Reconfigurable Control Variate Monte-Carlo Designs for Pricing Exotic Options

Anson H.T. Tse, David B.Thomas, K.H. Tsoi, Wayne Luk
Department of Computing
Imperial College London, United Kingdom
Email: {htt08,dt10,khtsoi,wl}@doc.ic.ac.uk

***Abstract*—Exotic options are financial derivatives which have complex features including path-dependency. These complex features make them difficult to price, as only computationally intensive Monte-Carlo methods can provide accurate prices. This paper proposes an FPGA-accelerated control variate Monte-Carlo (CVMC) framework for pricing exotic options. An optimised implementation of arithmetic Asian option pricing under this framework in a Virtex-5 xc5vlx330t FPGA at 200MHz is 24 times faster than a multi-threaded software implementation on a Xeon E5420 at 2.5GHz; it is also 2.4 times faster than the Tesla C1060 GPU at 1.3 GHz.**

I. Introduction and Related Works

Financial analysis and pricing applications are often computationally intensive, so there has been much interest in FPGA-accelerated option pricing. Numerical techniques, such as lattice and Monte-Carlo methods, are used for option valuation when there is no closed-form solution [1], [2], [3].

However, the use of control variate technique has not been reported. Control variate is one of the variance reduction techniques which aims at reducing variance as well as computation time for a given accuracy for Monte-Carlo simulation [4]. This paper explores the control variate Monte-Carlo method in FPGAs for exotic option pricing.

Our contributions are:

- A parallel hardware framework using the control variate Monte-Carlo (CVMC) method for pricing exotic options.
- A detailed implementation of arithmetic Asian option pricing using CVMC method under this framework.
- Evaluation of the FPGA and GPU implementations versus a highly optimized multi-threaded implementation on Intel Xeon 2.5GHz CPU, showing 24 times speedup for the FPGA, and 10 times for the GPU.

II. Background

An option is a type of financial instrument which provides the owner of the option with the right, but not the obligation, to buy or sell an underlying asset such as a stock or bond at some point in the future.

For simple European call options, the owner can exercise only at the expiry date. If the underlying asset price *S* at expiry date is higher than the strike price *K*, the owner can profit by buying the stock at the lower price *K* from the option issuer and then immediately selling it at the higher price *S* in the market, providing a gain of $(S-K)$. If the underlying asset price is lower than the strike price $(S<K)$, then the gain is zero because the option will not be exercised.

The payoff of European call option on expiry is:

$$P_{call} = \max(S-K, 0). \tag{1}$$

For path-dependent exotic options, the payoff on expiry depends on the entire underlying asset movement path. For example, the payoff of lookback call options is:

$$P_{call} = \max(\max(S(t_0), S(t_1), ..., S(t_n)) - K, 0) \tag{2}$$

where $t_1..t_n$ are the times at which the asset price is observed, and $S(t)$ is the asset price at time t.

The current price of a European option can be calculated with a closed-form solution called the Black-Scholes Equation [5]. However, there is no such solution for many exotic options (e.g. arithmetic Asian options), due to their highly path-dependent properties. Monte-Carlo methods are one way of solving this problem. The idea of Monte-Carlo pricing is to generate a large number of random paths for each probabilistic variable, then take the average of the payoffs to get the estimated expected payoff $\bar{x}$.

The 99% confidence interval of the payoff is given by:

$$\textit{Payoff}_{99\%} = \left[\bar{x} - 2.58\frac{\sigma_x}{\sqrt{N_{mc}}}, \bar{x} + 2.58\frac{\sigma_x}{\sqrt{N_{mc}}}\right] \tag{3}$$

where N_{mc} is the number of simulation and σ_x is the standard deviation of payoff x. The 99% confidence interval means the actual value has a chance of 99% to be inside the range of the interval.

Therefore, to improve accuracy (reduce confidence interval length) by a factor of n, the number of Monte-Carlo simulations has to be increased by a factor of n^2, which is the reason for the high computational complexity of Monte-Carlo methods.

