

We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

5,500

Open access books available

136,000

International authors and editors

170M

Downloads

Our authors are among the

154

Countries delivered to

TOP 1%

most cited scientists

12.2%

Contributors from top 500 universities



WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index
in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com



Spread Option Pricing on Single-Core and Parallel Computing Architectures

Shiam Kannan and Mesias Alfeus

Abstract

This paper introduces parallel computation for spread options using two-dimensional Fourier transform. Spread options are multi-asset options whose payoffs depend on the difference of two underlying financial securities. Pricing these securities, however, cannot be done using closed-form methods; as such, we propose an algorithm which employs the fast Fourier Transform (FFT) method to numerically solve spread option prices in a reasonable amount of short time while preserving the pricing accuracy. Our results indicate a significant increase in computational performance when the algorithm is performed on multiple CPU cores and GPU. Moreover, the literature on spread option pricing using FFT methods documents that the pricing accuracy increases with FFT grid size while the computational speed has opposite effect. By using the multi-core/GPU implementation, the trade-off between pricing accuracy and speed is taken into account effectively.

Keywords: spread option, single core, parallel computing

1. Introduction

Spread options have widespread uses across many industries, remarkably in the seasonal commodity market futures. One notable example is the crack spread, which is the pricing difference between a barrel of crude oil and the petroleum products refined from it. Traditionally, crack spreads involve the purchase of oil futures and the simultaneous sale of futures of the refined product, whether it be gasoline, heating oil, or other similar products. Refiners seek a positive spread between the prices of crude oil and refined products, meaning that the price of the input (oil in this case) is lower than the price of the output (gasoline, kerosene, etc.).

Beyond the oil industry, spread options are utilized by suppliers in industries as disparate as the soybean and electricity markets. Soybean spread options are known as crush spreads, and electricity spread options are known as spark spreads. Similar to crack spreads, both these options seek to maximize the spread between input costs and output prices for suppliers to maximize profit.

The use of spread options across such disparate fields is but a testament to their widespread use. Considering the popularity of spread options, it thus prompts the needs of accurate and fast pricing of price of these options. This paper dwells on the discussion of fast algorithms for these options.

As mentioned previously, pricing spread options tends to be something that cannot be easily performed using traditional closed-form solutions. Therefore, we explore the utilization of the fast Fourier transform method to price these securities. Specifically, we perform the FFT on the spread options payoff, assuming knowledge of the model joint characteristic function, which we represent as a pointwise multiplication of the characteristic function and the complex gamma function in the Fourier domain.

Regarding our implementation, we adapt the parallel computing Toolbox in MATLAB to take advantage of the multi-core capabilities of GPU processing, to substantially improve the performance and computational efficiency of the algorithm for spread options. This methodology may serve a great deal especially in model calibration and risk management approach. For measuring performance, we only time the execution of the inverse Fast Fourier Transform, as the prior steps are merely initialization of the necessary arrays. For measuring accuracy, we compute the Euclidean norms of each of the resulting option price arrays from each implementation (single-core and GPU), and find the percent error between the norms as follows:

$$\frac{\text{GPU norm} - \text{single core norm}}{\text{single core norm}} \times 100 \quad (1)$$

The main contribution of this paper is the use of parallel computation for the spread option value using two-dimensional Fourier transform. Our implementation is developed both for single-core processors, as well as for parallel processing on multi-core/GPU systems. For both execution methods, we have implemented our algorithm using MATLAB built-in functions to produce a version of the algorithm compatible for multi-core systems. To ascertain the impact of the different environments as well as the different methods of execution on the computational efficiency of the algorithm, we record the times of execution for different values of FFT grid size N , which is the number of grid points of discretization of the characteristic function along the two asset dimensions, for both the classical single-core implementation and the multi-core/GPU implementation. This approach completely eliminates the trade off between computational accuracy and speed, that is, we price spread option accurately and in a fastest possible way.

2. Model description

2.1 Spread option valuation

We fix the trading time horizon T and consider a filtered probability space $(\Omega, \mathcal{F}, P, (\mathcal{F}_t)_{t < T < T})$ defined in the usual way.

