CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2014; **26**:1626–1639
Published online 6 February 2012 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cpe.2825

SPECIAL ISSUE PAPER

Acceleration of option pricing technique on graphics processing units

Bowen Zhang [1,*,†] and Cornelis W. Oosterlee [1,2]

[1] *Delft University of Technology, Delft Institute of Applied Mathematics, Delft, The Netherlands*
[2] *CWI, Centrum Wiskunde and Informatica, Amsterdam, The Netherlands*

SUMMARY

The acceleration of an option pricing technique based on Fourier cosine expansions on the graphics processing unit (GPU) is reported. European options, in particular with multiple strikes, and Bermudan options will be discussed. The influence of the number of terms in the Fourier cosine series expansion, the number of strikes, as well as the number of exercise dates for Bermudan options is explored. We also give details about the different ways of implementing on a GPU. Numerical examples include asset price processes on the basis of a Lévy process of infinite activity and the stochastic volatility Heston model. Furthermore, we discuss the issue of precision on the present GPU systems. Copyright © 2012 John Wiley & Sons, Ltd.

Received 17 December 2009; Revised 23 March 2011; Accepted 19 January 2012

KEY WORDS: option pricing; Fourier cosine expansions; graphics processing units implementation; options with multiple strikes; Riccati ODEs; jump and stochastic volatility processes

1. INTRODUCTION

In this paper, we deal with a topic from computational finance, that is, the efficient pricing of options on stocks or other assets. Several methods for pricing these contracts exist. The Feynman–Kac theorem relates the conditional expectation of the value of an option contract payoff function under the risk–neutral measure to the solution of a partial differential equation. Various pricing techniques can therefore be developed, such as partial-(integro) differential equation finite-difference solvers, monte Carlo simulations, or numerical integration methods. Option pricing techniques need to be accurate, robust, and fast. The latter feature is particularly necessary when the mathematical asset price models are calibrated to real market data. Option values, with many different parameter values for the underlying asset price process, are then computed thousands of times to fit the mathematical model. In this paper, we focus on this setting and show that it may make sense to perform this task on a graphics processing unit (GPU)-based computer.

Integration-based option pricing with the help of the fast Fourier transform is a common practice (see, for example [1–4]).

A highly efficient pricing method is the method based on Fourier cosine expansions (COS method) [5, 6], on the basis of Fourier cosine series expansions. In this method, on the basis of the conditional expectation formula, the conditional density function of the underlying is approximated by a series expansion that is connected to the characteristic function. The COS method is applicable if the characteristic function of the stochastic asset price process (i.e., the Fourier

*Correspondence to: Bowen Zhang, Delft University of Technology, Delft Institute of Applied Mathematics, Delft, The Netherlands.
†E-mail: bowen.zhang@tudelft.nl

Table I. Comparison of CPU times between different CPUs.

N	16	32	64
ms(CPU1)	0.337	0.388	0.506
ms(CPU2)	0.1032	0.1503	0.2270
max. abs. error	0.0059	9.1396e-08	1.4211e-14

ms, millisecond; max. abs. error, maximum absolute error.

transform of the conditional density function) is available. This is certainly the case for state-of-the-art asset price models, such as the Lévy jump processes and the Heston stochastic volatility process, which we discuss in the present paper. However, also more involved *hybrid* stochastic processes, for example, for interest rates and equity can be considered, as long as we can obtain a characteristic function. For such asset models with stochastic volatility and stochastic interest rate, such as the Heston-Hull-White or the Heston–Gaussian two-factor model, the analytic characteristic function is typically not available. However, after some appropriate reformulations of the stochastic differential equation (SDE) system (see, for example [7]), the coefficients of the characteristic function can be found as the solution of a Riccati system of ordinary differential equations (ODEs), as described in [8]. These ODE systems can be solved numerically by means of an explicit Runge–Kutta method or by other ODE solvers. We will show that this task can also be performed efficiently on a GPU using Compute Unified Device Architecture (CUDA) [9].

In practice, the option values obtained from a mathematical model should be consistent with market option prices. Usually, options with many different strike prices are needed for calibration. In the COS method, European option prices *for a vector of strikes* can be computed in one computation, which accelerates the calibration procedure significantly.

To further accelerate the calibration procedure, two approaches directly come into mind. The easiest is to purchase a faster CPU! As an example, Table I compares error convergence and CPU times between the CPU used in [5] (CPU 1: Intel(R) Pentium(R) 4 CPU, 2.80GHz with cache size 1MB) and a faster CPU (CPU 2: Intel(R) Core(TM)2 Duo CPU, E6550 @ 2.33GHz cache size 4MB) for European calls under a geometric Brownian motion asset process. Time is in milliseconds.[‡]

The faster CPU gives a satisfactory acceleration, but one needs to wait (sometimes up to two years) for an acceleration by a factor two.

Another possibility to accelerate the pricing engine is to run the program, or parts of it, on the popular GPU, which supports parallel computation. Executing a code on a GPU is worthwhile if

1. a program can be divided into several independent parts;
2. a program does not contain many sequential parts; and
3. a program does not require much memory transfer from host to device or vice versa.

We will study the suitability of the GPU for the important task of calibration, which is traditionally performed with European options. In that respect, it is a challenge to accelerate the pricing of European options. A well-known insight in Numerical Mathematics and Scientific Computing is that it is often easy to accelerate a suboptimal solver on novel hardware, but it is hard to improve the computing times for optimal solvers.

In our paper, we will demonstrate a significant performance improvement on the GPU, because of parallelization, when pricing European options for several options simultaneously, that is, with *multiple strikes*.

Earlier work in the direction of GPU acceleration of an integration-based option pricing method [10] achieved an impressive speed-up on the GPU for options with early exercise features. By a relatively large number of space and time points, the advantages of the GPU implementation were shown. As the COS method exhibits an exponential rate of convergence, a very large number of computations is not necessary to obtain accurate option values. We will focus on a speed-up on the GPU with not-more-than-necessary terms in the Fourier cosine expansion.