Variance reduction techniques have been proposed to improve accuracy by reducing the variance instead of increasing the number of simulations. The control variate method is a variance reduction technique which estimates the target value using a control variable y. The variable $\bar{y}$ is computed using the same set of random data of the computation of $\bar{x}$. The true expected value of $E(y)$ should be calculable using a closed-form solution. The control variate estimator of x_c is given by:

$$x_c = x + c(y - E(y)) \tag{4}$$

TABLE I
THE *init()*, *update()* AND *calculate()* FUNCTION FOR SOME EXAMPLE CALL OPTIONS

	init(x,o)	*update(x,S,o)*	*calculate(y,x,o)*
Asian	$x \leftarrow S_0$	$x \leftarrow x + S$	$y \leftarrow \max(0, \frac{x}{(n+1)} - K)$
Lookback	$x \leftarrow S_0$	$x \leftarrow \max(x, S)$	$y \leftarrow \max(0, x - K)$
Barrier	$x_1 \leftarrow 0$; $x_2 \leftarrow S_0$	*if* $S > B, x_1 \leftarrow 1$; $x_2 \leftarrow S$	*if* $x_1 = 1, y \leftarrow 0$, *else* $y \leftarrow \max(0, x_2 - K)$
European	$x \leftarrow S_0$	$x \leftarrow S$	$y \leftarrow \max(0, x - K)$

Algorithm 1 Control variate Monte-Carlo pricing algorithm

1: (Let o be all the option parameters)
2: *t_sum* = 0 //target option payoff sum
3: *c_sum* = 0 //control option payoff sum
4: *c2_sum* = 0 //square of control option payoff sum
5: *tc_sum* = 0 //target times control option payoff sum
6: **for** $i = 1$ to N_{mc} **do**
7: $S = S_0$
8: *t_temp* $\leftarrow$ *init*$_t$*(t_temp,o)* ; *c_temp* $\leftarrow$ *init*$_c$*(c_temp,o)*
9: **for** $i = 1$ to *Steps* **do**
10: $W \leftarrow$ *NextRandomNumber()*
11: $S \leftarrow Se^{drift+vsqrdt \times W}$
12: *t_temp* $\leftarrow$ *update*$_t$*(t_temp,S,o)*
13: *c_temp* $\leftarrow$ *update*$_c$*(c_temp,S,o)*
14: **end for**
15: *t* $\leftarrow$ *calculate*$_t$*(t, t_temp,o)* ; *c* $\leftarrow$ *calculate*$_c$*(c, c_temp,o)*
16: *t_sum* $\leftarrow$ *t_sum + t* ; *c_sum* $\leftarrow$ *c_sum + c*
17: *c2_sum* $\leftarrow$ *c2_sum* + c^2 ; *tc_sum* $\leftarrow$ *tc_sum* + $c \times t$
18: **end for**
19: $E(t) \leftarrow$(*t_sum* / N_{mc}) ; $E(c) \leftarrow$(*c_sum* / N_{mc})
20: $E(c^2) \leftarrow$(*c2_sum* / N_{mc}) ; $E(tc) \leftarrow$(*tc_sum* / N_{mc})
21: *Var(c)* $\leftarrow E(c^2) - (E(c))^2$; *Cov(t,c)* $\leftarrow E(tc) - E(t)E(c)$
22: *True(c)* $\leftarrow$ *control_option_true_equation(o)*
23: *adjustment* $\leftarrow -\frac{Cov(t,c)}{Var(c)}(E(c) - True(c))$
24: $E_{cv}(t) \leftarrow E(t) + adjustment$
25: *TargetOptionPrice* $\leftarrow e^{-rT}E_{cv}(t)$

Therefore, $E(x_c) = E(x)$, and $Var(x_c)$ is minimized by choosing $c = -Cov(x, y)/Var(y)$, such that

$$Var(x_c) = Var(x) - Cov(x, y)^2/Var(y) \tag{5}$$

As a result, the variance of the estimated value is reduced and thus the length of confidence interval is shortened. The higher the correlation between the control variable and the target estimating variable, the more effective is the control variate method. The required number of simulations can be significantly reduced for a given confidence interval.

This CVMC method can be applied to exotic option pricing. Apart from simulating the payoff of the target exotic option, the payoff of a correlated control option is also simulated at the same time. The only condition is that the closed-form solution of the control option must be known. The actual performance difference of using the control variate method over the pure Monte-Carlo method will be discussed in Section V.

III. PARALLEL ARCHITECTURE

Under Black Schole's model the stock price movement is governed by a geometric Brownian motion process, and the stock price is given by the equation:

$$S_{i+1} = S_i e^{((r-v^2/2)\delta t + v\sqrt{\delta t}W)} \tag{6}$$

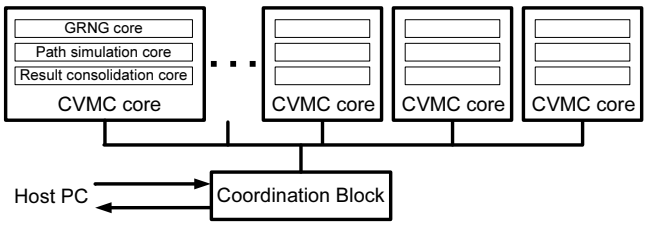

Fig. 1. Overall hardware architecture.