The goal is to compute the value of an European spread option between stock price processes $S_1 = \{S_1(t)\}_{0 \leq t \leq T}$ and $S_2 = \{S_2(t)\}_{0 \leq t \leq T}$ with maturity time T and exercise price K .

At expiration time T , the payoff is given by

$$S_T(S_1, S_2, K) = \max \{S_1(T) - S_2(T) - K, 0\}, \quad (2)$$

and under a risk-neutral conditional measure¹ Q its value at time t is given by

¹ Specifically: Under the risk-neutral measure associated with taking the continuously compounded savings account as the numeraire, and (for expositional simplicity) assuming a constant interest rate r .

$$S_t(S_1, S_2, K) = e^{-r(T-t)} E^Q [(S_1(T) - S_2(T) - K)^+ | \mathcal{F}_t]. \quad (3)$$

Under normality assumption, Eq. (3) can be analytically approximated as done in [1]. However, a departure from normality assumption ushers into numerical computational difficulties. In option pricing literature of finance, Fourier transform-based method is usually the best candidate to approximate the solution for Eq. (3), in this case whenever the joint characteristic function of the asset price processes, S_1 and S_2 are available. For spread options pricing valuation methodology using two-dimensional fast Fourier transform (FFT) techniques was coined in [2, 3]. We cite the important formula from [3] that gives the price for spread option

$$S_t(S_1, S_2, K) = e^{-r(T-t)} E^Q [(S_1(T) - S_2(T) - K)^+ | \mathcal{F}_t] \\ = \frac{1}{(2\pi)^2} e^{-rT} \int \int_{R+i\epsilon} e^{iuX'_0} \Phi(u, T) \hat{P}(u) d^2u \quad (4)$$

$$\hat{P}(u) = \frac{\Gamma(i(u_1 + u_2) - 1) \Gamma(-iu_2)}{\Gamma(iu_1 + 1)} \quad (5)$$

$$\Gamma(z) = \int_0^\infty e^{-t} t^{z-1} dt, \quad (6)$$

where $X_0 = \log S$, $\Phi(u, T) = E[e^{iu(X'_T - X'_0)}]$, is the joint characteristic function of the log return.

Fast and accurate pricing is often the most desirable feature of the model. In [4], the authors consider spread options pricing in C++ using fast Fourier transform in the west (FFTW). They observed the trade off between fast computation and numerical accuracy; pricing accuracy is monotonic in the number of FFT grid size used in the price computation. However, using a large number of FFT grid size slow down the speed of price computation.

2.2 Model characteristic function

Fast Fourier transform (FFT) method is generically applicable in finance because it only requires the specification of the characteristic function of the random variable. In terms of spread options, one just need a characteristic function of the joint distribution of the financial variables in question. Here, we employ two characteristic functions: one based on two-dimensional normal distribution and the other one based on two-dimensional normal inverse Gaussian (NIG) distribution.

2.2.1 Two-dimensional geometric Brownian motion (GBM)

The characteristic function for a spread option comprised of two assets, each of which is modeled as a correlated GBM, is given by

$$\Phi_{GBM}(u; T) = \exp \left(iu \left(rTe - \frac{\sigma^2 T}{2} \right)' - \frac{u \Sigma_{GBM} u' T}{2} \right) \quad (7)$$

where $e = [1, 1]$, $\Sigma_{GBM} = \begin{pmatrix} \sigma_1^2 & \sigma_1 \sigma_2 \rho \\ \sigma_1 \sigma_2 \rho & \sigma_2^2 \end{pmatrix}$, and $\sigma^2 = \text{diag}(\Sigma_{GBM})$, $i = \sqrt{-1}$, $r, \sigma_i : i = 1, 2$ denote the risk-free rate, volatilities, respectively, and ρ is the

correlation parameter between two asset prices processes S_1 and S_2 . Here, Σ_{GBM} is the covariance matrix.

2.2.2 Two-dimensional normal inverse Gaussian (NIG) Levy process

Let S denote a two-dimension NIG random variable. The characteristic function of $S = (S_1, S_2)$ is given by

$$\Phi_{\text{NIG}}(u; T) = \exp \left(iu' \mu T + \delta T \left[\sqrt{\alpha^2 - \beta \Delta \beta'} - \sqrt{\alpha^2 - (\beta + iu) \Delta (\beta + iu)'} \right] \right), \quad (8)$$

where $\alpha, \delta \in R_+$, $\beta, \mu \in R^2$, and $\Delta \in R^{2 \times 2}$ is a symmetric, positive-definite matrix. Moreover, the structural matrix Δ is assumed to have determinant $\text{Det}(\Delta) = 1$.

