Collaborative processing of Least-Square Monte Carlo for American Options

Jinzhe Yang
Imperial College &
Aberdeen Asset Management
London, UK
jinzhe.yang12@imperial.ac.uk

Ce Guo
Imperial College
London, UK
ce.guo10@imperial.ac.uk

Wayne Luk
Imperial College
London, UK
w.luk@imperial.ac.uk

Terence Nahar
Aberdeen Assent Management
London, UK
terence.nahar@aberdeen-asset.com

***Abstract*—American options are popularly traded in the financial market, so pricing those options becomes crucial in practice. In reality, many popular pricing models do not have analytical solutions. Hence techniques such as Monte Carlo are often used in practice. This paper presents a CPU-FPGA collaborative accelerator using state-of-the-art Least-Square Monte Carlo method, for pricing American options. We provide a new sequence of generating the Monte Carlo paths, and a pre-calculation strategy for the regression process. Our design is customisable for different pricing models, discretisation schemes, and regression functions. The Heston model is used as a case study for evaluating our strategy. Experimental results show that an FPGA-based solution could provide 22 to 64.5 times faster than a single-core CPU implementation.**

I. Introduction

Monte Carlo methods are numerical computational methods which can simulate problems that are impossible or impractical to have an analytical solution. Although Monte Carlo methods make it possible to simulate options pricing process in the sophisticated market [1], overnight calculations cannot reach the run time requirement. There is a trend in the financial industry of using High Performance Computing technologies to accelerate complex simulation models for quick and accurate result. By the influence of these modern technologies, practitioners do not only refer to the simulation result to make decisions, but also in some circumstances, let the machine do the decisions itself.

The Heston model [2] is a commonly accepted stochastic volatility [3] model, especially for equity derivatives: unlike the Black-Scholes model [4], the volatility of the asset in Heston model follows a random process, instead of constant or deterministic. The model intuitively extends the Black-Scholes model as a special case. For derivatives whose underlying is the spot itself, it directly models stock price at time t for pricing. Heston's setting takes into account non-lognormal distribution of the asset returns, leverage effect, important mean-reverting property of volatility and it remains analytically tractable.

Much research based on FPGA with Monte Carlo methods to simulate option pricing process has been done in the past few years. However, to the best of our knowledge, little research has been done on Least-Square Monte Carlo (LSMC). It is not a new method, but it is a relatively novel approach for the finance industry, and it has a significant characteristic in simplifying the simulation progress [5]. It needs to traverse all the Monte Carlo paths, and update the regression coefficient matrix under some conditions. So the problem would be transformed into solving a linear equation. After the linear equation is solved, the approximation function coefficients would be acquired. The acquired approximation function would be used for updating information based on those paths that fulfils the previous conditions. This progress is activated iteratively until all the steps are operated.

It is easy to conclude that a pipelined design of LSMC has the following restrictions: traversing all the Monte Carlo paths for three times, and solving the linear equation. Our pipelined design has the following characteristics: by adjusting the calculation sequence and pre-calculate the inverse of the coefficient matrix, we only need to traverse twice, and also the pre-calculate process provides flexibility to the regression functions. In consideration of CPU-FPGA bandwidth limitation, we store all the information on the on-board memory (DRAM), and also in reason that if we deploy traditional generation sequence on our FPGA design, we may suffer high offset problem. Therefore, we reorganise the sequence so both generation process and regression process reads from the DRAM in a serial order.

This paper presents the design and implementation of a CPU-FPGA collaborative platform to simulate the result of using Least-Square Monte Carlo method to price American options via Heston model. Our major contributions are listed as follows:

- A CPU-FPGA collaboration strategy, which could make full use of both calculation advantage and capacity of both CPU and FPGA.
- A pipelined design of LSMC for pricing American options, which provides availability for different models and discretisation schemes.
- A generalised solution that could support various accurate but complex regression function.

We have tested for different sets of data size parameters which are all commonly used in practice, and experimental results show that we could achieve 22 and up to 64.5 times faster than a single-core CPU implementation.

The rest of the paper is organised as follows. Section II

978-1-4799-6245-7/14/$31.00 ©2014 IEEE

briefly introduces the American options, Least-Square Monte Carlo method and Heston model. Detailed introduction to our system design, as well as the pipeline-friendly algorithmic development for the problem, would be presented in section III and section IV. Experimental results and performance estimation would be provided in section V.

II. Background

This section provides a short introduction about why the problem is important to the finance industry.

First consider the American Options. Options are contracts between two parties for a future transaction on an asset at the strike price, the reference price. The buyer of the option has the right, but not the obligation, to exercise the transaction. The option valuation generally depends on a number of different variables in addition to the value of the underlying asset.

American option has an expiration date, however, it can be exercised at any time before the expiration date. It is straightforward that an early exercise decision is made only when people could predict with high confidence that it could have more benefit than exercise at expiration, and possibly has the highest benefit compared with all the potential exercise dates before expiration.

American options are widely traded in the market, and it has no analytical solution. Hence, valuation and optimal American-style derivatives, or any derivatives with early excise features, is one of the most important practical problems in options pricing. This feature mathematically means decisions should be made for all time steps. More precisely, for each time point that could make early exercise, we need to choose between immediate exercise or hold the option for better opportunity in the future. Simulating exercise price is relatively easy, however, there is no analytical form for calculating the continuation.