where r is the interest rate, v is the volatility of the underlying stock price, δt is the time period between two time steps, W is a Gaussian random number, S_i is the underlying stock price at step i and S_{i+1} is the underlying stock price at step $i + 1$. We define the equations:

$$drift = \left(r - v^2/2\right)\delta t, \qquad vsqrdt = v\sqrt{\delta t} \tag{7}$$

such that

$$S_{i+1} = S_i e^{drift+vsqrdt \times W} \tag{8}$$

The values of *drift* and *vsqrdt* can be precomputed in advance, so that the stock price can be simulated with these two static values.

A one-pass variance and covariance computation method is used. The control variate Monte-Carlo option pricing algorithm is shown in Algorithm 1. We define *t_temp* and *c_temp* to be the temporary updating variables for the target and control option payoffs calculation. They are initialized by option specific initialization functions *init()* at the beginning of each path simulation (line 8). They are then updated with the corresponding updating functions *update()* after each step of stock price movement (line 12-13). The corresponding option payoffs of the target and control options are calculated by their option specific functions *calculate()* after a completed path is simulated (line 15). The sum of the target option payoff (*t_sum*), the sum of control option payoff (*c_sum*), the sum of the square of control option payoff (*c2_sum*) and the sum of the target option payoff times control option payoff (*tc_sum*) are accumulated correspondingly (line 16-17).

In the final stage of the algorithm, the simulated target option payoff, control option payoff, variance of the control option payoff, covariance of target and control option payoff are computed from *t_sum*, *c_sum*,*c2_sum* and *tc_sum* (line 19-21). The true value of the control option payoff is calculated with the closed-form equation (line 22). The final target option price is then obtained with the control variate adjustment and discounted backward to present time (line 23-25).

Different types of exotic options have different *init()*, *update()* and *calculate()* functions. Table I shows instances of these functions for some exotic and European options.

Our overall hardware design architecture is presented in Fig. 1. There are two main types of components in the design: one or more identical CVMC cores, and a single shared Coordination Block (CB). Each CVMC core contains a Gaussian random number generator (GRNG) core, a path simulation core and a result consolidation core; it is capable of generating random asset price paths, calculating payoffs

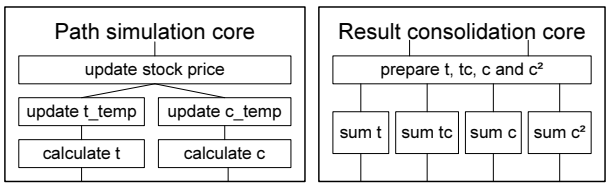

Fig. 2. Block diagram of path simulation core and result consolidation core.

of the target option and control option, and accumulating the payoffs and payoffs related statistical result. In other words, each CVMC core is capable of executing the main for-loop in Algorithm 1. Multiple identical CVMC cores are instantiated to make maximum use of the device. The total number of simulations required is distributed equally to each CVMC core.

The block diagrams of the path simulation core and result consolidation core inside the CVMC core are shown in Fig. 2. As pipelined operators are used and the output of all "update blocks" and "sum blocks" will be fed back to the input of themselves, there is a pipelined loop for each of them. The number of the pipelined stages must be identical for all the pipelined loops in order to guarantee a consistent computation schedule. Let p be the maximum number of pipeline stages for these pipelined loops. Pipelined registers are added to ensure the number of pipelined stages of all loops equal to p. We simulate a batch of p paths at the same time in this pipelined fashion. Therefore, p path simulations are completed every $p \times steps$ cycles. Let N_{batch} be the required number of batches, N_{mc} be the required number of Monte-Carlo simulations and C be the number of CVMC cores in the hardware. N_{batch} is defined as:

$$N_{batch} = \left\lceil \frac{N_{mc}}{p \cdot C} \right\rceil \tag{9}$$

The Coordination Block (CB) manages the CVMC cores, allowing them to work in parallel to price the same option. The CB is also responsible for communicating with the external controller, for example a PC. With the precious timing control, all the *t_sum*, *c_sum*, *c2_sum* and *tc_sum* values computed in CVMC cores will be sent back to CB, and are transferred to the external host for post-processing. The Gaussian random number generators in CVMC cores are also initialized by the CB. Different sequences of bits are sent to different Gaussian random number generators as the random seeds.