[‡]1 millisecond $= 10^{-3}$ second.

For certain modern option products, it also makes sense to calibrate to barrier options or other liquid financial products. The pricing of barrier options with the COS method is closely connected to the example of pricing Bermudan options in the present paper, see [6].

The outline of this paper is as follows. Section 2 gives an introduction of the COS option pricing method. For European options, different ways of implementation of the method are described in Section 3. Section 4 presents the speed-up for multistrike European options on the GPU. The acceleration of an explicit Runge–Kutta method for numerically solving systems of Riccati ODEs to approximate a characteristic function (if it is not available in closed form) is presented in Section 4.3. Section 5 gives pricing results for Bermudan options, which may be exercised early (before the maturity date). Here, the influence of the number of terms in the Fourier cosine expansion as well as the number of early exercise dates on the GPU speedup is discussed.

The GPU we work on is an NVIDIA GeForce 9800 GX2 produced by NVIDIA, Santa Clara, California, USA, which has two GPUs and 1 GB of memory (512 MB for each GPU); the CPU on the same computer, needed for data transfer, is an AMD Athlon(tm)64 X2 Dual Core Processor 4600+ (cache size 512 KB, 2412.364 MHz).

The results obtained are compared with timings on a CPU from an Intel(R) Core(TM)2 Duo CPU E6550 (@ 2.33 GHz cache size 4 MB).

2. COS PRICING METHOD

Starting from the risk–neutral valuation formula

$$v(x,t_0) = e^{-r\Delta t}\int_{-\infty}^{\infty} v(y,T)f(y|x)\mathrm{d}y,$$

where $v(x,t)$ is the option value and x, y can be any increasing functions of the underlying at t_0 and T, respectively; we truncate the integration range, so that

$$v(x,t_0) \approx e^{-r\Delta t}\int_{a}^{b} v(y,T)f(y|x)\mathrm{d}y \tag{1}$$

with $|\int_{\mathbb{R}} f(y|x)\mathrm{d}y - \int_a^b f(y|x)\mathrm{d}y| < TOL$. Error analysis of the various approximations is given in [5, 6].

The conditional density function of the underlying is then approximated by means of the characteristic function via a truncated Fourier cosine expansion as follows:

$$f(y|x) \approx \frac{2}{b-a}{\sum_{k=0}^{N-1}}' Re\left(\phi\left(\frac{k\pi}{b-a};x\right)\exp\left(-i\frac{ak\pi}{b-a}\right)\right)\cos\left(k\pi\frac{y-a}{b-a}\right), \tag{2}$$

where Re means taking the real part of the expression in brackets and $\phi(\omega;x)$ is the characteristic function of $f(y|x)$ defined as

$$\phi(\omega;x) = \mathbb{E}(e^{i\omega y}|x). \tag{3}$$

The prime at the sum symbol in (2) indicates that the first term in the expansion is multiplied by one-half. Replacing $f(y|x)$ by its approximation (2) in (1) and interchanging integration and summation gives us the COS algorithm to approximate the value of a European option:

$$v(x,t_0) = e^{-r\Delta t}{\sum_{k=0}^{N-1}}' Re\left(\phi\left(\frac{k\pi}{b-a};x\right)e^{-ik\pi\frac{a}{b-a}}\right)V_k, \tag{4}$$

where

$$V_k = \frac{2}{b-a}\int_a^b v(y,T)\cos\left(k\pi\frac{y-a}{b-a}\right)\mathrm{d}y \tag{5}$$

is the Fourier cosine coefficient of $v(y,T)$, which is available in closed form for several European option payoff functions.

Formula (4) can be directly applied to calculate the value of a European option, and it also forms the basis for pricing Bermudan options.

The COS algorithm exhibits an exponential convergence rate for all processes whose conditional density $f(y|x) \in C^{\infty}([a,b] \subset \mathbb{R})$. The size of the integration interval $[a,b]$ can be determined with the help of the cumulants [5].

2.1. *Pricing of European options with multistrike features*

With $X_t = \log(S_t/K)$, the solution for Equation (5) can be written as

$$V_k = U_k K, \tag{6}$$

where

$$U_k = \begin{cases} \frac{2}{b-a}(\chi_k(0,b) - \psi_k(0,b)), & \text{for a call,} \\ \frac{2}{b-a}(\psi(a,0) - \chi(a,0)), & \text{for a put,} \end{cases} \tag{7}$$

with

$$\chi_k(x_1,x_2) := \int_{x_1}^{x_2} e^x \cos\left(k\pi \frac{x-a}{b-a}\right) \mathrm{d}x, \tag{8}$$

$$\psi_k(x_1,x_2) := \int_{x_1}^{x_2} \cos\left(k\pi \frac{x-a}{b-a}\right) \mathrm{d}x. \tag{9}$$

Now, the pricing Formula (4) reads

$$v(x,t_0) = Ke^{-r\Delta t} Re\left({\sum_{k=0}^{N-1}}' \phi\left(\frac{k\pi}{b-a};x\right) \cdot e^{-ik\pi\frac{a}{b-a}} U_k\right). \tag{10}$$

Let us assume that we deal with a vector of strikes, $K = [K(1), \cdots, K(P)]^T$, where P is the number of strikes. In this setting, x and the truncated upper and lower bounds, a and b, are functions of K, which implies that they are also vectors with length P. For a vectorised version of the COS method, enabling an efficient computation of multistrike options, we define the following matrices:

- Φ is a $(P \times N)$ matrix with elements
$$\Phi(j,k+1) = \phi\left(\frac{k\pi}{b(j)-a(j)};x(j)\right) \exp\left(-ik\pi\frac{a(j)}{b(j)-a(j)}\right) U_k(a(j),b(j)),$$
$j = 1,\cdots,P, k = 0,\cdots,N-1.$
- Λ is a $(P \times P)$ diagonal matrix with $\Lambda(j,j) = K(j),\ j = 1,\cdots,P.$

With these matrices, the formula for pricing options with multistrike features reads

$$v(\mathbf{x},t_0) = e^{-r\Delta t}\ \Lambda\ Re(\Phi\ I), \tag{11}$$

where I is an $(N \times 1)$ vector with its first element equal to 0.5 and all the other elements equal to 1. In this way, option values for many strikes can be computed simultaneously.

