Parallel Option Pricing with BSDE method on GPU

Bin Dai
School of Computer Science and Technology
Shandong University
Jinan 250101, P.R. China
daviddaibin@hotmail.com

Ying Peng
School of Computer Science and Technology
Shandong University
Jinan 250101, P.R. China
pengy@sdu.edu.cn

Bin Gong
School of Computer Science and Technology
Shandong University
Jinan 250101, P.R. China
gb@sdu.edu.cn

***Abstract*—The development of the hardware changes program structure. Now the Graphic Processing Unit (GPU) has evolved into an extremely flexible, powerful and cost-efficient processor, which is specialized for compute intensive, massively data parallel computation. In the field of financial derivatives pricing and risk management, the Backward Stochastic Differential Equation (BSDE) is a robust tool. The aim of this paper is the efficient use of GPU acceleration for option pricing with BSDEs. Experimental results show that a GPU can achieve a superior performance, greater than 230x, compared with the CPU-only case.**

Keywords—GPU, High Performance Computing, Option Pricing, BSDE, Parallel

I. INTRODUCTION

The option pricing is a very important problem encountered in financial engineering, since the creation of organized option trading in 1973. Over the past two decades, as more computation has been applied to finance-related problems, finding efficient ways to implement option pricing models on modern architectures has become more important. Moreover, most complex mathematical models cannot be solved by analytical formulas. We must use numerical methods, which need much more computing efforts. The computation time may be very long to get relatively accurate results, thus acceleration becomes an efficiently way to solve the numerical methods.

NVIDIA® CUDA™ is a general purpose parallel computing architecture that leverages the parallel compute engine in NVIDIA GPUs to solve many complex computational problems in a fraction of the time required on a CPU. It includes the CUDA Instruction Set Architecture (ISA) and the parallel compute engine in the GPU. To program with the CUDA architecture, developers can, today, use C, one of the most widely used high-level programming languages, which can then be run at great performance on a CUDA enabled processor. Programmable GPU has evolved into a highly parallel, multi-threaded, multi-core processor, with an outstanding computing power and high bandwidth of memory. Now lots of acceleration work is developed with GPUs.

In financial engineering area, the Black-Scholes formula [1] has been a major computing tool which measures profit and loss as well as option transaction risk for investors. However, the Black-Scholes formula is established on the basis of six assumptions, and it is impossible to entirely satisfy these assumptions in the reality. Moreover, the Black-Scholes formula can hardly deal with non-linear problems. Thus different mathematical models are proposed to solve more and more complex financial problems. In [2], N.El Karoui et al. obtain the extension of the Black-Scholes formula by using BSDEs. Being different from the traditional Black-Scholes model, the BSDEs can handle with non linear problems. Now the BSDE theory is applied in financial area more and more frequently.

In this paper, we efficiently use the computing power of a GPU for solving option pricing problems with BSDEs by using NVIDIA's CUDA. We adopt a high accurate theta scheme method to solve the BSDEs. This method needs large amounts of calculation, and the computation time is very long for getting enough accurate result. Fortunately, the computation process of this method meets the conditions for parallel computing with GPU. We analyze the serial algorithm and design the acceleration strategy. We also conduct performance experiments to compare the running time between the GPU implementation and the CPU implementation and analyze factors affecting the GPU acceleration.

The remainder of this paper is organized as following. In Sect. II, we introduce related parallelization work in financial computing area. Sect. III presents the application of BSDE in option pricing and the numerical method for solving the BSDEs. In Sect. IV, we describe the GPU acceleration strategy according to the special structure of the numerical model. Sect. V shows the experimental results. Finally we give the conclusion in Sect. VI.

II. RELATED WORK

In the financial field, the time factor is very important. Financial institutions spend considerable amounts of money in their information systems, because any delay in information processing could cause potential economic losses. In order to get better computation efficiency and accuracy, lots of work has been done in this area.