Second, consider the Least-Square Monte Carlo method. In real market requirement, we wish to calculate an accurate and reliable result over some future time horizon. The full nested simulation of this process (see Figure 1) is generally considered with two steps: we first generate the economic scenario, by projecting a number of realisations of the key economic variables, or risk-drivers, in which the future liability is likely to depend. Based on the economic scenario result, we can calculate the liability at the projection date. The former step is real world scenarios (outer scenarios), and the latter step is market consistent scenarios (inner scenarios).

It is intuitive that if we could cut down some branches of outer scenarios or inner scenarios, without losing accuracy, the computation would be reduced immediately. We now use only one inner scenario for valuing our liabilities, rather than running many of the market-consistent scenarios. The liability valuation has now been calculated very inaccurate, nevertheless, by applying regression, we could correct the inaccuracy.

For the regression, we use as explanatory variables, the values of the risk drivers at $t = 1$, and as a response variable, the liability value. The corresponding regression curve gives an approximation of the liability value as a function of the input and the different risk drivers. The function we will select to do our regression is crucial for the quality of our estimations. There are many different ways to do the regression, we generally assume the approximation function of y and given set of data x is $y = f(x)$, so in order to have the best fitting curve, $f(x)$ has the least square error. More details about the Least-Squares regression would be found in Section IV.

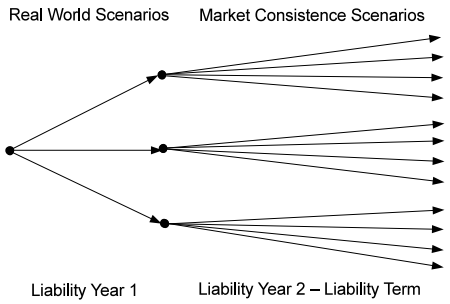

Fig. 1. Full nested valuation

Finally, consider the Heston Model. This paper selects the Heston model in a case study for various pricing models. Despite the tremendous success of the Black-Scholes model for options pricing, people found popularly-existed deficiencies in real market data, and perhaps the most important is the assumption for the underlying asset, the volatility of the return is constant. In reality, both strike price and maturity of traded options are generally considered to vary of implied volatility. A key concept to cope with the problem is that of stochastic volatility, and one of the most popularly used is the Heston model. Below is the definition of Heston model.

$$\Delta S_t = \mu\, S_t\, \Delta t + \sqrt{v_t}\, S_t\, \Delta W_t^S \tag{1}$$

$$\Delta v_t = \kappa\,(\theta - v_t)\,\Delta t + \xi \sqrt{v_t}\, \Delta W_t^v \tag{2}$$

where S_t is the price of the asset at time t, μ is the rate of return, θ is the long variance, κ is the rate at which v_t reverts to θ, and ξ is the volatility-of-volatility that determines the variance of volatility. The instantaneous variance v_t, is a time-independent Cox-Ingersoll-Ross Markov Process (CIR Process) [6]. ΔW_t^s and ΔW_t^v are two Wiener Processes, the correlation between ΔW_t^S and ΔW_t^v is ρ.

To calculate the stock price, the main difficulty is to solve the stochastic differential equation (SDE), so that the variables could be expressed as a function of finite set of state variables with known joint distribution. The calculation problem could then be reduced to sampling from the distribution. The stock price path follows a Markov Process, the calculation at time $t + 1$ solely based on information from time t:

$$v_{t+\Delta t} = f_v(v_t) \tag{3}$$

$$S_{t+\Delta t} = f_S(S_t) \tag{4}$$

where f_v and f_S are different discretisation methods for variance v and stock price S.

III. System Design

The process of pricing American options using LSMC involves a sequential memory read sequence of generating stock price paths and a reverse order of regression. Regarding the memory read/write sequence, we apply the strategy of two kernels in one chip, controlled by the same host code, with one kernel represents the generation, while the other one, regression.

A. Overall Design

We referred to previous research on FPGA [7] and found the restrictions which prevent those works for practical use. The main restrictions and challenges are:

1) Few works are stochastic volatility model, and the stock price paths are generated independently. More precisely, in each path, the stock price of each step only follows a normal distribution from S_0, and there is no relationship between the steps inside each path [8].
2) Traditionally, the stock price and variance are generated path-by-path, but are regressed step-by-step, which would bring high offset for memory allocation.
3) The regression function that previous work used is $y = a + bx + cx^2$, which is too simple and inaccurate. In practice, the regression function is much more complex. In addition, their FPGA solution only provides Cholesky-decomposition for solving 3×3 coefficient matrix of linear equation, which cannot be used for larger matrix (complex regression functions). In reality, the approximation function for regression could be binary, and the regression would be a 3-dimensional approximation [9].

To address challenge 1, we apply discretisation schemes to our hardware design. It is a pipeline-friendly algorithm because for each iteration, the calculation is only based on the result of the previous iteration, with several random numbers. In practice, lots of discretisation schemes are popularly used, and there is no generally accepted scheme that fits all requirements (e.g. jump diffusion, accuracy, computation speed). For a general purpose, our system design has no restriction to discretisation schemes, but we only provide the IMM-IJK scheme as a case study.

