OPTIMISING EXPLICIT FINITE DIFFERENCE OPTION PRICING FOR DYNAMIC CONSTANT RECONFIGURATION

Qiwei Jin, Tobias Becker, Wayne Luk

Department of Computing
Imperial College London
email: {qj04,tbecker,wl}@doc.ic.ac.uk

David Thomas

Department of Electrical and Electronic Engineering
Imperial College London
email: d.thomas1@imperial.ac.uk

ABSTRACT

This paper demonstrates a novel optimisation methodology to adjust stencil based numerical procedures from the algorithm level, so as to reduce not only the amount of hardware resource consumption per kernel but also the amount of computation required to achieve desired result accuracy, when mapping the algorithm to reconfigurable hardware using dynamic constant reconfiguration. As a result, less area is needed to support run-time reconfiguration, and less computational steps are required in the numerical procedure to obtain a result with given error tolerance. We analyse one thousand fixed point implementations on a Virtex-6 XC6VLX760 FPGA for randomly generated option pricing problems, which are representative of industrial computation. When comparing optimised implementations to the unoptimised ones, the reconfiguration area upper bound is reduced by 22%; the average number of computational steps is reduced by 23%; and the area-computation-time product is reduced by 40%; while the numerical errors of the results are kept below the error tolerant level used in industry.

1. INTRODUCTION

Ever since the financial crisis in 2008, the financial industry has been demanding higher computational power to understand its risk position in the market. The computationally complex models the industry is using can benefit from customised hardware accelerators for high computational throughput tailored to a particular requirement. Options are popular in the financial industry and pricing options usually involves solving partial differential equations (PDEs). In many cases a closed form solution cannot be found for a PDE and numerical methods such as the Explicit Finite Difference (EFD) method must be used [1]. Although the EFD method is widely used since it is relatively easy to apply, its computational complexity grows quadratically with increasing accuracy.

The EFD method is often used in financial institutions to evaluate large portfolios involving multiple asset classes, and even running on a large corporate computational grid, the process can take several hours. Reconfigurable hardware such as FPGAs can be used to accelerate this process effectively and energy efficiently [2]. Higher performance and energy efficiency can be achieved by developing reconfigurable versions of an application [3]. In addition, dynamically reconfigurable EFD implementations on FPGAs can have a factor of 4.7 performance improvement over a design using a static configuration [4].

This work extends our previous work [4] by making use of carefully chosen coefficients for constant multipliers, so as to reduce the amount of required hardware resources per kernel, and reduce the amount of computation required with given result accuracy requirement. We discuss the application of our work to the financial EFD method, though it is applicable to any stencil computation with constant coefficients. The main contributions of our paper are:

- A novel methodology based on normalising option parameters and matching statistical moments, which optimises the coefficient values in EFD solvers in order to reduce hardware resource consumption and number of computational steps in reconfigurable computation, while preserving result accuracy (Section 3).
- A workflow realising the methodology with two approaches: one minimises hardware resource utilisation, while the other minimises the amount of computation required in the algorithm (Section 4).
- An experimental evaluation of our methodology and workflow based on the amount of hardware resource consumed, the amount of computation required and error analysis in a financial option pricing application, showing that the area-time product can be reduced by up to 40% for given accuracy (Section 5).

2. THE FINITE DIFFERENCE MODEL

The finite difference model solves the Black Scholes PDE by discretising both time and the underlying asset price, and mapping both onto a two-dimensional grid. There are three main kinds of finite difference methods in common use: implicit, explicit and Crank-Nicolson. We consider the explicit mechanism as it is the most intrinsically parallelisable method amongst all three.