With the emergence of computer clusters and multi-core CPUs, parallelization techniques have been widely used in financial computing applications. In [4], R. K. Thulasiram et

978-0-7695-4313-0/10 $26.00 © 2010 IEEE
DOI 10.1109/GCC.2010.47

al. study multithreaded algorithms for pricing American Style options including both vanilla and complicated options. They propose two different algorithms according to the structure of the binomial lattice and compare the running time results between them on the Efficient Architecture for Running THreads(EARTH) platform. In [5], H. Sak et al. focus on a partial differential equation (PDE) approach to price the Asian option. They solve the PDE in parallel by using both explicit and Crank–Nicolson's implicit finite-difference methods and evaluate the performance in massively parallel processors (MPP). In [6], A. V. Gerbessiotis et al. describe a latency-tolerant parallel algorithm for the multiplicative binomial tree option pricing model and analyze the performance of the algorithm under the BSP model of computation.

The Monte Carlo (MC) simulation is widely used in option pricing. However, the computational effort can be substantial when high accuracy is required. In [7], G. Pauletto shows the parallelization potentials of the MC methods and briefly presents several methods for obtaining pseudo-random numbers on a parallel computer. In [8], J. W. L. Wan et al. develop parallel algorithms for pricing American options on multiple assets. Their methods are based on the low discrepancy (LD) mesh method which combines the quasi-Monte Carlo technique with the stochastic mesh method. The algorithms are implemented using MPI and near optimal speedup results are presented.

Presently, because of the high speed and low price of GPUs, more and more research work begins to focus on GPU accelerated financial computing problems. In [9], M. Lee et al. parallelize a computationally demanding financial application based on Monte-Carlo methods with a single GPU and achieve a superior performance. In [10], L. A. Abbas-Turki et al. efficiently use CPU and GPU clusters for a general path-dependent exotic European pricing and compare their speed and energy consumption. Using the Fourier Space Time - stepping (FST) method, V. Surkov implements option pricing algorithms on a GPU and obtains good efficiency [11]. V. Agarwal et al. optimize European Option and Collateralized Debt Obligation pricing algorithms through Monte Carlo simulation on IBM's Cell engine and GPU in [12]. Compared with the use of RapidMind SDK, both algorithms obtain better efficiency.

Although much parallelization work has been done in financial computing area, little work among them is based on the BSDE. The aim of our work is to explore the opportunity of GPU accelerated option pricing with the BSDE.

III. OPTION PRICING WITH BSDES

The BSDE is proved to be a robust tool for derivatives pricing and risk hedging in financial area. The mathematical formula for a BSDE is shown as following:

$$\begin{aligned} -dY_t &= f(Y_t, Z_t, t)dt - Z_t dW_t, t \in [0,T] \\ Y_T &= \xi \end{aligned} \tag{1}$$

where $W_t = (W_t^1, \cdots, W_t^d)^*$ is a d-dimensional Brownian motion defined on some complete, filtered probability space $(\Omega, F, P, \{F_t\}_{0 \le t \le T})$, and the notation * is the transpose operator for matrix or vector. $Y_T = \xi$ is the terminal condition of the BSDE and ξ is a F_T measurable random variable. The solution of the BSDE is a couple of variables (Y_t, Z_t), and the generating function $f(\bullet)$ is a function of (Y_t, Z_t).

While applied in option pricing, the solution Y_t represents the option price at time t, and Z_t helps determine the risk hedging strategy. The generating function $f(\bullet)$ is:

$$f(t, y, z) = -ry - \sigma^{-1}(\mu - r)z \tag{2}$$

where r is the return rate, σ represents the volatility, and μ is the expected return rate.

For the European options which can be exercised only at its expiration, the terminal condition of the BSDE is given at the expiration time T by:

$$\begin{aligned} Y_t &= \xi = \phi(W_T) \\ &= \max(S_T - X, 0) \quad (call\ option) \\ &= \max(X - S_T, 0) \quad (put\ option) \end{aligned} \tag{3}$$

where $\phi(.)$ is a function of the Brownian motion W_T and it represents the payoff of the option holder at the expiration time T. S_T is the asset price at the expiration time and X is the strike price.