To address challenge 2, we provide a new generation sequence. Take the stock price at time t of path n ($S_{n,t}$) as an example. If we use traditional generating sequence on the FPGA, so $S_{n,t}$ and $S_{n,t+\Delta t}$ would be stored on the DRAM as neighbours, so obviously, during the generation, the FPGA reads from the DRAM in a serial sequence without offset. However, when doing the regression, we need stock price information of time t from different paths, the offset is N, as N denotes the total number of paths. To avoid the offset N, we reorganise the sequence so that during the generating process, the DRAM is read and written in a linear sequence, and during the regression, the sequence is revised but still it is a linear sequence.

To address challenge 3, the reason why some previous works are using simple regression function is that FPGA could not solve linear equation easily. The solution could be simplified as: we update the coefficient matrix of the linear equation as soon the new step of Monte Carlo path is updated, we use the CPU to calculate the inverse of the matrix instead of directly solving the linear equation on FPGA. Detailed solution for challenge 2 & 3 would be provided in section IV.

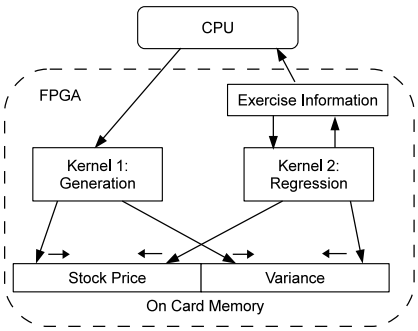

Fig. 2. System Design

Our system design could be simplified as figure 2, where the two kernels are represented as kernel 1 (generation) and kernel 2 (regression). The reason why the kernel is partitioned is: the read/write sequence on the DRAM is opposite for the two progresses, and the calculation inside each process is significantly different from the other. As mentioned before, we merge the three traverse process into two, so each kernel is one of the two traverses. The stock price and variance are generated by kernel 1, write to the DRAM in one dimensional linear sequence, and could be read from the DRAM in a inverse sequence by kernel 2. Kernel 1 has a data transfer from the CPU and to the DRAM, while kernel 2 has data transfer to and from the DRAM, and eventually transfer to the CPU. The block exercise information in the figure is also a DRAM block, the exercise prices and dates are stored in linear sequence. Kernel 2 reads the exercise information for each time step and writes back after it is updated. The exercise information would be read by the CPU after all the iterations are activated.

B. The Valuation Framework

Assume a finite time horizon $[0, T]$, where 0 is the start date of the contract, T is the expire date. We are focusing on valuing American-style derivative securities with random cash flows which may occur during $[0, T]$. [10] implies the value of American option equals to the maximised value of the discounted cash flows from the option, where the maximum is taken over all stopping times. Introduce the notation $C_{s,t}$ to denote the cash flow paths generated by the option, and for all s, $t < s \leq T$, the conditions of the option not being exercised at or prior to time t, the option holder follows the optimal stopping strategy. The LSMC method is objective to provide a pathwise approximation that maximises the value of the American option.

The process of generating stock price paths is illustrated as Algorithm 1. The algorithm provides a general method

for pricing models without jump diffusion [11]. The change of pricing model and/or discretisation scheme only influences how $v_{t+\Delta t}$ and $S_{t+\Delta t}$ is expressed. For a detailed explanation, different pricing model has the different calculation for ΔS_t and Δv_t, and the discretisation scheme defines how the variables could be expressed as a function of finite set of state variables with known joint distribution. When applying different discretisation scheme to the same model, it changes not only how the function of finite set of state variables is expressed, but also the accuracy, bias, convergence, e.t.c..

Algorithm 1 Generate Stock Price Path

for i = 1 **to** paths **do**
  for j = 1 **to** steps **do**
    Generate $\Delta W^v_{t+\Delta t} \sim N(0,1)$, $\Delta W^S_{t+\Delta t} \sim N(0,1)$
    Correlate $\Delta W^v_{t+\Delta t}$ and $\Delta W^S_{t+\Delta t}$ by:
    $\Delta W^v_{t+\Delta t} = \Delta W^v_{t+\Delta t}\sqrt{\Delta t}$
    $\Delta W^S_{t+\Delta t} = \rho \Delta W^v_{t+\Delta t} + \sqrt{1-\rho^2}\sqrt{\Delta t}\Delta W^S_{t+\Delta t}$
    $v_{t+\Delta t} = f_v(v_t, \Delta W^v_{t+\Delta t})$
    $S_{t+\Delta t} = f_S(S_t, v_{t+\Delta t}, \Delta W^v_{t+\Delta t}, \Delta W^S_{t+\Delta t})$

As for the model with jump diffusion, the only change for algorithm 1 is to add the functions of simulating the jump diffusion, before calculating $v_{t+\Delta t}$ and $S_{t+\Delta t}$. For some models, an additional random number might be needed. For example, for a double-jump model, a uniformed random number between $[0, 1]$ is required for sampling the number of jumps. The jump diffusion model is also applicable in our strategy.

