N_v, has a range of values from 20 to 75, with 5 increases. This numerical experiment will be an iterated as much as $N_t = 3000$ as the number of time points. The maximum grid combination in this experiment is $N_s = 190$ and $N_v = 75$. Non-uniforms grids can be seen in Figure 3**.**

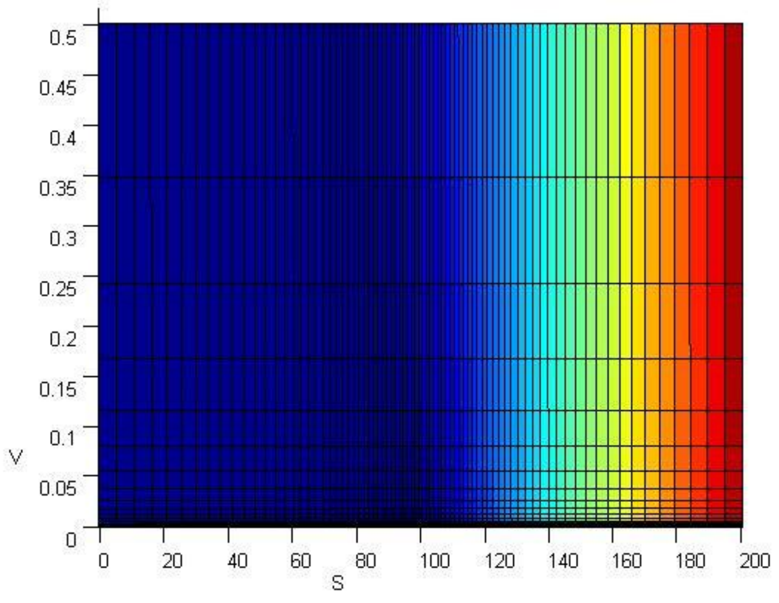


Figure 3. Non-uniform grids

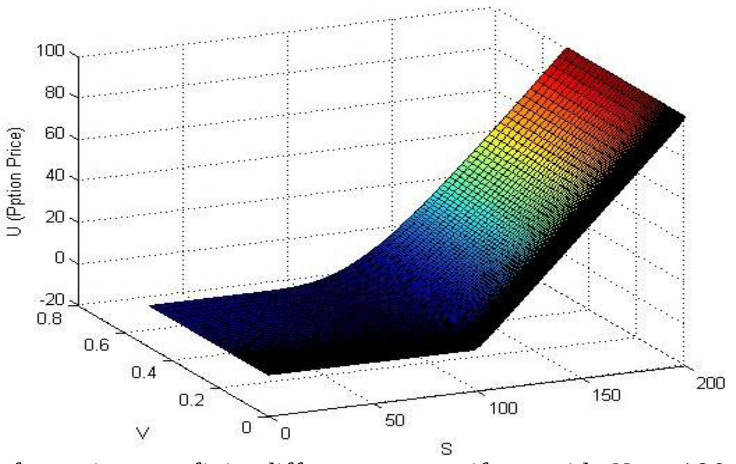


Figure 4. Surface prices use finite difference non-uniform grids $N_s = 190$ and $N_v = 75$

The exact value of the stock price, $S = 102$ and volatility, $v = 0.05$ with the exact value of the option price using the closed form solution of the Heston model is 4.2783. The results of finite difference non-uniform grids with a combination of up to grids $N_s = 190$ and $N_v = 75$ can be seen in Table 2 resulting in a value of 4.2801 using the GPU algorithm and 4.2805 using the CPU algorithm. Experiments were also carried out by increasing grid N_s and N_v to reach the limit before instability was achieved. The results obtained show grids $N_s = 190$ and $N_v = 150$ are the limits before instability is achieved.

Table 2. Relative error numerical finite difference non-uniform grids CPU and GPU solutions

N_s	N_v	CPU		GPU	
		Price	**Error**	**Price**	**Error**
80	20	4.2767	-0.0016	4.2760	-0.0023
90	25	4.2868	0.0085	4.2864	0.0081
100	30	4.2811	0.0028	4.2807	0.0024
110	35	4.2797	0.0014	4.2792	0.0009
120	40	4.2814	0.0031	4.2810	0.0027
130	45	4.2808	0.0025	4.2804	0.0021
140	50	4.2812	0.0029	4.2808	0.0025
150	55	4.2819	0.0036	4.2814	0.0031
160	60	4.2796	0.0013	4.2792	0.0009
170	65	4.2800	0.0017	4.2796	0.0013
180	70	4.2813	0.0030	4.2809	0.0026
190	75	4.2805	0.0022	4.2801	0.0018
190	150	4.2705	-0.0078	4.2799	0.0016

The error is obtained by calculating the difference in the option price of the numerical result with the exact value. The error in the experimental results has a variety of values, where at each increase in the number of grids, the error does not always decrease. If we look further, the overall error continues to decrease as grid size increases. In the maximum grid combination $N_s = 190$ and $N_v = 150$, the smallest is obtained obtained at 0.0016. So that it can be ascertained that increasing the number of grid points will increase accuracy. Table 3 shows the GPU algorithm can produce values that are closer to the exact values and are more accurate when grids are enlarged. Enlargement grids also run faster when processed using the GPU, compared to when processed using CPU results per exact price, CPU numeric, and numerical GPU. The price-end-result using non-uniform finite difference grids $N_s = 190$ and $N_v = 75$ after an iteration of $N_t = 3000$ is shown in Figure 4.