The covariance matrix corresponding to the two-dimensional NIG-distributed random variable S is

$$\Sigma_{\text{NIG}} = \delta (\alpha^2 - \beta \Delta \beta')^{-\frac{1}{2}} \left(\Delta + (\alpha^2 - \beta \Delta \beta')^{-1} \Delta \beta \beta' \Delta \right) \quad (9)$$

2.3 FFT algorithm

The FFT algorithm for spread option pricing along the line of [3] can be described as follows

Algorithm 1: FFT algorithm for Spread Option pricing

- 1 2-D FFT (N, \bar{u}, ϵ);
 - Input** : N , a power of two; \bar{u} , truncation width; ϵ , damping factor.
 - Define** : $u(k) = (u_1(k_1), u_2(k_2))$ and $x(l) = (x_1(l_1), x_2(l_2))$
 - 2 Set $X_0 = [\log(\frac{S_1}{K}), \log(\frac{S_2}{K})] \in x(l)$
 - 3 **forall** $k, l \in \{1, \dots, N-1\}^2$ **do**
 - 4 $H(k) = (-1)^{k_1+k_2} \Phi(u(k) + i\epsilon) \hat{P}(u(k) + i\epsilon)$;
 - 5 $C(l) = (-1)^{l_1+l_2} e^{-rT} \left(\frac{\eta N}{2\pi}\right)^2 e^{-\epsilon x(l)'};$
 - 6 **end**
 - 7 $V = \mathcal{R}_e(C \times \text{fft2}(H))$ /* with $O(N^2 \log N)$ complexity*/
 - 9 $P \leftarrow K \times V$ /* using an efficient interpolation technique*/;
- Output:** P
-

3. Numerical results

3.1 Implementation outlook

As mentioned earlier, two versions of the algorithm were programmed in MATLAB, namely a single-core variant and a multi-core GPU variant. In MATLAB, the Parallel Processing Toolbox was used to exploit multi-core GPU capabilities to run the algorithm. Among its capabilities is the ability to run for loops and perform array operations in parallel, both on multi-core CPUs as well as GPUs.

As in the Algorithm 1 given above, the for loop in lines 3–6 was run in parallel, across six CPU cores, employing the parfor directive available in the MATLAB Parallel Processing Toolbox. We also sought to run computationally heavy functions on the GPU we had available, to improve the efficiency of our algorithm beyond what would be possible on a multi-core CPU. In that regard, we executed the

inverse FFT, as described in line 7, on the GPU. We accomplished this by copying our H and A arrays onto the GPU, such that any further processing of those arrays would only occur on the GPU. To perform this operation, we utilized the MATLAB inbuilt function `gpuArray()` and copied the two aforementioned arrays to the GPU after the for loop. To transfer the GPU results (following execution of the inverse FFT) back to the local workspace, we used the MATLAB function `gather()`.

Table 1 shows the market parameters, and these inputs are taken from [5] where d_1 and d_2 represent the dividend rate for S_1 and S_2 , respectively.

The computer used to produce the following results was an ASUS ROG Strix Scar II GL704GW, with an Intel Core i7-8750H processor clocked at 2.20 Hz and comprising of 6 cores, 16GB RAM, and an NVIDIA GeForce RTX 2070 GPU with 8GB memory, running Windows 10. The computational times of the algorithm are tabulated in **Table 2** and **Figure 1**. In a single run, we compute the price of 10 options, that is, $N_O = 10$. The pricing accuracy is gauged using the root mean square error (rmse):

$$\text{rmse} = \sum_{j=1}^{N_O} \left(\frac{P_{FFT}^j - P_{Monte}^j}{P_{Monte}^j} \right)^2, \quad (10)$$

where P_{Monte} represents the benchmark price computed using Monte Carlo method with 1000000 simulations and 1000 time step and P_{FFT} is the price from two-dimensional FFT.