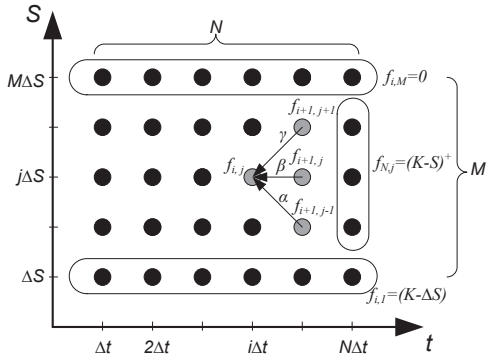

Fig. 1. A 5×6 finite difference grid, the grey elements show $f_{i,j}$ and the three values it depends on in time step $i+1$. Note that $(x)^+ \equiv \max(x, 0)$

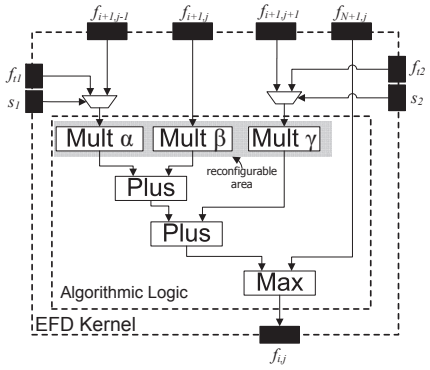

Fig. 2. Dynamic EFD kernel for American option pricing.

The Black Scholes PDE with one variable (asset) following geometric Brownian motion has the following form:

$$\frac{\partial f}{\partial t} + rS\frac{\partial f}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 f}{\partial S^2} = rf \tag{1}$$

where $f(S,t)$ denotes the price of the option, S denotes the value of the underlying asset, *t* denotes a particular time, r is the risk-free interest rate, σ is the volatility of the underlying asset. For convenience we define the option describer $\kappa \equiv (S, K, r, t, \sigma)$, where K is the strike price of the option.

Suppose the time to maturity for the option is T. We discretise T by dividing it into N equally spaced intervals: $\Delta t = T/N$. We then determine an asset prices $S_{\max}$ and $S_{\min}$ for the option and discretise the asset price space between $S_{\max}$ and $S_{\min}$ into M equally spaced intervals ΔS. $S_{\max}$ and $S_{\min}$ represents the maximum and minimum asset prices under consideration in the EFD grid. We call N the number of time steps and M the number of asset price steps, N and M together define a particular grid setting for the EFD model. In addition, we define the number of computational steps (problem size) to be $M \times N$.

An efficient approach to compute within a finite difference grid can be obtained by a change of variable technique [5]. By discretising over $Z = \ln S$ instead of S, Equations 2 can be obtained, where stencil coefficients α, β and γ are constants throughout one grid.

$$\begin{aligned}
\alpha &= \frac{1}{1+r\Delta t}\left(-\frac{\Delta t}{2\Delta Z}(r - \sigma^2/2) + \frac{\Delta t}{2\Delta Z^2}\sigma^2\right) \\
\beta &= \frac{1}{1+r\Delta t}\left(1 - \frac{\Delta t}{\Delta Z^2}\sigma^2\right) \\
\gamma &= \frac{1}{1+r\Delta t}\left(\frac{\Delta t}{2\Delta Z}(r - \sigma^2/2) + \frac{\Delta t}{2\Delta Z^2}\sigma^2\right)
\end{aligned} \tag{2}$$

The payoff function $f_{i,j}$ can have different forms, for example, Equation 3 and Equation 4 are the payoff functions for European and American options respectively, where $K - j\Delta S$ is the early exercise price of the American option.

$$f_{i,j}^{EU} = \alpha f_{i+1,j-1} + \beta f_{i+1,j} + \gamma f_{i+1,j+1} \tag{3}$$

$$f_{i,j}^{AM} = \max(K - j\Delta S, f_{i,j}^{EU}) \tag{4}$$

Figure 1 shows how the EFD method updates nodes in a 5×6 grid. As an initial condition, the values for nodes in the rightmost column is calculated by $\max(K - S_{N+1,j}, 0)$ and the algorithm runs from right to left.