C. Regression

Continued from the previous valuation process, at the maturity of the option, as introduced before, the option holder has the right, but not obligation to exercise at the strike price. So the investor exercises the option if it is in the money, or allows it to expire if it is out of the money. However, before the expiration time, the option holder must choose whether to exercise immediately or to continue the life of the option and revisit the exercise decision. The investor exercises as soon as the immediate exercise value is greater than or equal to the value of continuation, because the option is maximised pathwise.

At time t, the cash flows from continuation, of course, are not known at this time step. No-arbitrage valuation theory, however, implies that the value of the option, assuming that it cannot be exercised until after t, is given by taking the expectation of the remaining discounted cash flows $C_{s,t}$ with respect to the risk neutral pricing measure.

The LSMC method for pricing American options uses least squares to approximate the conditional expectation functions at each time step in a backward sequence, because $C_{s,t}$ is generated recursively [12][13]. $C_{s,t}$ can be differ from $C_{s,t+\Delta t}$ because it might stop at time $t+1$, thereby changing all subsequent cash flows.

In the estimation, at time $t-1$, we regress only from paths when it is in-the-money at time t, because the exercise decision is only relevant when the option is in-the-money. Once the conditional expectation function at time $t-1$ is estimated, we can determine whether early exercise at time $t-1$ is optimal for an in-the-money path ω by comparing the immediate exercise value with the continuation, and repeating for each in-the-money path. As soon as the exercise decision is identified, the cash flow path can then be approximated. This recursive process rolls back from the expire date, until the exercise decisions at each exercise time along each path have been determined.

The American option is then valued by starting at time zero, moving forward along each path until the first stopping time occurs, discounting the resulting cash flow from the exercise date back to time zero, and eventually taking the average over all paths. Algorithm 2 is a generalised regression process.

Algorithm 2 Regression

for i = steps **to** 1 **do**
  for j = 0 **to** paths **do**
    Update the coefficient matrix of step t
  Calculate the approximation function
  for i = 0 **to** paths **do**
    Calculate continuation of step i, path j
    if $K - S_{i,j} < continuation$ **then**
      Update exercise information

IV. Kernel Design

This section will introduce the kernel design of the LSMC simulation of Heston model, as well as the introduction of our algorithmic improvements in a detailed way. Our pipelined design contains two main parts in FPGA: scenario generator, Least-Square regression.

A. Scenario Generation

There are two steps in generating stock price path: generate two random number series, and use the random number pairs to calculate corresponding stock price. As mentioned in section II, ΔW^S_t and ΔW^v_t has the correlation ρ. So we first generate two Gaussian distributed random number series (x, y) without correlation, and the ΔW^S_t and ΔW^v_t would be calculated as follows:

$$\Delta W^v_t = x\sqrt{\Delta t} \tag{5}$$

$$\Delta W^S_t = (\rho x + y\sqrt{1-\rho^2})\sqrt{\Delta t} \tag{6}$$

Although the Uniformed Euler-Maruyama discretisation method is a classic way that provides an accurate yield in many cases, however, when simulating a long-term stock price path or large time intervals, the convergence would be erratic and uncontrollable in some cases. We referred to an improved method, which at least for Heston model, yields the best results as measured by a strong convergence measure. [14] applied the implicit Milstein method (IMM) to the variable, combined

with their bespoke IJK scheme for the logarithm of the stock price. The IMM method discretises the variance as follows:

$$\begin{aligned} v_{t+\Delta t} =& v_t + \kappa\Delta t(\theta - v_t)) + \xi(\sqrt{v_t}\,\Delta W_t^v) \\ &+ \frac{1}{4}\xi^2(\Delta W_t^{v\,2} - \Delta t) \end{aligned} \tag{7}$$

The IMM method preserves the positivity for mean-reverting square root process. The scheme for the logarithm of stock price is their IJK scheme [15]:

$$\begin{aligned} lnS_{t+\Delta t} =& lnS_t + \mu\,\Delta t - \frac{1}{4}\Delta t(v_t + v_{t+\Delta t}) + \rho\sqrt{v_t}\,\Delta W_t^v \\ &+ \frac{1}{2}(\sqrt{v_t} + \sqrt{v_{t+\Delta t}})(\Delta W_t^S - \rho\,\Delta W_t^v) \\ &- \frac{1}{4}\xi\rho(\Delta W_t^{v2} - \Delta t) \end{aligned} \tag{8}$$

It is specifically tailored to stochastic volatility models, where typically ρ is highly negative. The combined scheme is called IJK-IMM scheme, and it is a quasi-second order.

It could be easily concluded that both v and S has a data dependency with the corresponding v and S of previous time step. Thus, researchers always generate it in a horizontal way, as shown in figure 3.

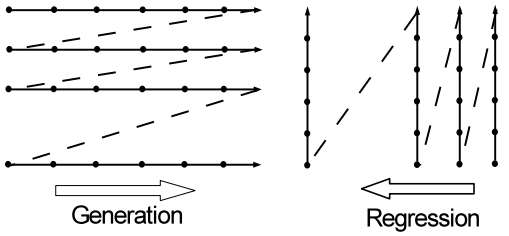

Fig. 3. Traditional Calculation Sequence

In figure 3, each dot means an early exercise time. The dots in the same row indicate all the early exercise times in the same Monte Carlo path, the columns indicate different Monte Carlo paths, and the arrows describes the calculation sequence. The figure implies that traditionally, the paths are generated path-by-path, while the regression is step-by-step.