IV. Implementation of Asian Options Pricing

This section presents the implementation of CVMC arithmetic Asian option pricing using our designed hardware architecture. There is no closed-form solution for arithmetic Asian options and pricing them fast and accurately is a challenging problem in finance. Therefore arithmetic Asian options are perfect candidates to be priced using the hardware accelerated CVMC framework. European options are chosen as control options, as they have a closed-form solution.

The payoff of arithmetic Asian options at expiry time is:

$$payoff = \max\left(0, \frac{1}{n+1}\sum_{i=0}^{n} S_i - K\right). \tag{10}$$

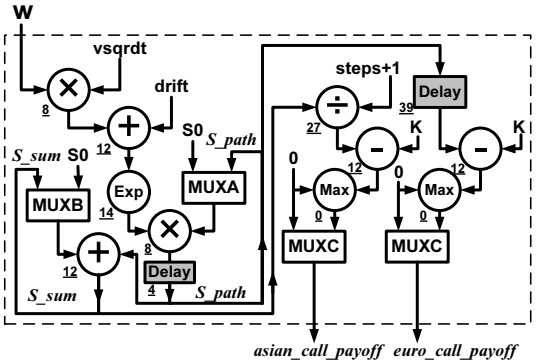

Fig. 3. Architecture of the price movement path simulation core.

TABLE II
XC5VLX330T FPGA RESOURCE CONSUMPTION

	10 CVMC Cores		16 pure MC Cores	
Resource	**Used**	**%**	**Used**	**%**
Slices	44,118	85%	41,968	80%
FFs	130,195	62%	110,789	53%
LUTs	79,587	38%	95,749	46%
RAM	10	3%	16	4%
DSP48Es	180	93%	192	100%

The *init()*, *update()* and *calculate()* functions of arithmetic Asian call options and the controlling European options are shown in Table I. The architecture of the path simulation core is shown in Fig 3.

The dynamic input parameter is the Gaussian random number W generated by piecewise linear generation method [6]. The floating-point operators used are from Xilinx Floating-Point Operator 4.0. The small underlined number near each operator is the number of pipeline stages (latency) of that operator.

As discussed in the Section III, a batch of p payoff results (Asian call payoff and European call payoff) are generated for every $p \times steps$ cycles and passed to the result consolidation core. The product of Asian and European call payoff, and the square of the European call payoff, are computed with two multipliers. These results are accumulated until the completed number of batches reaches N_{batch}.

V. Result Evaluation

This section investigates the advantage of using the CVMC method over the pure Monte-Carlo method for arithmetic Asian option pricing on FPGA. We also compare the performance of the implementations against GPU and CPU.

Our FPGA implementations targeted a Xilinx xc5vlx330t FPGA chip on an Alpha Data ADM-XRC-5T2 card. We design our hardware architecture manually in VHDL to maximize performance. The design is synthesized, mapped, placed and routed using Xilinx ISE 10.1.03. Single precision floating point arithmetic is used. There are 10 CVMC cores in the design. Resource utilisation is summarised in Table II. We have also implemented arithmetic Asian option pricing using the pure Monte-Carlo method on xc5vlx330t. As the number of floating point operators is decreased, 16 pure MC cores can be fitted in an xc5vlx330t device.

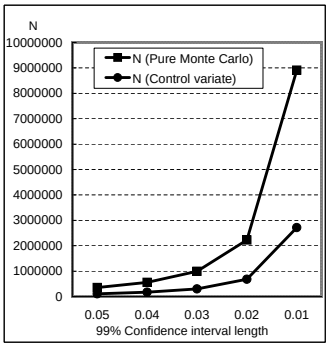

Fig. 4. The required number of simulation versus the 99% confidence interval length.

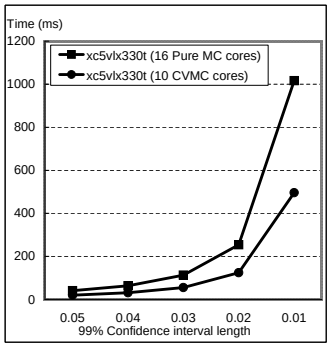

Fig. 5. The required computation time versus the 99% confidence interval length.

We choose an arithmetic Asian call option with parameters S_0*=100, K=105, v=0.15, r=0.1, T=1* and *steps=365*. The number of Monte-Carlo simulations is 1,000,000. The $Var(t), Var(c)$ and $Cov(t, c)$ of the tested arithmetic Asian call option is 33.47, 152.36 and 59.54 respectively. The 99% confidence interval of the option price is [3.392, 3.408]. The 99% confidence interval length is 0.016.