The convergence of the EFD method is defined as the rate at which the error decreases with the number of computational steps [6]. Equations 5 illustrates the relationship between the number of time steps N required to achieve a given accuracy, the number of computational steps in the grid and two main sources of errors. The discretisation error is caused by transforming the model from a continuous mathematical space to a discretised computational space; and finite precision error is caused by using number representations of insufficient accuracy. For convenience we define an elastic factor $\lambda = \Delta Z/\Delta t$. If the number of time steps N is given, the convergence of the EFD method can only increase by decreasing λ (increasing the number of asset price steps M) and vise versa.

$$\begin{aligned}
&Num.\ Comp.\ Steps \propto N^2/\lambda \\
&Discretisation\ Error \propto N^z, z = f(\Delta t, \lambda) <= -1 \\
&Finite\ Precision\ Error \propto sqrt(N)
\end{aligned} \tag{5}$$

In previous work we have described a parallel hardware architecture which makes use of dynamic constant reconfiguration [4]. The work demonstrates the effectiveness of dynamic constant reconfiguration by studying the use of fixed point constant multipliers in a parallel option pricing architecture, as shown in Figure 2.

3. OPTIMISING DYNAMIC CONSTANT RECONFIGURATION

We propose that three key steps should be followed in optimising high performance designs in reconfigurable hardware from software implementations:

- use custom data formats to reduce area while preserving sufficient result accuracy;
- use constant specialisation and reconfiguration to further reduce area and energy consumption;
- use carefully selected constants to reduce the upper bound of reconfigurable area.

Each step yields higher performance than the previous step mainly due to higher level of parallelism achievable given a fixed amount of hardware resource. Recent work focuses on applying the first step and trying to reduce the floating point data-width aggressively, in order to trade the amount of computation for hardware resource consumption. For example, using a low precision data path, higher performance can be achieved by increasing sampling points in a numerical integration routine [7]. The second step has been addressed by [4] and [3].

To the best of our knowledge, the third step has not been published in public domain. This paper discusses following and going beyond the third step to optimise the EFD option pricing model. One of the main concepts to optimise the dynamic constant reconfiguration is to minimise the upper bound hardware resource consumption of the reconfigurable area (the gray area in Figure 2), so that higher parallelism can be achieved by placing more dynamic kernels in an FPGA.

The idea of our optimisation process is balancing the following two goals: (a) to reduce hardware resource consumption by carefully setting the EFD grid, so that the constant multipliers used in each dynamic kernel consume minimal amount of hardware resource; (b) to reduce the number of computational steps required in the EFD grid, so that the result meets the precision requirement without wasting unnecessary computation.

There are three key steps in the process:

1. normalising the option coefficients so that fixed point datapaths, which consume less hardware resources, are used instead of floating point datapaths;
2. identifying the nice constants in fixed point number representation which yield constant multipliers that consume less hardware resource than the biggest constant multiplier of the same number format; and adjust the EFD algorithm to make use of the nice constants;
3. ensuring the number of computational steps required in the optimised EFD scheme is not larger than the original scheme.

The first step makes use of fixed point number representation, by making sure that the finite precision error does not effect the quality of the result. Our previous work claims that fixed point numbers should not be used for option pricing, because the range of the inputs is not predictable, and unlike Monte Carlo simulations, finite precision error accumulates in the EFD scheme [2]. A normalisation procedure can be used to overcome this. Equation 3 and Equation 4 have type $\kappa \rightarrow \mathbb{R}$, where $\kappa \equiv (S, K, r, t, \sigma)$ describes an option. Although it is possible to define a range for all five inputs, the intermediate variables $f_{i+1,j-1}$, $f_{i+1,j}$ and $f_{i+1,j+1}$ can range from zero to strike price K for an American put option. Since $0 < K < \infty$, we need to reserve enough bits for the integer part of the fixed point representation to avoid integer overflow in K. The normalisation procedure is shown in Equations 6 to narrow down the numerical range of the intermediate variables. The option price is then calculated by $f = f' \times K$, where f' is the result from the normalised space.

$$K' = t' = 1,\ S' = \frac{S}{K},\ r' = r \times t,\ \sigma' = \sigma \times \sqrt{t} \quad (6)$$

As a result, the previously unbounded variable f is now bounded from zero to one, allowing the use of fixed point numbers in our design.