Due to the bottleneck of data transfer between the CPU and the FPGA, most of our data would be stored in the DRAM. However, it brings high offset of memory access, because it generates path-by-path, and calculated from backward step-by-step. The offset is the number of paths, which could always be relatively high. By analysing the pricing progress, we found the regression part has a much stronger requirement of stock price data continuity, since it needs to operate all the stock price step-by-step. In addition, if we could find an alternative solution to generate price path step-by-step, the pricing process would become pipeline-friendly.

We present an alternative sequence of generation. It is generated step-by-step by using a customised memory address generator, and the progress is shown in figure 4. Therefore, the stock price and variance would be loaded sequentially during the backward regression. We also make benefit from our new sequence of generation, we could accumulate for the coefficient matrix of regression while generating the path.

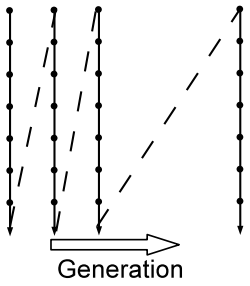

Fig. 4. Step-by-Step Generation Sequence

From the IMM-IJK method, $v_{t+\Delta t}$ is calculated by v_t, ΔW_t^v, $\Delta W_{t+\Delta t}^v$, and $S_{t+\Delta t}$ has additional data dependency on $v_{t+\Delta t}$, ΔW_t^S, $\Delta W_{t+\Delta t}^S$, which shows at each time step, the variance and stock price only directly need the result of the previous step. Detailed illustration for our new generation sequence, as well as updating the coefficient matrix, would be introduced together with the regression part.

For a simple illustration, and considering the space limit, we use simple Eular-Maruyama discretisation scheme and one accumulator for the graphic explanation of our kernel design.

$$v_{t+\Delta t} = v_t - \kappa(v_t - \theta)\Delta t + \xi\sqrt{v_t}\Delta W_t^v \tag{9}$$

$$S_{t+\Delta t} = S_t(1 + \mu\Delta t + \sqrt{v_{t+\Delta t}}\Delta W_t^S) \tag{10}$$

The real design for the valuation process and multiple accumulators would be similar but much more complex. Here, and also for the figure 8, the approximation function is only a two dimensional approximation function $f_t(S) = a + bS_t + cS_t^2$. The design for kernel 1 could be separated into two parts, one for the variance, and the other, the stock price. Figure 5 shows the design for generating variance, where $v_{t+\Delta t}$ is the output for kernel 1, and it is also the input for generating stock price. Figure 6 shows the design for calculating the stock price, as well as how to use the accumulator to calculate the coefficients of the matrix. We only show a single accumulator for $S_{t+\Delta t}^2$, however, we have multiple accumulators in our real design. The multiplexer MUX works as follows: if it is in-the-money ($S_{t+\Delta t} < K$), the MUX outputs the stock price, otherwise, it outputs zero. The reason why there is an accumulator would be illustrated later.

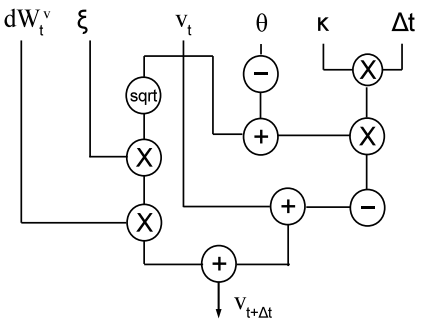

Fig. 5. Generating Variance

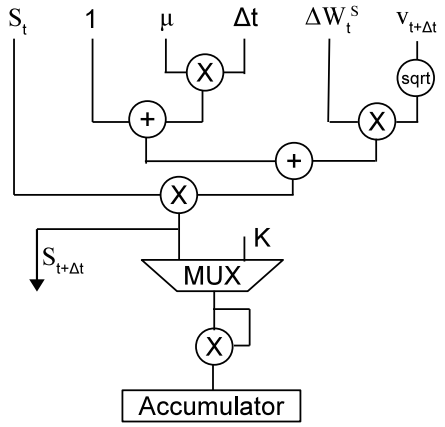

Fig. 6. Generating Stock Price

B. Least-Square Regression

After the paths are completely generated and stored, the backward calculation for continuation is now activated. For each time step t, we calculate both exercising cash-flow and continuation cash-flow.