S_1	S_2	r	T	d_1	d_2
100	96	0.1	1.0	0.05	0.05

Table 1.
Market pricing parameters.

Grid points: 2^N	Pricing accuracy	Single core (s)	GPU parallel (s)	GPU speed factor
(a) 2d-GBM model: $\sigma_1 = 0.2, \sigma_2 = 0.1, \rho = 0.5$				
8	1.10E-04	2.671126	0.213765	12.49561902
9	4.29E-05	10.155822	0.254708	39.87241076
10	1.97E-05	39.884087	0.650358	61.32635718
11	1.06E-05	157.833762	2.422815	65.14478489
12	6.19E-06	317.858516	9.474695	33.54815284
13	3.80E-06	1316.938937	38.519949	34.18849119
(b) 2d-NIG model: $\mu_1 = 0, \mu_2 = 0, \alpha = 6.20, \delta = 0.150, \beta_1 = -3.80, \beta_2 = -2.50, \rho = 0, \mu_1 = 0, \mu_2 = 0$ and $\Delta = 1$				
8	1.20E-04	2.528521	0.21573	11.72087536157
9	5.21E-05	9.768831	0.31099	31.41244621944
10	3.00E-05	39.372496	1.03477	38.04969824066
11	2.12E-05	162.069306	4.57217	35.44691939427
12	1.72E-05	651.850506	19.92393	32.71695934733
13	1.50E-05	2592.307476	83.77870	30.94232156861

Table 2.
Computational accuracy and processing times (a) 2d-GBM, (b) 2d-NIG.