2.2. *COS method for Bermudan options*

The pricing formula for a Bermudan option with $\mathcal{M}$ exercise dates, with $m = \mathcal{M}, \mathcal{M}-1, \ldots, 2$, is divided into a stage in which a *continuation value* is computed and a stage in which this value is compared with the payoff, $g(x,t_{m-1}) \equiv v(x,T)$. These stages are given by

$$\begin{cases} c(x,t_{m-1}) = e^{-r\Delta t} \int_{\mathbb{R}} v(y,t_m) f(y|x)\mathrm{d}y, \\ v(x,t_{m-1}) = \max\left(g(x,t_{m-1}), c(x,t_{m-1})\right), \end{cases} \tag{12}$$

followed by the final computation,

$$v(x,t_0) = e^{-r\Delta t} \int_{\mathbb{R}} v(y,t_1) f(y|x)\mathrm{d}y. \tag{13}$$

In this description, we have $x := \ln(S(t_{m-1})/K)$, $y := \ln(S(t_m)/K)$, and $v(x,t), c(x,t)$ as the option value and the continuation value at time t, respectively. For vanilla options $g(x,t) \equiv (\alpha K(\exp(x)-1))^+$ with $\alpha = 1$ for a call and $\alpha = -1$ for a put.

Practically, for each time step, we first determine an early exercise point, x_m^*, for which $c(x_m^*, t_m) = g(x_m^*, t_m)$ by means of the Newton method.[§] At each time step, t_m, we then can split the Fourier cosine coefficients $V_k(t_m)$ into two parts:

$$V_k(t_m) = C_k\left(a, x_m^*, t_m\right) + G_k\left(x_m^*, b\right), \qquad \text{for a call,} \tag{14}$$

$$V_k(t_m) = G_k\left(a, x_m^*\right) + C_k\left(x_m^*, b, t_m\right), \qquad \text{for a put,} \tag{15}$$

for $m = \mathcal{M}-1, \mathcal{M}-2, \ldots, 1$, and

$$V_k(t_M) = G_k(0,b), \qquad \text{for a call,}$$

$$V_k(t_M) = G_k(a,0), \qquad \text{for a put.}$$

Here,

$$G_k(x_1,x_2) = \frac{2}{b-a}\int_{x_1}^{x_2} g(x,t_m)\cos\left(k\pi\frac{x-a}{b-a}\right)\mathrm{d}x, \tag{16}$$

$$C_k(x_1,x_2,t_m) = \frac{2}{b-a}\int_{x_1}^{x_2} \hat{c}(x,t_m)\cos\left(k\pi\frac{x-a}{b-a}\right)\mathrm{d}x, \tag{17}$$

with

$$\hat{c}(x,t_m) = e^{-r\Delta t}{\sum_{k=0}^{N-1}}' Re\left(\Phi\left(\frac{k\pi}{b-a};x\right)e^{-ik\pi\frac{a}{b-a}}\right)V_k(t_{m+1}),$$

from the Fourier cosine expansion.

$G_k(x_1,x_2)$ is known analytically, like for European options, and for Lévy processes,

$$C(x_1,x_2,t_m) \equiv C_k(x_1,x_2,t_m))_{j=0}^{N-1},$$

can be written as

$$C(x_1,x_2,t_m) = e^{-r\Delta t}\mathrm{Im}(M_s u + M_c u)/\pi.$$

Here, Im means taking the imaginary part of the expression in brackets. $M_s u$ stands for the first N elements of $D^{-1}(D(m_s)\cdot D(u_s))$ [¶] and $M_c u$ denotes the computation of the first N elements of $D^{-1}(D(m_c)\cdot sgn \cdot D(u_s))$, in reversed order, see [6].

In this description, we have

$$sgn = [1,-1,1,-1,\ldots]^T,\ m_s = [m_0, m_{-1}, \cdots, m_{1-N}, 0, m_{N-1}, \cdots, m_1]^T,$$

$$m_c = [m_{2N-1}, m_{2N-2}, \cdots, m_1, m_0]^T,\ u_s = [u_0, u_1, \cdots, u_{N-1}, 0, \cdots, 0]^T,$$

with elements

$$m_j = \frac{(x_2-x_1)}{b-a}\pi i, \quad \text{if} \quad j = 0,$$

$$m_j = \frac{\exp\left(ij\frac{(x_2-a)\pi}{b-a}\right) - \exp\left(ij\frac{(x_1-a)\pi}{b-a}\right)}{j}, \quad \text{if} \quad j \neq 0.$$

Finally, $u_j = \phi(j\pi/(b-a))V_j(t_{m+1})$ and $u_0 = \frac{1}{2}\phi(0)V_0(t_{m+1})$.

For all time steps, $m = \mathcal{M}-1, \cdots, 1$, the approximation of $V_k(t_m)$ is recovered from (14) or (15). Option value $v(x,t_0)$ is obtained by inserting $V_k(t_1)$ into (13) and then applying (4) with T replaced by t_1.

[§]If x_m^* lies outside the interval $[a,b]$, we set x_m^* equal to the nearest boundary point.
[¶]D(vector) denotes the discrete Fourier transform, whereas D^{-1} stands for the inverse discrete Fourier transform.

2.3. *Underlying asset processes*

In this paper, we discuss two different underlying asset processes, the Carr-Madan-Geman-Yor (CGMY) process, a Lévy jump process, and the Heston stochastic volatility process. For efficient use of the COS method, the characteristic function related to these processes is required.