For the Least-Square Regression, we have three steps, generating the regression coefficient matrix, solve the linear equation, and update the exercise price and exercise date. The generating progress follows the idea of minimising the least square error of the fitting curve $f(x)$. Here, we use a polynomial regression function as an example. Assume the approximation function is:

$$y = b_0 + b_1 x + b_2 x^2 + \cdots + b_m x^m \tag{11}$$

where $b_0, b_1, \cdots, b_m$ are unknown coefficients. It's obvious that these coefficients must yield zero first derivatives.

$$\begin{aligned}
\sum_{i=1}^{n} y_i &= b_0 \sum_{i=1}^{n} 1 + b_1 \sum_{i=1}^{n} x_i + \cdots + b_m \sum_{i=1}^{n} x_i^m \\
\sum_{i=1}^{n} x_i y_i &= b_0 \sum_{i=1}^{n} x_i + b_1 \sum_{i=1}^{n} x_i^2 + \cdots + b_m \sum_{i=1}^{n} x_i^{m+1} \\
\sum_{i=1}^{n} x_i^2 y_i &= b_0 \sum_{i=1}^{n} x_i^2 + b_1 \sum_{i=1}^{n} x_i^3 + \cdots + b_m \sum_{i=1}^{n} x_i^{m+2} \\
&\vdots \\
\sum_{i=1}^{n} x_i^m y_i &= b_0 \sum_{i=1}^{n} x_i^m + b_1 \sum_{i=1}^{n} x_i^{m+1} + \cdots + b_m \sum_{i=1}^{n} x_i^{2m}
\end{aligned} \tag{12}$$

Therefore, we can get a polynomial approximation formula:

$$Xb = Y \tag{13}$$

where

$$b = [\ b_0 \ \ b_1 \ \cdots \ b_k\ ]^T \tag{14}$$

$$X = \begin{pmatrix}
n & \sum_{i=1}^{n} x_i & \cdots & \sum_{i=1}^{n} x_i^k \\
\sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 & \cdots & \sum_{i=1}^{n} x_i^{k+1} \\
\cdots & \cdots & \cdots & \cdots \\
\sum_{i=1}^{n} x_i^k & \sum_{i=1}^{n} x_i^{k+1} & \cdots & \sum_{i=1}^{n} x_i^{2k}
\end{pmatrix} \tag{15}$$

$$Y = [\ \sum_{i=1}^{n} y_i \ \ \sum_{i=1}^{n} x_i y_i \ \cdots \ \sum_{i=1}^{n} x_i^k y_i\ ]^T \tag{16}$$

In the polynomial approximation formula, n is the given set of data, and in our case, the number of paths that is in-the-money at time t. b is the coefficient vector we are going to calculate, x_i could be the stock price at time t, and y_i could be the discounted corresponding cash-flows.

The regression function we actually use can be found in [16]. It is a three dimensional approximation. More precisely, the regression function is $f(S, v)$ instead of $f(S)$. The variables in function f are independent variables: S is the stock price, and v is the corresponding variance.

The most difficult problem for the regression step, and also for this paper, is efficiently solving the linear equation with symmetric coefficient matrix. Obviously there are myriads of methods for solving linear equations, nevertheless, few algorithms have good performance on FPGA. There are two ways of solving a problem: directly and indirectly. Direct method solves the linear equation problem by using an efficient and data-flow-friendly algorithm (e.g. iterative methods for solving linear equation), and the indirect method is to find an alternative solution. We have both solutions, however, in consideration of the flexibility of regression function, we only present the indirect method in this paper.

The key issue is most of the algorithms for solving the linear equation need to access memory randomly, or in other words, data dependency is with large offset. Some publications had been focused solving Cholesky-decomposition in the FPGA, which is complicated for data flow acceleration. Instead, we present a collaborative CPU-FPGA strategy to solve the coefficient equation.

Take the following scenario as an example. Obviously, in each time step, we need to generate and solve $Xb = Y$, note that the stock prices (and variance, if it is a three dimensional approximation) only contribute to the coefficients in matrix X if and only if the stock price is in-the-money. Consider that Y has the data dependency on continuation which could only be acquired during the regression progress, and the stock price contributes to coefficients in Y is under the same condition with X, so we calculate X and Y separately. As for the matrix X of time step t, it is generated simultaneously with the stock price in the same time step. Figure 7 provides a Gantt chart to illustrate the time usage. For the CPU, the shorter line indicates

the calculation of matrix inverse, while the longer indicates generating the random numbers for the next step.

The proposed CPU-FPGA collaboration strategy is as follows: when the stock price in time step $t-1$ ($t>1$) is fully generated, the FPGA output the matrix X_{t-1} to the CPU and starts calculating the stock price of time t. So at time t, X_{t-1}^{-1} is being calculated, and also the random numbers for time $t+1$ would be generated, both on the CPU, while the FPGA holds the calculation of generating the variance, stock price, and matrix X of time t, concurrently.

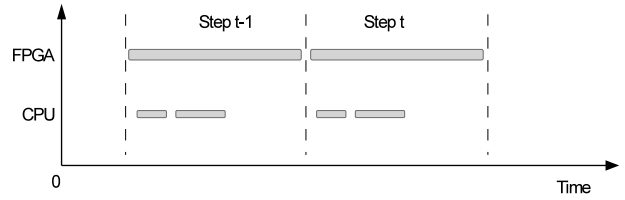

Fig. 7. Time Usage of the Collaborative Strategy

After the paths and the corresponding matrix X have been fully generated, the regression step is activated. Once the kernel moves to a new time step $t+1$ of regression, the X_t^{-1} would be transferred from the CPU to the FPGA. By continuously accumulating the contribution to Y_t, under the same condition when generating X_t, we eventually acquire Y_t, as well as the corresponding X_t^{-1}. So the linear equation $X_t b_t = Y_t$ for time t could be transformed and solved in a simple way, by calculating $b_t = X_t^{-1} Y_t$. In addition, at the same time, exercise information is updated as we have already received all the required information for current step t during regression in step $t+1$.