In order to calculate the option price with the BSDE, we adopt the theta scheme method [3], which is of high accuracy and suitable for the parallel implementation with GPU. The theta scheme method discretizes the continuous BSDEs on time-space discrete grids. On each grid point, it uses the Monte Carlo method to approximate mathematical expectations, and uses space interpolations to compute values at non-grid points.

By performing iteratively the full discretization theta scheme method, we could get the value of Y_t and Z_t on every grid point, as shown in equation (4).

$$\begin{aligned} y_i^n &= E_{h,t_n}^{t_n,x_i}[y^{n+1}] + \Delta t_n\{(1-\theta_1^n)E_{h,t_n}^{t_n,x_i}[f(t_{n+1}, y^{n+1}, z^{n+1})] + \theta_1^n f(t_n, y_i^n, z_i^n)\} \\ 0 &= E_{h,t_n}^{t_n,x_i}[y^{n+1}\Delta W_{t_{n+1}}] + \Delta t_n(1-\theta_2^n)E_{h,t_n}^{t_n,x_i}[f(t_{n+1}, y^{n+1}, z^{n+1})\Delta W_{t_{n+1}}] \\ &\quad - \Delta t_n\{(1-\theta_2^n)E_{h,t_n}^{t_n,x_i}[z^{n+1}] + \theta_2^n z_i^n \end{aligned} \tag{4}$$

where i indicates the space layer, n indicates the time layer on the grid. E[X] denotes the conditional mathematical expectation of the random variable X, which can be approximated with the Monte Carlo method. We show in equation (5) the approximation of $E_{h,t_n}^{t_n,x_i}[y^{n+1}]$, other conditional expectations can be gotten similarly, shown in detail in [[3]].

$$E_{h,t_n}^{t_n,x_i}[y^{n+1}] = \frac{\sum_{k=1}^{N_E} I_h y^{n+1}(\hat{x}_i^k)}{NE} \tag{5}$$

In (5), $\hat{x}_i^k$ can be derived with $\hat{x}_i^k = x_i + \Delta_h^k W_{t_{n+1}}$. For each space point x_i, let $\Delta_h^k W_{t_{n+1}} (k = 1,2,\cdots,NE)$ be the NE times realizations of $\sqrt{\Delta t_n} N(0,1)$, where N(0,1) is a standard

normal distribution with mean zero and variance 1. $I_h u(\hat{x}_i)$ represents an interpolation approximation of the function u(x) at the space point $\hat{x}_i$ by using the values of u(x) at a finite number of the space points X_j's near the space point $\hat{x}_i$.

IV. PARALLEL ALGORITHM

We first describe the computation process of the algorithm. Then we present the parallel optimization strategy on GPU with CUDA according to the algorithm process.

A. Algorithm

According to the numerical algorithm represented in Sect. III, the whole process for option pricing can be divided into three steps.

Step1: constructing the time-space discrete grid, as shown in Figure 1. .

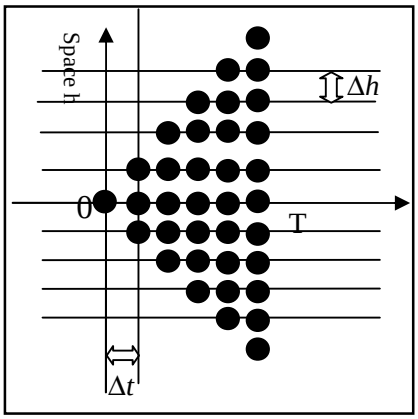

Figure 1. Time-Space Grid

We evenly divide the time period [0, T] into N time steps with interval $\Delta t = T/N$ to get N+1 time layers. The value of the space interval Δh is supposed to be the same as Δt. Then for every space point x_i, its value for calculating equation (5) can be derived with the value of Δh.

Step 2: calculating the terminal conditions which represent the possible option prices at the expiration time T (on time layer N) by using equation (3).

Step 3: calculating option price iteratively from the time layer N-1 to the time layer 0 with equation (4), until we get the value of (Y_0, Z_0), which represent (Y_t, Z_t) at the time layer 0.