The CGMY process, as defined in [11], is a generalization of the Variance Gamma process with the following characteristic function:

$$\phi_{\mathrm{CGMY}}(\omega, t) = \exp(tC\Gamma(-Y)\left[(M - i\omega)^Y - M^Y + (G + i\omega)^Y - G^Y\right]). \tag{18}$$

Four parameters need to be calibrated to market data: parameter $Y : Y < 2$ controls whether the CGMY process has finite or infinite activity. Parameter $C : C > 0$ controls the kurtosis of the distribution, and non-negative parameters G, M give control over the rate of exponential decay on the right and left tails of the density, respectively.

In the Heston stochastic volatility model, the underlying and the volatility are modeled by the following SDEs,

$$\begin{aligned} dx_t &= \left(r - \frac{1}{2}\mu_t\right) dt + \sqrt{\mu_t} dW_{1,t}, \\ d\mu_t &= \lambda(\bar{\mu} - \mu_t)dt + \eta\sqrt{\mu_t} dW_{2,t}, \end{aligned} \tag{19}$$

where x_t and μ_t denote the log–asset price process and the variance of the asset price process, respectively. Parameters λ, $\bar{\mu}$, and η represent the speed of mean–reversion, the long-term mean value of variance, and the volatility of volatility parameters, respectively. Moreover, $W_{1,t}$ and $W_{2,t}$ are Brownian motions, correlated with correlation coefficient ρ.

For the log–asset price in the Heston model, an analytic characteristic function can be found, which reads

$$\begin{aligned} \phi(\omega, \Delta t, \mu_0) = \exp\left(i\omega r \Delta t + \frac{\mu_0}{\eta^2}\left(\frac{1 - e^{-D\Delta t}}{1 - Ge^{-D\Delta t}}\right)(\lambda - i\rho\eta\omega - D)\right) \cdot \\ \exp\left(\frac{\lambda\bar{\mu}}{\eta^2}\left(\Delta t(\lambda - i\rho\eta\omega - D) - 2\log\left(\frac{1 - Ge^{-D\Delta t}}{1 - G}\right)\right)\right), \end{aligned}$$

with $D = \sqrt{(\lambda - i\eta\rho\omega)^2 + (\omega^2 + i\omega)\eta^2}$ and $G = \lambda - i\eta\rho\omega - D/\lambda - i\eta\rho\omega + D$.

For the value of D, we take the square root whose real part is non-negative.

Remark 2.1. (Advantage of COS method on GPU) From [12], we already know that, compared with the execution on a CPU, a GPU is favorable for the individual time-consuming operations in the COS method. Moreover, the elements of the sum in (4) can be computed simultaneously. The GPU is therefore expected to outperform the CPU implementation, in particular, when many computations are necessary, like for European options with multistrike features.

3. EUROPEAN OPTIONS

A European option can be viewed as a special case of a Bermudan option with only one possible exercise date (the expiry time). For European options, the Fourier and inverse Fourier transform operations are not needed.

3.1. *Different ways of graphics processing unit implementation*

In this section, we discuss different ways of implementation on a GPU. Consider a simple case where we need to price one vanilla option.

From (4), the COS algorithm can be decomposed into two steps, that is, computations on each element of a vector, $Re(\exp(-ik\pi\frac{a}{b-a})\phi(\frac{k\pi}{b-a}; x)V_k)$, which can be parallelized, and the summation of vector elements.

We consider three ways of GPU implementation:

1. Directly run the whole code on the GPU, referred to as GPU1;
2. All operations related to each vector element are parallelized on the GPU, whereas the summation is performed on the CPU. This hybrid GPU/CPU way of implementation is referred to as GPU2; and
3. When summing up the elements of a vector, we can split the vector in two vectors, each of size $N/2$, and sum up these two on the GPU. The procedure is repeated until $N = 1$. The summation of pairs of elements can be parallelized on the GPU this way. The number of operations for the summation can be reduced from N to $\log_2(N)$, referred to as GPU3.

Figure 1 presents a comparison of the time consumed by the aforementioned three ways of GPU implementation and also the CPU time. Clearly, GPU1 is faster than the CPU only when N, the number of terms in the Fourier cosine expansion, is large, whereas the implementations GPU2 and GPU3 are faster than either of the CPU implementation or GPU1. Moreover, as N increases, the speedup of GPU2 and GPU3 also increases. GPU3 is slightly slower than GPU2 for small N, but when N is very large, GPU3 beats GPU2.

Unlike the CPU or GPU1, the time for GPU2 and GPU3 does not increase much as N increases, until $N \approx 2^{16}$.

For Bermudan options, implementation GPU3 is preferred because the complete code then runs on the GPU, and data transfer can be reduced. With GPU2, we would need to transfer data at each time step, which consumes time.

In this paper, we will use implementation GPU3 for all numerical examples to follow.

3.2. Numerical example

We take as an example the CGMY model with $Y = 1.5$. The other parameters are chosen as $S_0 = 100, r = 0.1, K = 80, C = 1, M = 5, G = 5$. Table II compares time and accuracy of the CPU and the GPU results, with time measured in milliseconds.