Comparison of GPU algorithm performance with CPU algorithm was done by conducting numerical experiments by increasing the number of phased grids can be seen in Table 3.

Tabel 3. Performance: CPU vs GPU with 3000 time steps (N_t) and various grids

N_s	N_v	Grid points	CPU Time (s)	GPU Time (s)	Speedup (times)	Speedup (s)
80	20	1600	0.243	0.176	1.38X	0.067
90	25	2250	0.348	0.171	2.04X	0.177
100	30	3000	0.467	0.172	2.72X	0.295
110	35	3850	0.607	0.171	3.55X	0.436
120	40	4800	0.766	0.179	4.28X	0.587
130	45	5850	0.921	0.179	5.15X	0.742
140	50	7000	1.111	0.181	6.14X	0.93
150	55	8250	1.321	0.176	7.51X	1.145
160	60	9600	1.547	0.181	8.55X	1.366
170	65	11050	1.793	0.186	9.64X	1.607
180	70	12600	2.033	0.182	11.17X	1.851
190	75	14250	2.323	0.193	12.04X	2.13
190	150	28500	4.765	0.307	15.52X	4.458

Based on the experimental results, stable GPU performance is always superior to the CPU. In finer grids as they approach the K strike price and around the point $v_0 = 0$, the grid sizes of N_s and N_v are increases $N_s + 10$ and $N_v + 5$ gradually, GPU performance continues to increase $12.04X$ faster. Experiments were also carried out by increasing N_v grids to reach the limit before instability was achieved. The results are obtained on the grids $N_s = 190$ and $N_v = 150$ where computing performance reaches $15.52X$ faster. The bigger the grid, the CPU performance will decrease while the GPU performance is stable.

5 Conclusions

This study aims to solve the equations of the Heston model using numerical solutions with finite difference non-uniform grids based on the Compute Unified Device Architecture (CUDA) parallel programming to get accurate and fast results. Based on this research, finite difference non-uniform grids with GPU algorithms can produce values that approach exact values and are more accurate when grids are enlarged.

The error in the experimental results continues to decrease every time N_s is increased by 10 points, and N_v is increased by 5 points. The results of the finite difference non-uniform grids numerical solution with a maximum combination of grids $N_s = 190$ and $N_v = 75$ produce a value of 4.2805 with an error of 0.0018, compared to the combination value of internal grids $N_s = 80$ and $N_v = 20$ that produces a value of 4.2760 with an error of -0.0023. In the stability experiment with a combination of grids $N_s = 190$ and $N_v = 150$, the error obtained descreases to 0.0016. This proves that increasing the number of grid points will increase accuracy.

Enlargement grids also run faster when processed with the GPU. The computational process is faster 1.38 times in the combination of initial grids $N_s = 80$ and $N_v = 20$ and continues to increase until it has an acceleration of $12.04X$ faster on the grid $N_s = 190$ and $N_v = 75$. In the stability experiment with a combination of grids $N_s = 190$ and $N_v = 150$, computing performance reaches $15.52X$ faster.

Acknowledgements

This research was supported in part by Universitas Atma Jaya Yogyakarta (UAJY). This research was conducted using the GPU computers in the Graduate Program of Informatics Engineering UAJY.

References

[1] P. Kutik and K. Mikula, “Diamond–cell finite volume scheme for the heston model,” *Discrete & Continuous Dynamical Systems*, **8** (5), 913−931, 2015.

[2] K. J. in’t Hout and K. Volders, “Stability of central finite difference schemes for the heston PDE,” *Numerical Algorithms*, **60** (1), 115–133, 2012.

[3] W. Li and S. Wang , “Pricing european options with proportional transaction costs”, *Computers and Mathematics with Applications*,**73** (11), 2454−2469, 2017.

[4] D. Castillo, A. M. Ferreiro, J. A. García-Rodríguez, and C. Vázquez,” Numerical methods to solve PDE models for pricing business companies in different regimes and implementation in GPUs,” *Applied Mathematics and Computation*, **219** (24), 11233−11257, 2013.

[5] B. Zhang and C. W. Oosterlee, “Pricing of early-exercise Asian options under Lévy processes based on Fourier cosine expansions,” *Applied Numerical Mathematics*, **78**, 14−30, 2014.

[6] J. Cheng, M. Grossman, and T. McKercher, *Professional CUDA C Programming,* John Wiley & Sons, Indianapolis, 2014.

[7] E. Tandelilin, *Portofolio dan Investasi: Teori dan Aplikasi,* Kanisius, Yogyakarta, 2010.

[8] G. I. Ramírez-Espinoza, “Conservative and finite volume methods for the pricing problem,” *Master Thesis,* Faculty of Mathematics and Natural Science, Bergische Universität Wuppertal, Wuppertal, 2011.

[9] T. F. Crack, *Basic Black-Scholes: Option Pricing and Trading,* 2009.

[10] Y. Chen, “Numerical Methods for Pricing Multi-Asset Options,” *Master Thesis,* Graduate Department of Computer Science, University of Toronto, Toronto, 2017.

[11] F. D. Rouah, *The Heston Model and Its Extensions in Matlab and C#,* John Wiley & Sons, Hoboken, 2013.