The process for calculating the corresponding option price on an arbitrary grid point is shown in Figure 2. .

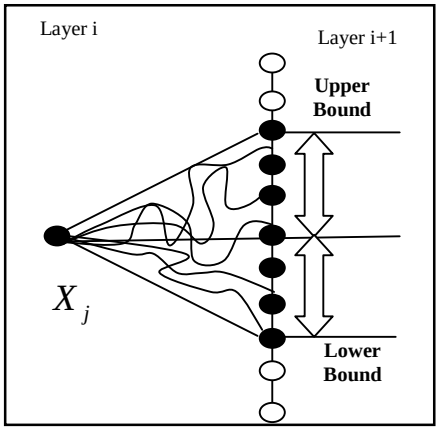

Figure 2. Calculation Process on a Grid Point

As mentioned in Sect. III, the Monte Carlo simulation is used to approximate the mathematical expectation E[X]. In Figure 2. , for each time of MC simulation, the geometric Brownian motion start from the space point X_j on time layer i, along the path from time layer i to time layer i+1, in this way we get the projection value of $\hat{x}_i^k$ on time layer i+1. By using space interpolation, $I_h u(\hat{x}_i)$ can then be calculated. We use cubic spline interpolation to achieve better accuracy. According to equation (5), after NE times of Monte Carlo simulations, we could get the value of E[X] and use them to compute equation (4). At last the solution of the BSDE (Y_0, Z_0) is achieved.

Therefore, for calculating the option price of a grid point X_j on time layer i, the grid points on time layer i+1 is needed. Without loss of generality, we define the upper and lower bound $C = c * \sqrt{\Delta t}$ for the geometric Brownian motion on a single time step, where c is a constant. The upper and lower bound could also be defined by the number of space points $Ps = C/\Delta h$. In this way, the number of space points on time layer i can be gotten by $M_i = i * Ps * 2$, and we define M_0=1.

B. Parallel Implementation

As shown in Sect. IV.A, the algorithm for option pricing with the BSDE can be performed with three steps. For step 1 and step 2, the construction of the time-space discrete grid and the calculation of the terminal conditions, we do not need large amount of calculation. To verify that, we measure the running time of step 1 and step 2 on a CPU. We let N and NE represent respectively the number of time steps and the number of Monte Carlo simulations. When (N, NE) are fixed at (64, 40000), the running time of step 1 and step 2 is only 0.281 ms. Comparing with 2,098,757 ms, which is the running time of all steps, the cost of step 1 and step 2 seem insignificant. Thus for these two steps, we implement them on CPU.

For the step 3, each option price corresponding to space point X_j at time layer t_i needs NE times of MC simulation. Thus step 3 is the most time-costing part in the serial algorithm. Moreover, on each time layer, the option price of different space points can be calculated completely independently. Thus step 3 is very suitable to be parallelized.

We implement the parallel algorithm on a GPU by using CUDA.

CUDA programs consist of GPU programs called kernel, thread block arrays executing kernels called grid, and SIMD thread group (block) which executes kernel programs. Threads within a block can cooperate among themselves by sharing data through the shared memory and synchronizing their execution to coordinate memory accesses.

To parallelize step 3 with GPU, we let one kernel function, which consists of all the threads, calculate all the grid points of one time layer. Each thread of CUDA calculates one grid point. The number of thread equals to the number of space point on time layer N. The parallelization strategy with CUDA is shown in detail in Figure 3. .