Table II shows that with $N = 1024$, the GPU implementation is 1.5 times faster than running the code on the CPU. However, it is not necessary to take such a large value of N in the COS method in practice, as with $N = 256$, the option values already differ less than one basis point. Therefore, in

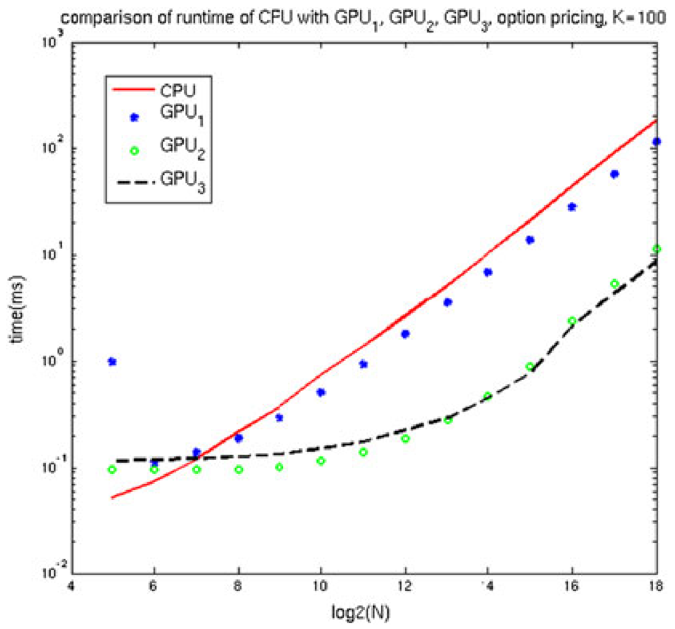


Figure 1. Comparison of different graphics processing unit (GPU) implementations.

the present setting (COS method, small values of N), the use of a GPU is not really advantageous. However, when dealing with multiple strikes, presented in Section 4, many more computations are needed so that the GPU performance will be more profound.

4. MULTIPLE STRIKE OPTION PRICING

In this section, we focus on pricing European options, but now with multiple strikes, on the GPU. The parameters used, next to $S_0 = 100$, in the CGMY process and for the Heston process are the following:

$$\begin{array}{ll} \text{CGMY:} & r = 0.1, C = 1, G = 5, M = 5, Y = 1.5, T = 1; \\ \text{Heston:} & r = 0.040, \lambda = 1.577, \eta = 0.575, \mu_0 = 0.018, \rho = -0.57, T = 10. \end{array}$$

We price European call options with vectors of strikes, as shown in Table III. To efficiently implement (11) on a GPU, we first divide the P-axis and the N-axis in different blocks and threads, as shown in Figure 2.

Then each element of a $(P \times N)$-matrix can be calculated *simultaneously* as illustrated in Figure 3.

When performing the summation on each row of matrix v, as the final step in (11), we divide the $(P \times N)$-matrix into smaller submatrices. For instance, with $N = 128$ and 21 strikes, the corresponding 21×128 matrix can be subdivided into 56 matrices of size 3×16. The elements of these smaller submatrices are copied to shared memory as shown in Figure 4.

Table II. Comparison of time and precision for CPU and GPU implementation of the COS method for a single European option.

N	CPU (time)	GPU (time)	CPU (value)	GPU (value)
256	0.193…	0.182	27.974744	27.974733
1024	0.691…	0.433	27.974744	27.974733

GPU, graphics processing unit.

Table III. Number of strikes used in the numerical examples.

Number of strikes	Value of K
3	$80, 100, 120$
5	$80, 90, 100, 110, 120$
9	$80, 85, \cdots, 115, 120$
13	$70, 75, \cdots, 125, 130$
17	$60, 65, \cdots, 135, 140$
21	$50, 55, \cdots, 145, 150$

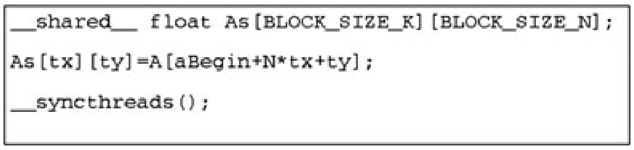
```
int i=blockIdx.x*blockDim.x+threadIdx.x;
int j=blockIdx.y*blockDim.y+threadIdx.y;
```

Figure 2. Blocks and threads.

```
v[i*Ndim+j]=exp(-r*T)*K[j]*real(cf[i*Ndim+j]

            *exp((x[i]-a[i])*omega[i*Ndim+j])*U[i]);
```

Figure 3. Parallelization of COS method for options with multistrike.



```
__shared__ float As[BLOCK_SIZE_K][BLOCK_SIZE_N];
As[tx][ty]=A[aBegin+N*tx+ty];
__syncthreads();
```

Figure 4. Data transfer from global memory to shared memory.

Here, $aBegin$ is the first location of As, that is, the blocks with shared memory, and tx, ty are the thread indices of As. Then, as we run the program, data transfer only happens within the shared memory and not in the global memory, which saves us many GPU time.

4.1. *Convergence and precision*

Tables IV and V present the convergence behavior and the precision of option values with 5 and 21 strikes, respectively, for the two underlying processes. Time is again measured in milliseconds. Option values obtained with $N = 2^{16}$, and in double precision, are taken as the reference values. We calculate the maximum absolute error, for varying values of N, over the strike vectors.

Both the GPU and CPU timing results are shown to be extremely fast, as we need only $N = 64$ for the CGMY process and $N = 128$ for Heston's model to obtain converged option values on the GPU and the CPU. However, the execution time on the GPU is significantly smaller than on the CPU. As we are in the milliseconds range, one might question the relevance of this gain in speed.

Table IV. Convergence and maximum absolute error when pricing a vector of five strikes.

		CGMY model		
	N	32	64	128
MATLAB	ms	0.413230	0.745590	1.388770
	max. abs. error	1.3409e-05	$<10^{-14}$	$<10^{-14}$
GPU	ms	0.141144	0.143051	0.152826
	max. abs. error	0.000027	0.000034	0.000034
		Heston model		
	N	64	128	256
MATLAB	ms	1.206600	1.958680	3.873950
	max. abs. error	4.2839e-04	2.2218e-08	$<10^{-14}$
GPU	ms	0.154972	0.159979	0.182867
	max. abs. error	0.000534	0.000104	0.000104

CGMY, Carr-Madan-Geman-Yor model; MATLAB, matrix laboratory; GPU, graphics processing unit; max. abs. error, maximum absolute error; ms, millisecond.

Table V. Convergence and maximum absolute error when pricing a vector of 21 strikes.