Fig. 4 shows the number of simulations required versus the 99% confidence interval length for using pure Monte-Carlo and control variate Monte-Carlo methods to price arithmetic Asian options. We can see that the number of simulations required for the pure Monte-Carlo method is 3.28 times more than the control variate Monte-Carlo method. One may consider that the pure Monte-Carlo core in FPGA consumes fewer resources, and therefore the degree of parallelism is higher. Our result also shows that 6 more cores can fit in the xc5vlx330t FPGA using pure Monte-Carlo. However, the reduced parallelism with fewer cores for the control variate method is more than compensated by the benefit of reduced variance. For a given confidence interval length (accuracy), the 10 CVMC FPGA cores is 2 times faster than the pure Monte-Carlo FPGA implementation with 16 cores as shown in Fig. 5.

The acceleration of the FPGA implementations and GPU implementations of Asian option pricing using CVMC method are in comparison to a reference software implementation. The reference PC used an Intel Xeon quad-core E5420 2.5GHz processor with 16GB RAM. The multi-threaded software implementation is written using C language with *gsl* library and compiled using Intel compiler *icc* with maximum speed optimization options which fully utilize *SSE2* parallelism. The targeted GPU is nVidia Tesla C1060 with 4GB of on board RAM. The GPU implementation is a translation from the software C code using CUDA API. The Mersenne Twister and Box-Muller transformation is used for Gaussian random number generation. The CUDA code is written in a way to ensure all GPU cores execute concurrently for random number generation and path simulation. The summary of the performance comparison is shown in Table III.

From the results, it can be seen that a speedup of 24 times over the CPU is achieved by xc5vlx330t FPGA. For the performance of the GPU, a speedup of 10 times is achieved by Tesla C1060. The xc5vlx330t FPGA is 2.4 times faster than the Tesla C1060 GPU.

TABLE III
PERFORMANCE OF THE ASIAN OPTION PRICING USING CVMC METHOD

	FPGA	**GPU**	**CPU**
Type	xc5vlx330t	Tesla C1060	Xeon E5420
Frequency	200MHz	1.3GHz	2.5GHz
Time(ms)	184ms	443ms	4,446ms
Speedup	24x	10x	1x
Max Power	29W	200W	80W

VI. CONCLUSION

This paper presents a high performance hardware architecture for exotic option pricing using the control variate Monte-Carlo (CVMC) method. To our knowledge, this is the first reported hardware implementation of CVMC method in the literature. A hardware implementation of arithmetic Asian option pricing using CVMC method is described, and its performance is compared with a GPU implementation using CUDA and a multi-threaded software implementation. By exploiting the efficient Gaussian random number generators, massive parallelism and highly pipelined datapath, our FPGA implementation outperforms a comparable software implementation by 24 times, and outperforms the GPU implementation by 2.4 times. In addition, the FPGA implementation consumes much less power than the GPU and software implementation. This improvement in speed and power consumption provides an attractive solution to financial institutions to shorten the pricing time and reduce costs by energy savings.

Future work includes the design of a distributed financial computation framework in a heterogeneous cluster consisting of FPGAs, GPUs and CPUs. The application of CVMC method to other computation problems will also be investigated.

ACKNOWLEDGMENT

The support of the Croucher Foundation, UK EPSRC, AlphaData, HiPEAC network of Excellence and Xilinx is gratefully acknowledged.

REFERENCES

[1] A. H. T. Tse, D. B. Thomas, and W. Luk, "Accelerating quadrature methods for option valuation," in *Proc. IEEE Symp. on FPGAs for Custom Computing Machines*, 2009.

[2] G. Zhang, P. Leong, C. Ho, K. Tsoi., D.-U. Lee, C. Cheung, R. Cheung, and W. Luk, "Reconfigurable acceleration for Monte-Carlo based financial simulation," in *Proc. Int. Conf. on Field-Programmable Technology*. IEEE, 2005, pp. 215–224.

[3] X. Tian and K. Benkrid, "American option pricing on reconfigurable hardware using least-squares monte carlo method," in *Proc. Int. Conf. on Field-Programmable Technology*, 2009, pp. 263 –270.

[4] J. Hull and A. White, "The use of the control variate technique in option pricing," *Journal of Financial and Quantitative Analysis*, vol. 23, no. 03, pp. 237–251, 1988.

[5] F. Black and M. Scholes, "The pricing of options and corporate liabilities," *Journal of Political Economy*, vol. 81, no. 3, pp. 637–654, 1973.

[6] D. B. Thomas and W. Luk, "Non-uniform random number generation through piecewise linear approximations," in *Proc. Int. Conf. on Field Programmable Logic and Applications*, 2006, pp. 1–6.