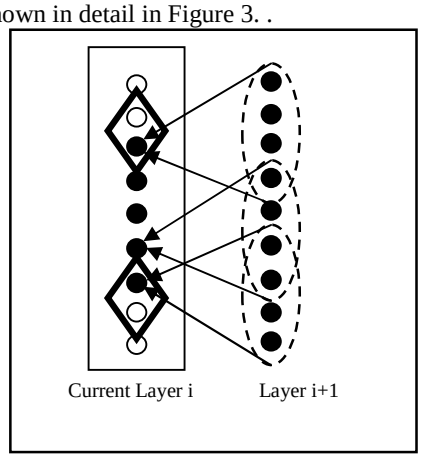

Figure 3. Parallelization Strategy with CUDA

In Figure 3. , the *rectangle* includes all the grid points in the current time layer i. The job of the kernel function is to calculate the option price of all the grid points on the current time layer. A *filled circle* represents a grid point which need to be calculated on the current time layer i. The job of an active thread is to calculate the option price of a grid point on the current time layer. The *ellipse* contains all the needed grid points on time layer i+1 for calculating the option price of a grid point on time layer i. When we process the calculation backwardly from time layer i+1 to time layer i, the number of grid point decreases. An *empty circle* represents a grid point which need not to be calculated on the current layer i. For these grid points, threads do not do any job.

Empty threads affect the efficiency. By optimizing our code, the impact is slight. In CUDA, Threads are assigned to Streaming Multiprocessors (SM) in block granularity. And SM schedules thread execution. So the best way to efficiently utilize GPU is to make all the threads in a block do the same job. When empty threads exist, the blocks consist of empty threads does not do any job. And empty threads wait for active threads in the blocks which consist of both empty and active threads when we synchronize threads. In our program, we recount and redefine the number of active threads before executing kernels. In this way, there are no empty threads affecting the efficiency.

V. EXPERIMENTAL RESULTS

The parallel algorithm is implemented with C and CUDA. Runtime experiments are performed on a GPU. The results are compared with the serial code on a CPU. We analyze our performance results from actual runs on an LANGCHAOYINGXIN NP3588 PC with an Inter(R) Core(TM) i7 920 (2.67GHz) CPU and 4GB DDR3 memory. The GPU is a NVIDIA Tesla C1060 with a total 4GB GDDR3 memory.

TABLE I. shows the performance of entire application comparison between the CPU implementation and the GPU implementation, including the running time and speedups. We vary the number of time steps N and the number of MC simulations NE. As indicated in [3], higher accuracy will be achieved when the value of (N, NE) increases.

TABLE I. COMPARISON BETWEEN CPU RUNNING TIME AND GPU RUNNING TIME

N	NE	Time on CPU	Time on GPU	Speed up
4	1000	0.041s	0.035s	1.71x
8	4000	0.893s	0.117s	7.63x
16	10000	13.311s	0.468s	28.44x
32	20000	176.753s	1.832s	96.48x
64	40000	2098.757s	9.193s	228.30x
128	80000	22934.438s	98.672s	232.43x

The number of threads per block is fixed at 128. As shown in table 1, when the number of N and NE are small (such as N=4, NE=1000), the speedups seem insignificant. The reason is that there are not enough threads to make all the SMs in GPU busy. Thus most of the running time is spent on data transfer between CPU and GPU. When we increase the value of (N, NE), the CPU time increase greatly, and the speedups also rise obviously. When (N, NE) = (128, 80000), the speedup achieves 232.43X.

Figure 4. shows the running time on GPU for (N, NE) = (32, 20000) and (64, 40000) when varying the number of threads per block.

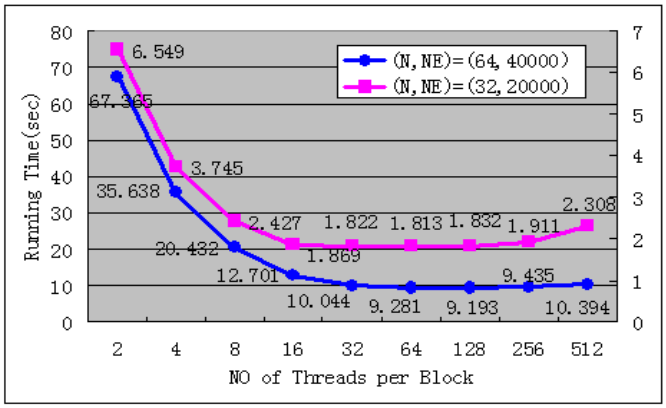

Figure 4. Running Time in Different Thread

In the programming mode of CUDA, SMs schedule thread execution in block granularity. Each block is executed as 32-thread warps. Thus warps are scheduling units in SM. Therefore, when the thread count is less than 32, SMs are not efficiently utilized. The maximum performance is observed when the number of threads per block is more than 32(64 or 128). Increasing the thread count in block further doesn't gain any more performance.