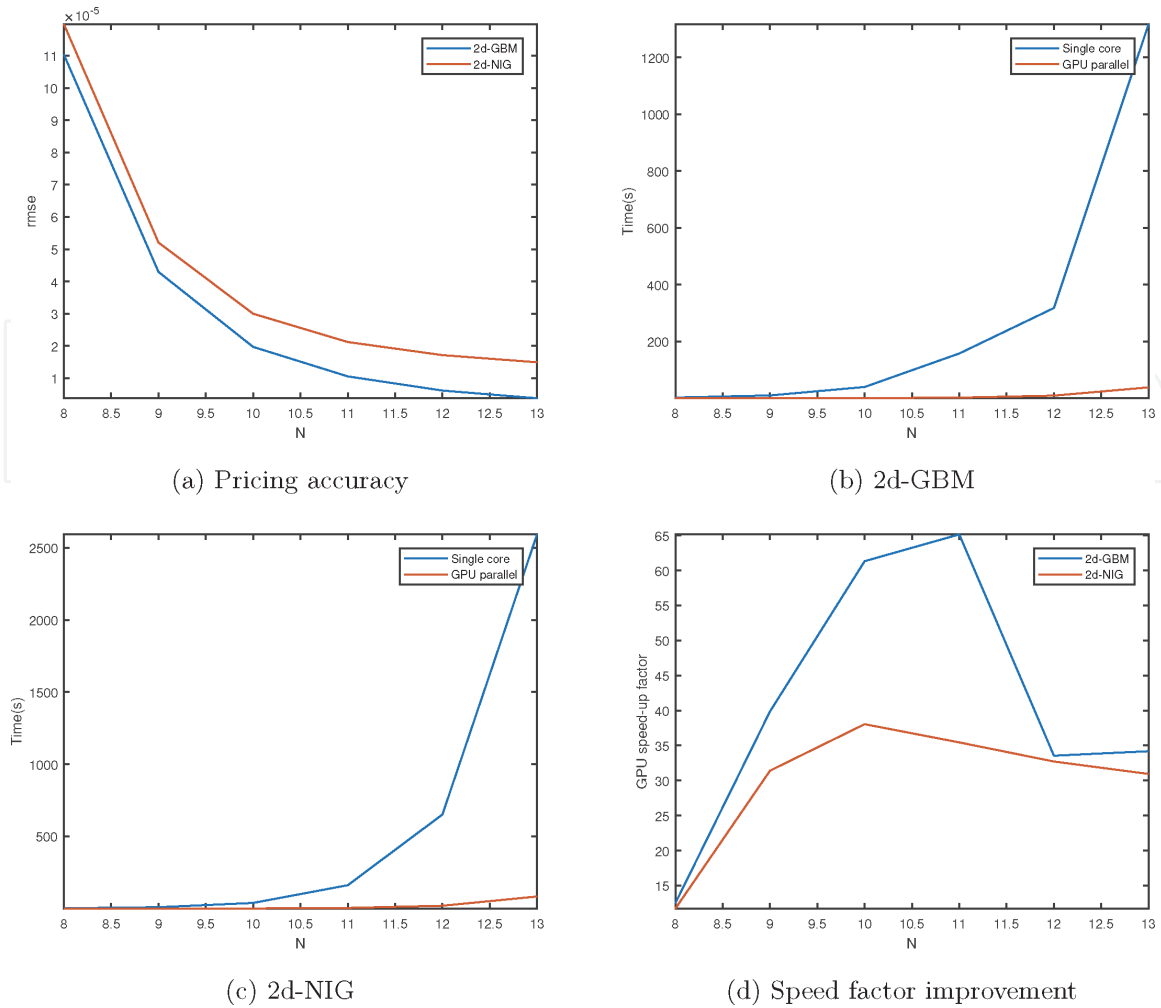


Figure 1. Numerical results for spread option pricing.

From **Table 2**, we can see that the optimal values of N (in terms of computation efficiency) are in the middle of the tested range, 9 and 10 for both the 2d-GBM and 2d-NIG models. The decline in performance for larger values of N is due to the increased memory requirements. When compared to 2d-GBM, 2d-NIG seems to have less of an increase in performance when executed on GPU, which could be because it is more computationally heavy (such as calculating the characteristic function $\Phi_{NIG}(u; T)$, which involves two square root and exponential calculations, as opposed to simply one in the GBM model).

4. Conclusion

In this work, we built on the literature on fast and accurate pricing of spread options based on two-dimensional FFT method using parallel computation. We examined the effectiveness of this approach by comparing the computational times of CPU and GPU implementations of the FFT Spread Option Pricing Algorithm in MATLAB. We have taken benchmark prices from Monte Carlo simulations with 1000000 paths and 100 discretization time steps. Our results decisively conclude that the execution of the algorithm on a GPU significantly improves computational performance, decreasing the time taken to run by a factor of up to almost 60x. Considering how common spread options are in the financial market, a faster way to price these securities means increased efficiency in transactions involving spread options, and the FFT algorithm implemented for this project also vastly improves

the accuracy of spread option pricing. This approach is very useful to accurate calibration of spread options which is recognized to be a challenging exercise.

As an extension to this work, one could develop a 3-asset spread option pricing algorithm using the 3D Fast Fourier Transform Algorithm. Such a scheme, while computationally heavy, could be rendered more efficient by harnessing the power of GPUs through the tools available in MATLAB.

IntechOpen

Author details

Shiam Kannan¹ and Mesias Alfeus^{2,3*}


1 Ridge High School, NJ, USA

2 University of Wollongong, Wollongong, Australia

3 University of Stellenbosch, South Africa

*Address all correspondence to: malfeus@uow.edu.au; mesias@sun.ac.za

IntechOpen

© 2020 The Author(s). Licensee IntechOpen. This chapter is distributed under the terms of the Creative Commons Attribution License (<http://creativecommons.org/licenses/by/3.0>), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited. 

References

[1] Kirkpatrick S, Gelatt C, Vecchi M. Optimization by simulated annealing. *Science*. 1983;1(23):671-680

[2] Dempster M, Hong S. Spread option valuation and the fast Fourier transform. In: *Mathematical Finance-Bachelier Congress*. Vol. 2000. 2002. pp. 203-220

[3] Hurd T, Zhou Z. A Fourier transform method for spread option pricing. *SIAM Journal on Financial Mathematics*. 2010; 1:142-157

[4] Alfeus M, Schlögl E. On spread option pricing using two-dimensional Fourier transform. *International Journal of Theoretical and Applied Finance*. 2019;22(5)

[5] Hurd T, Zhou Z. A Fourier transform method for spread option pricing. *SIAM Journal on Financial Mathematics*. 2010; 1(1):142-157

IntechOpen