As mentioned before, the decision of exercise or hold the option is made now. Assume continuation denotes the value of not immediate exercise of path n at time t, continuation $= f(S_{n,t}, v_{n,t})$. If and only if we can acquire more benefit by exercising now (exercise $>$ continuation), we update the exercise price matrix. The regression progress is proceeding recursively, until the first time step is calculated.

Figure 8 shows our hardware design of the regression part.

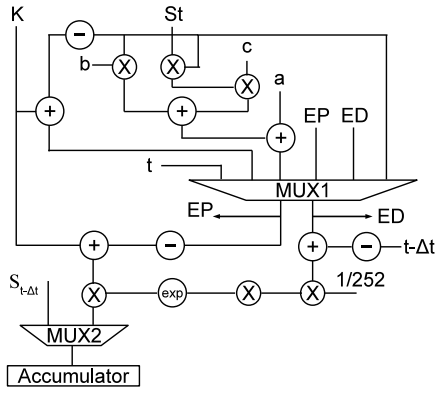

Fig. 8. Kernel Design for Regression

Because the vector Y of time step t requires exercise information of time step $t+1$, so it could only be generated during the regression process. Assume in time step t, we merge the updating exercise information of t and updating coefficients of vector Y of time $t-1$ together. EP and ED represent for exercise price and exercise date, respectively. The six inputs of multiplexer MUX1 are: time t, cash flow at time t, continuation, stock price at time t, and exercise price and exercise date from previous results. If the cash flow is greater than the continuation, the MUX1 outputs the EP as S_t, and ED as t, otherwise, EP and ED remain unchanged. The inputs of MUX2 are: stock price of the previous step, and the cash flow based on updated exercise information, after discounted to previous time step. If it is in-the-money for the previous step, the MUX2 outputs $S_{t-\Delta t}$ to the accumulator, otherwise, it outputs zero.

V. Hardware Implementation and Evaluation

A. Experimental Evaluation

The mathematical derivation of our hardware design targets a MAX3 acceleration system. The hardware is described in a Java based MaxJ language and compiled with Maxeler MaxCompiler. The acceleration system is equipped with a Xilinx Vertex-6 V-SX4757 FPGA card. It communicates with the host computer via a PCI-Express interface. In our implementation, we set the clock frequency to 100MHz.

We also build a CPU-based system by implementing a traditional sequence of generating the Monte Carlo paths, on the CPU platform in a server with a Xeon X5650 CPU, with 6 cores 12 threads running at 2.67GHz. The experimental code for both CPU version and the host code for our FPGA design are written in C programming language, and complied with Intel C compiler with the highest compiling optimisation.

Experimental results are shown in table I. One condition is that if the number of steps T is also extremely big, which means early exercise interval we estimate is short, we might face bottleneck on the CPU version because the CPU cache is not large enough. This would be expressed as the experimental result of scenario 1 and scenario 4. Also, in consideration that in practice, 10M paths would be enough for most calculation requirements, so we haven't further tested the case when the number of paths N is extremely large. We have 5 test cases, for the first 3 tests, we set $T = 252$ and keep it unchanged, 252 means the total trading days per year, and we have 3 cases with $N =$ 1M, 5M, 10M, respectively. For the remaining 2 test cases, we have $N =$ 1M and keep it unchanged, while the T is 2520 and 5040, which means that for each trading day, the early exercise opportunity we estimate is 10 and 20.

Steps	Paths	CPU	CPU+FPGA	speedup
252	1,000,000	83.26s	3.78s	22.0x
252	5,000,000	423.78s	13.86s	30.6x
252	10,000,000	834.36s	26.46s	31.53x
2520	1,000,000	2489.63s	39.23s	63.5x
5040	1,000,000	4986.91	77.35s	64.5x

TABLE I
FPGA PERFORMANCE

Interestingly, we found that the speedup changed significantly between different sets of data size. We explain scenario 3 and 4 to illustrate the reason. Scenario 3 and 4 are both

popularly used data size parameter sets in practice, one attempt is increasing the number of Monte Carlo paths, and the other, decreasing the discretisation time intervals. The total amount of data is the same, however, the CPU execution time and the speedup varies. The difference between the two scenarios is that scenario 3 is doing regression for 252 times, but each time updating 10M paths, while scenario 4 is having regression for 2520 times but each time only for 1M paths. The information is stored in a two dimensional array, however, the cache is not big enough to handle data in such amount. So scenario 4 might be suffering from a much more serious cache missing problem than scenario 3, because they are both stored in the same "structure", but read/write offset is quite different. If we revert the column and the row, scenario 3 would face a similar cache problem as scenario 4. As for our pipelined design, the time consuming change is brought by the times that kernels are revoked and the number of paths. In addition, in our FPGA design, all the information is stored in the DRAM in a one dimensional linear sequence, so all of the offsets are 1.