VI. CONCLUSION

In this paper, we design an efficient parallel algorithm for option pricing based on the BSDEs with GPU. To get the solution of the BSDEs, we adopt a high accurate theta scheme method. We conduct the performance analysis by comparing the running time between GPU implementation and CPU implementation. We also analyze the effects of the thread number per block. The experimental results manifest optimistic speedups. The parallelized program using CUDA showed greater than 230x performance gain compared with the serial run time on a CPU. Max performance was observed when we use 64 or 128 threads in one block. The experimental results show that the GPU architecture is well suited for solving the BSDEs in parallel. In the future, we will focus on parallelization works for more complex financial computing problems, such as path-dependent and multi-dimensional option pricing.

ACKNOWLEDGMENT

This work is supported by grants from the National High Technology Research and Development Program of China (863 Program) (No. 2006AA01A113) and Science and Technology R & D Program of Shandong Province (No. 2009GG10001001)

REFERENCES

[1] F. Black, and M. Scholes, "The pricing of options and corporate liabilities," Journal of Political Economy 81, pp. 637–654, 1973.

[2] N.El. Karoui, S. Peng, and M.C. Quenez, "Backward stochastic differential equations in finance," Mathematical Finance, Vol. 7, No. 1,pp. 1-71, 1997.

[3] W. Zhao, L. Chen, and S. Peng, "A New Kind of Accurate Numerical Method for Backward Stochastic Differential Equations," SIAM Journal on Scientific Computing, Vol. 28, Issue 4, pp. 1563-1581,2006.

[4] R. K. Thulasiram, L. Litov, H. Nojumi, C. T. Downing, and G. R. Gao, "Multithreaded algorithms for pricing a class of complex options," Proceedings 15th International Parallel and Distributed Processing Symposium, pp. 18, 2001.

[5] H. Sak, S. Özekici, and İ.Bodurog˜lu, "Parallel computing in Asian option pricing," Parallel Computing, Vol.33, Issue 2,pp. 92-108, 2007.

[6] A. V. Gerbessiotis, "Architecture independent parallel binomial tree option price valuations," Parallel Computing, Vol.30, Issue 2, pp. 301-316,2004.

[7] G. Pauletto, "Parallel Monte Carlo Methods for Derivative Security Pricing," Numerical Analysis and Its Applications, Vol. 1988, pp. 650-657, 2001.

[8] J. W. L. Wan, ,K. Lai, A. W. Kolkiewicz, and Ken Seng Tan, "A Parallel Quasi-Monte Carlo Approach to Pricing," International Journal of High Performance Computing and Networking, Vol. 4, pp. 321-330, 2006.

[9] M. Lee, J. Jeon, J. Bae, and H. S. Jang, "Parallel Implementation of a Financial Application on a GPU," Proceedings of the 2nd International Conference on Interaction Sciences, pp. 1136-1141,2009.

[10] L. A. Abbas-Turki, S. Vialle, B. Lapeyre, and P. Mercier, "High Dimensional Pricing of Exotic European Contracts on a GPU Cluster, and Comparison to a CPU Cluster". 2009 IEEE International Symposium on Parallel&Distributed Processing, Vols. 1-5, pp. 2414-2421, 2009.

[11] V. Surkov, "Parallel option pricing with Fourier Space Time-stepping method on Graphics Processing Units," 2008 IEEE International Symposium on Parallel&Distributed Processing, Vols. 1-8 , pp. 2850-2856, 2008.

[12] V. Agarwal, L. K. Liu, and D. Bader, "Financial modeling on the Cell Broadband Engine," 2008 IEEE International Symposium on Parallel&Distributed Processing, Vols. 1-8 , pp. 1957-1968, 2008.