		CGMY model		
	N	32	64	128
MATLAB	ms	1.335130	2.690250	5.340340
	max. abs. error	1.3409e-05	$<10^{-14}$	$<10^{-14}$
GPU	ms	0.154018	0.169992	0.200987
	max. abs. error	0.000053	0.000053	0.000053
		Heston model		
	N	64	128	256
MATLAB	ms	3.850890	7.703350	15.556240
	max. abs. error	6.0991e-04	2.7601e-08	$<10^{-14}$
GPU	ms	0.177860	0.209093	0.333786
	max. abs. error	0.000534	0.000144	0.000144

CGMY, Carr-Madan-Geman-Yor model; MATLAB, matrix laboratory; GPU, graphics processing unit; max. abs. error, maximum absolute error; ms, millisecond.

However, within a calibration setting, option prices have to be computed several thousands of times, which immediately turns a small gain into a significant profit. The advantage of the use of the GPU becomes more pronounced when the value of N and the number of strikes, $K(P)$, increase; since then more arithmetic operations are required. As shown in Tables IV and V, the acceleration on the GPU for Heston's model increases for five strikes from 12 to 21, as N increases from 128 to 256. For 21 strikes, the speedup on the GPU is a factor, 37 for $N = 128$ and 47 for $N = 256$. Because more computations are necessary to evaluate the characteristic function of the Heston model, the speedup on the GPU for the Heston model is higher than for the CGMY model. However, because of the fact that the GPUs used in this test give computed values in single precision, round-off errors can build up during the computation of the characteristic function on the GPU. A larger maximum absolute error for the Heston model is therefore observed in the tables.

Figure 5 presents the speedup obtained on the GPU for different numbers of strikes, with $N = 128,\ 512,\ \text{and}\ 2048$. It displays a speedup of 30–40 for 21 strikes on the GPU with $N = 128$ and a speedup of 60–70 for 13 and 17 strikes with $N = 512$. The GPU appears to be a very satisfactory architecture for pricing options at multiple strikes.

Figure 5 also displays the influence of data transfer on the GPU time. The data transfer time is the time to transfer the input (in this case, the value of the strikes) from the CPU to the GPU at the beginning plus the time to transfer the output (the option values) back to the CPU after the computations have finished.

Graphics processing unit time is the sum of the data transfer and the computation time. For larger input sizes, the GPU is advantageous regarding the computations but the data transfer time increases. The influence of data transfer time is more pronounced when the number of terms in the cosine expansion, N, reaches 1000.

For a closer look, we refer to Table VI from which we see that because of the parallelization, the GPU computation time hardly changes when N increases from 2048 to 8192. On the other hand, data transfer time is more significant with increasing N, which gives a smaller acceleration for large N and many strike values, as also shown in Figure 5.

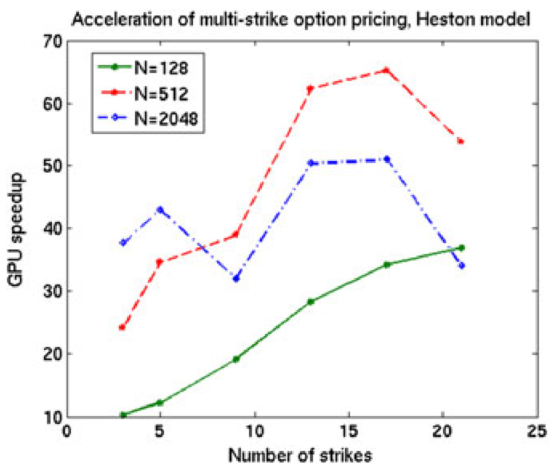


Figure 5. Graphics processing unit speed-up for Heston model with different number of strikes, $N = 128,\ 512,\ \text{and}\ 2048$.

Table VI. Influence of data transfer on the computational time on a GPU.

N	2048	4096	8192
Data transfer time excluded	0.086069	0.087023	0.084877
Data transfer time included	0.493050	1.350164	4.626989

GPU, graphics processing unit.

Table VII. Convergence and maximum absolute error when pricing a vector of 11 strikes.

		Set 1 with $T = 0.1$ (1 month)		
	N	512	1024	2048
MATLAB	ms	5.9980	13.4580	27.9570
	max. abs. error	2.4248e-04	4.0405e-06	1.1445e-07
GPU	ms	0.188828	0.279903	0.492811
	max. abs. error	0.000255	0.000025	0.000025
		Set 2 with $T = 0.0001$ (1 h)		
	N	1024	2048	4096
MATLAB	ms	9.4810	26.9409	60.3240
	max. abs. error	1.6419e-04	7.5000e-08	4.8746e-16
GPU	ms	0.280857	0.494957	1.353979
	max. abs. error	0.000019	0.000010	0.000010

MATLAB, matrix laboratory; GPU, graphics processing unit; max. abs. error, maximum absolute error; ms, millisecond.

4.2. *Option pricing with short maturity times*

In certain practical applications, profit opportunities may arise from the calibration of Levy processes for European option contracts with a very short time to maturity.

In this section, we report on the recent implementation of the COS method on a different GPU architecture, a Tesla C1060, which also supports double precision. The true advantages of this GPU for the COS method implementation are for short maturity options because we then need a larger value of N (many terms in a cosine expansion) for convergence with the COS method, thus more computations need to be performed.

In Table VII, the computational time in milliseconds is recorded on this GPU and on the CPU for options under the CGMY model. A vector of strikes, $K = 90, 92, \cdots, 110$, is priced simultaneously. The maximum absolute error[‖] over all strikes for short maturities is also presented. The model parameters used are $C = 0.5$, $G = 10$, $M = 30$, $Y = 0.5$ (Set 1) and $C = 0.25$, $G = 5$, $M = 50$, $Y = 1.5$ (Set 2).