B. Performance Estimation

Based on our experimental result, we provide an approximation of performance estimation. Assume we have N Monte Carlo paths, T steps, and the clock frequency is F. Let R_1 and R_2 be the revoke time for kernel 1 and kernel 2, respectively, and L_1 and L_2 be the latency, which equals to the cycle number between the first data is input to and output from FPGA of kernel 1 and kernel 2, respectively. So the total execution time of FPGA, t_{total}, could be approximated by:

$$\begin{aligned} t_{total} &\approx \sum_{i=1}^{T}(\frac{(N+L1)}{F} + \frac{(N+L2)}{F} + R_{1,i} + R_{2,i}) \\ &\approx \frac{2NT}{F} + T(\hat{R_1} + \hat{R_2}) \end{aligned} \tag{17}$$

where $\hat{R_1}$ and $\hat{R_2}$ is the average revoke time for kernel 1 and kernel 2, respectively. L_1 and L_2 could be ignored if N is large enough. At present, our experimental result are only based on a single FPGA. By comparing cases with different N, we estimate that $\hat{R_1} \approx \hat{R_2} = 0.005$s, so $T(\hat{R_1} + \hat{R_2}) \approx 0.01$s, and equation 17 could be changed to:

$$t_{total} \approx \frac{2NT}{F} + 0.01T \tag{18}$$

As for scalability, although we have not tested our scheme on multiple FPGAs, but we could provide an analytical illustration for why the speedup would grow almost linearly when we deploy the scheme on more FPGAs.

VI. Conclusion

This paper has presented a pipeline-friendly solution for pricing American options using Least Square Monte Carlo method. Instead of providing an accelerator for a special case, we provide a general strategy that could support different models and discretisation schemes, especially different regression functions in two dimensional, three dimensional or even high dimensional approximations. By changing the accumulator, our design could fit different regression functions properly. Our implementation result shows 22 to 64.5 times speedup compared with a single-core CPU implementation.

This work shows the potential of reconfigurable computing for pricing American options in practice, and the flexibility for different requirements. Future work includes raising the clock frequency to around 200 MHz, so that the speedup of a single FPGA design can be doubled to reach 130 times; this can be achieved by adopting optimisations such as mixed precision representation [17] and idle function elimination by run-time reconfiguration [18].

Acknowledgment

The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement n 289032. This work is also supported in part by the UK EPSRC, by the Maxeler University Programme, by the HiPEAC NoE, by Altera, and by Xilinx.

References

[1] P. Glasserman, *Monte Carlo Methods in Financial Engineering*. Springer Science, 2004.

[2] S. L. Heston, "A closed-form solution for options with stochastic volatility with applications to bond and currency options," *The Review of Financial Studies*, vol. 6, no. 2, pp. 327–343, 1993.

[3] P. Jackel, "Stochastic volatility models: past, present and future," *Incorporating the Quantitative Finance Review*, 2005.

[4] F. Black, "The pricing of options and coporate liabilities," *Journal of Political Economy*, pp. 637–654, 1973.

[5] F. Longstaff and E. Schwartz, "Valuing American options by simulation: A simple least-squares approach," *Review of Financial Studies*, vol. 14, no. 1, pp. 133–147, 2001.

[6] J. Cox, J. Ingersoll, and S. Ross, "An intertemporal general equilibrium model of asset prices," *Econometrica*, vol. 53, pp. 363–384, 1985.

[7] X. Tian and K. Benkrid, "Implementation of the Longstaff and Schwartz American option pricing model on FPGA, *Journal of Signal Processing Systems*, pp. 79–91, 2012.

[8] X. Tian, "American option pricing on reconfigurable hardware using least-squares Monte Carlo method," in *International Conference on Field-Programmable Technology*, 2009.

[9] D. Yang, G. Peterson, H. Li, and J. Sun, "An FPGA implementation for solving least square problem," in *Field Programmable Custom Computing Machines*, 2009.

[10] A. Bensoussan, "On the theory of option pricing," *Acta Applicandae Mathematicae*, pp. 139–158, 1984.

[11] R. C. Merton, "Option pricing when underlying stock returns are discontinuous," *Journal of Financial Economics*, pp. 125–144, 1976.

[12] H. White, "Asymptotic theory for econometricians," *Academic Press*, 1984.

[13] T. Amemiya, *Advanced Econometrics*. Basil Blackwell, 1985.

[14] C. Kahl, "Positive numerical integration of stochastic differential equations," Ph.D. dissertation, University of Wuppertal and ABNAMRO, 2004.

[15] C. Kahl and P. Jackel, "Fast strong approximation Monte Carlo schemes for stochastic volatility models," ABNAMRO and University of Wuppertal, Tech. Rep., 2005.

[16] M. J. Cathcart, "Monte Carlo simulation approaches to the valuation and risk management of unit-linked insurance products with guarantees," Ph.D. dissertation, Heriot-Watt University, 2012.

[17] G. Chow *et al*, "A mixed precision Monte Carlo methodology for reconfigurable accelerator systems," in *International Symposium on Field Programmable Gate Arrays*, 2012.

[18] X. Niu *et al*, "Automating elimination of idle functions by run-time reconfiguration," in *Field-Programmable Custom Computing Machines*, 2013.