For the experiments in Tables IV and V, we reached convergence with $N = 128$, and we obtained speedups of 9 and 26 on the GPU for 5 and 21 strikes, respectively. In Table VII, $N = 1024$ and $N = 2048$ were used in the two experiments, respectively, for convergence at short maturities, and the Tesla GPU then achieves speedups of 48 and 54.

4.3. *Riccati ordinary differential equations and characteristic function*

In this subsection, we will implement an explicit Runge–Kutta ODE solver to determine the characteristic function for Heston's model. The Riccati ODEs arise in the determination of a characteristic function. Sometimes, as in the case of the Heston model, the characteristic function is known analytically. In the cases for which we cannot find an analytic expression, we may resort to the numerical solution of the Riccati ODEs. The numerical procedure is, for example, necessary for systems of SDEs that include, for example, stochastic interest rate and stochastic volatility. In this numerical procedure, an operation such as taking the square root of a complex number is not needed, in contrast to the evaluation of the analytic solution. For the numerical ODE solver, the influence of single precision arithmetic is therefore less pronounced, and we even obtain a higher accuracy than with the analytic characteristic function.

From [8], we know that the characteristic function for the Heston model (19) is of the following form:

$$\phi_{x,\mu}(\omega, t) = e^{A(\omega,t)+B_\mu(\omega,t)\mu_0+B_x(\omega,t)x_0}, \tag{20}$$

[‖]Reference values have been obtained with $N = 2^{20}$.

with the coefficients $A(\omega,t)$, $B_x(\omega,t)$, and $B_\mu(\omega,t)$ given by the following Riccati ODEs:

$$\begin{cases} \frac{\partial}{\partial t}B_x(\omega,t)=0, & B_x(\omega,0)=i\omega,\\ \frac{\partial}{\partial t}B_\mu(\omega,t)=0.5\eta^2\mu_t^2-(\lambda-i\rho\eta\omega)\mu_t-0.5i\omega-0.5\omega^2, & B_\mu(\omega,0)=0,\\ \frac{\partial}{\partial t}A(\omega,t)=\lambda\bar{\mu}\mu_t+i(r-q)\omega, & A(\omega,0)=0. \end{cases} \tag{21}$$

It is easy to see from (21) that $B_x(\omega,t) = i\omega$, $A(\omega,t)$, and $B_\mu(\omega,t)$ are now solved numerically by the explicit fourth order Runge–Kutta method and inserted in the general characteristic function (20), on the basis of which we employ the COS method.

Table VIII shows the corresponding timing results on the GPU and the CPU for 5 and 21 strikes, respectively. Compared with the results in Tables IV and V, a higher speed-up is achieved on the GPU here. For 5 strikes, with $N = 128$ and $N = 256$, the GPU timings are around 50 times faster than the CPU results. For 21 strikes, the acceleration on the GPU is a factor of 100. Note that for 21 strikes because of the increased data transfer, the speed-up reduces as N increases, but because of the exponential convergence rate, $N = 128$ is sufficient for convergence.

Figure 6 shows the speedup on the GPU with $N = 128,\ 256,$ and 512 and a different number of strikes. Of course, option pricing with an analytic characteristic function is still the fastest, but the numerical solution of Riccati ODEs can be performed highly efficiently on these units.

Table VIII. Convergence and maximum absolute error for 5 and 21 strikes, Heston model's characteristic function obtained by the fourth order Runge–Kutta method.

		5 strikes		
	N	64	128	256
MATLAB	ms	37.9491	50.5196	81.1083
	max. abs. error	4.2848e-04	8.4949e-08	1.0650e-07
GPU	ms	1.091957	1.121998	1.253843
	max. abs. error	0.000443	0.000013	0.000013
		21 strikes		
MATLAB	ms	84.9870	145.0005	268.2299
	max .abs. error	6.0979e-04	1.5607e-07	1.2847e-07
GPU	ms	1.228094	1.402855	2.689838
	max. abs. error	0.000611	0.000037	0.000037

MATLAB, matrix laboratory; GPU, graphics processing unit; max. abs. error, maximum absolute error; ms, millisecond.

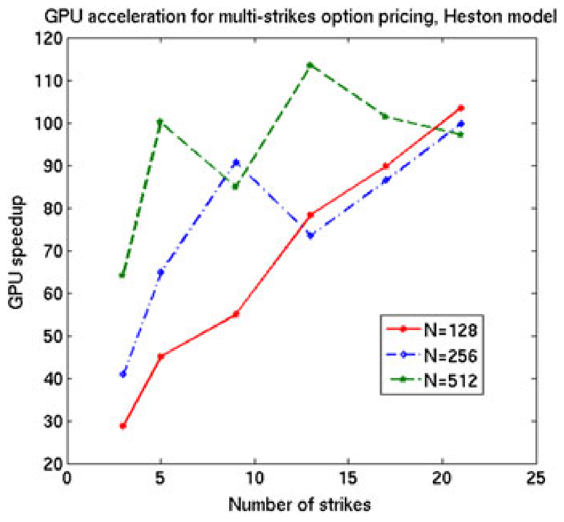


Figure 6. Graphics processing unit (GPU) acceleration for Heston model with different strike vectors and N, characteristic function obtained by fourth order Runge–Kutta method.

5. BERMUDAN OPTIONS

In this section, we consider the pricing of Bermudan options with a discrete number of early exercise points. Also here, we do not focus on large values of N or on many time points, $\mathcal{M}$. With $N = 160$ in the COS method, we obtain the price of a Bermudan option with an error smaller than 10^{-9}. Moreover, Bermudan options with up to 64 time points ($\mathcal{M} = 64$) are typically sufficient to obtain a very satisfactory approximation of the value of an American options (that can be exercised at any time before expiry) by a repeated Richardson extrapolation, see [13].

We will show in this section that by exploiting the parallelism of a GPU, the COS algorithm is somewhat faster than on the CPU with a realistic number of terms, N. The reduction of the total execution time is however not as impressive as in the previous sections.

We use as a numerical example a Bermudan put option under the CGMY process. The model parameters are $S_0 = 100,\ K = 80,\ \mathcal{M} = 10,\ C = 1,\ M = 5,\ G = 5,\ Y = 1.5$. Tables IX and X compare time and accuracy on the GPU and CPU.

With N relatively small, the Bermudan option price converges well, and the resulting error is small. However, as mentioned before, as N becomes larger, which is probably necessary for other types of options, the advantage of using the GPU will become more pronounced.

We increase the number of exercise dates, $\mathcal{M}$, to 20, 40, and 80 and compare the GPU and CPU execution times. The model parameters are as before. We use $N = 512$. The timing results are listed in Table XI.

With $N = 512$, the GPU is twice as fast as the CPU for different numbers of exercise dates. The number of exercise dates does not influence the speedup of the GPU. This is because the algorithm can not be parallelized in time, as we use values at t_{m+1} to calculate values at t_m in the backward recursion procedure. This results in a recursion of $m = \mathcal{M} - 1, \cdots, 1$ in the CUDA implementation.

Table IX. Comparison between CPU and GPU execution times for the CGMY model for a Bermudan option; different numbers of terms in the Fourier cosine expansion.

N	CPU (time)	GPU (time)
256	13.15…	11.02
512	24.69…	13.69
1024	46.65…	27.34

CGMY, Carr-Madan-Geman-Yor; GPU, graphics processing unit.

Table X. Precision on the GPU, Bermudan put option, for different numbers of terms in the Fourier cosine expansion.

N	CPU (value)	GPU (value)
256	28.829781987399432	28.829739
512	28.829781987399425	28.829721
1024	28.829781987399404	28.829756

GPU, graphics processing unit.

Table XI. Comparison of CPU and GPU times for the CGMY model, Bermudan option with a different numbers of exercise dates.

$\mathcal{M}$	CPU (time)	GPU (time)
20	51.04…	26.09
40	104.00…	50.88
80	210.20…	100.54

GPU, graphics processing unit.

6. CONCLUSIONS

The COS method is a highly efficient pricing method for European and early exercise options. It is a challenge to implement an efficient method, which requires a small number of terms in the Fourier cosine expansion and a small number of exercise dates to approximate the value of an American option, efficiently on a GPU.

We implemented the COS algorithm on the GPU using CUDA. The GPU timings and option values were compared with those obtained on the CPU. To exploit the advantages of a GPU, we split a vector (or matrix) and perform summation in parallel, which enhanced the GPU performance.

A highly satisfactory performance on the GPU was observed especially in the case of multiple strike European option computations. Then, we find GPU speedup ranging from 10 to 100, depending on the form of the characteristic function and on the number of strikes that is computed simultaneously.

Although computation on a GPU to date still exhibits some disadvantages, such as a somewhat time-consuming memory transfer, it is a highly promising architecture for option pricing.

For involved SDE asset price models, the characteristic function must be determined numerically by a Riccati ODE solver, as an analytic expression is not available. On the basis of the results in this paper, the numerical ODE treatment by means of a fourth order Runge–Kutta method is also highly efficient on a GPU.

A GPU-based architecture may serve very well as a calibration engine in option pricing.

A comparison between a GPU and a single CPU may not be completely fair. Instead, we should have compared it with a multicore CPU machine (if it would have been available to us). This would have improved the CPU performance.

However, it should be noted that with a multicore CPU, also the data transfer time between the CPU and the GPU will improve. Because data transfer time dominates the total execution time so far, we also expect an increased speedup of the total GPU time compared with the times reported in this manuscript.

REFERENCES

1. Carr PP, Madan DB. Option valuation using fast Fourier transformation. *Journal of Computational Finance* 1999; **2**:61–73.
2. Chourdakis K. Option pricing using the fractional FFT. *Journal of Computational Finance* 2004; **8**(2):1–18.
3. Dempster MAH, Hong SSG. Spread option valuation and the fast Fourier transform. *Technical Report WP 26/2000*, The Judge Institute of Management Studies, University of Cambridge, CB2 1AG, Cambridge, UK, 2000.
4. Jackson K, Jaimungal S, Surkov V. Option pricing with regime switching Lévy processes using Fourier space time–stepping. In *Proceeding of the Fourth IASTED international Conference on Financial Engineering and Applications*, 2007; 92–97.
5. Fang F, Oosterlee CW. A novel option pricing method for European options based on Fourier-cosine series expansions. *SIAM Journal on Scientific Computing* 2008; **31**:826–848.
6. Fang F, Oosterlee CW. Pricing early-exercise and discrete barrier options by Fourier-cosine series expansions. *Numerische Mathematik* 2009; **114**:27–62.
7. Grzelak LA, Oosterlee CW. On the Heston model with stochastic interest rates. *SIAM Journal on Financial Mathematics* 2011; **2**:255–86.
8. Duffie D, Pan J, Singleton K. Transform analysis and asset pricing for affine jump-diffusions. *Econometrica* 2000; **68**:1343–1376.
9. NVIDIA Cuda Compute Unified Device Architecture, Programming Guide, Version 1.0, 2007.
10. Surkov V. Parallel option pricing with Fourier space time–stepping method on graphics processing units. *Parallel Computing* 2010; **36**(7):372-380.
11. Carr P, Geman H, Madan DB, Yor M. The fine structure of asset returns: an empirical investigation. *Journal of Business* 2002; **75**(2).
12. Zhang B, Oosterlee CW. Option pricing with COS method on Graphics Processing Unit. In *Proceeding of the 23rd IEEE International Parallel & Distributed Processing Symposium*, May 2009; 25–29, paper number PDCoF-108.
13. Lord R, Fang F, Bervoets F, Oosterlee CW. A fast and accurate FFT-based method for pricing early-exercise options under Lévy processes. *SIAM Journal on Scientific Computing* 2008; **30**:1678–1705.