As the second step, we try to identify nicer constants which yields small fixed point constant multipliers (measured in number of consumed LUTs, compared to the largest constant multiplier in the same number format). To make the constant multipliers used in the EFD kernel consume less LUT resource, to one extreme, we would like the stencil coefficients α, β and γ to be in the form 2^E ($E \in \mathbb{Z}$) in binary. As a result, the constant multipliers can be implemented by shift operators in fixed point arithmetic, which consumes less LUTs than those implemented by adders.

Although we would like to make the constant multipliers consume as less LUTs as possible, the stencil coefficients must satisfy Equations 7, which are the first two requirements to guarantee the result convergence [5].

$$\alpha + \beta + \gamma = 1,\ \alpha > 0,\ \beta > 0,\ \gamma > 0. \quad (7)$$

In addition, all lattice based numerical procedures must match statistical moments of the underlying asset process. For stock option pricing, the Black Scholes PDE is constructed by creating a delta-hedged riskless portfolio, based on a log-normal process which describes the underlying asset price movement, as shown in Equation 8, where dW is a Wiener process.

$$d \ln S = dZ = (r - \frac{\sigma^2}{2})dt + \sigma dW. \quad (8)$$

Such process has the following mean and variance over a discretised time period Δt [1]:

$$\mu = (r - \frac{1}{2}\sigma^2)\Delta t, \;\; Var = \sigma^2 \Delta t \tag{9}$$

The EFD procedure must match mean μ and variance Var to ensure result convergence [1]. The speed of convergence is determined by how well the procedures matches higher moments such as skewness and kurtosis, the higher the convergence speed, the less number of computational steps is needed to obtain a result of desired accuracy. The trinomial stencil used in the EFD method has enough degrees of freedom to match one higher moment (skewness or kurtosis) [8], although as long as the first two moments match and Equations 7 are satisfied, the stencil coefficients α, β and γ can vary while the EFD result is guaranteed to converge to the true result.

Figure 3 illustrates the relationship between Δt, ΔZ and the stencil coefficients in an EFD grid. To ensure that the statistical moments described in the stencil match the theoretical moments described in Equations 9, the following equality must hold:

$$\mu = \mu', \;\; Var = Var', \tag{10}$$

$$\mu' = \alpha \times dZ - \gamma \times dZ, \tag{11}$$

$$Var' = \alpha \times (dZ - \mu')^2 + \beta \times (\mu')^2 + \gamma \times (dZ + \mu')^2. \tag{12}$$

Variables μ and Var are defined in Equations 9. Solving Equations 7 for the stencil coefficients α, β and γ gives us the coefficients in the form of $(\Delta t, \Delta Z, \kappa) \rightarrow \mathbb{R}$, similar to those in Equation 2 [1]. This relationship can be used to explore the stencil coefficients space and find smaller constant multipliers to be used in the dynamic EFD kernel. Figure 5 shows one example of the change in the values of the coefficients α, β and γ against $\lambda = \Delta Z / \Delta t$, while Δt stays constant. It can be seen that these coefficients are all continuous convex functions against λ which increase or decrease monotonically within range [0,1]. The relative error, defined as the absolute difference between the EFD result and the Black Scholes formula result using double precision arithmetic ($r.e. = |result_{EFD} - result_{BS}|$), is also shown in Figure 5 for each λ. It can be observed that the relative error increases with the asset price step ΔZ, with exceptions at the left boundary when one of the coefficient becomes very close to zero and conditions specified in Equations 7 are no longer valid.

As the third step, we discuss the number of computational steps required in the EFD algorithm. The EFD method requires the smallest number of computational steps to converge to the true result, when both the model skewness and kurtosis are close to their theoretical values, one common practice is to set the EFD grid equivalent to a trinomial tree [1].

[1] Any mathematical tool with a symbolic solver, such as Mathematica, can be used for this purpose

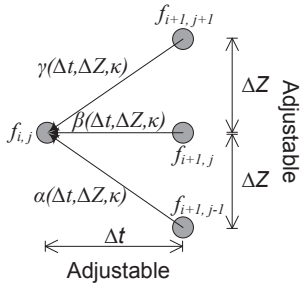

Fig. 3. Relationship between the stencil parameters and Δt and ΔZ. Note that user can vary Δt and ΔZ to obtain different stencil parameters.

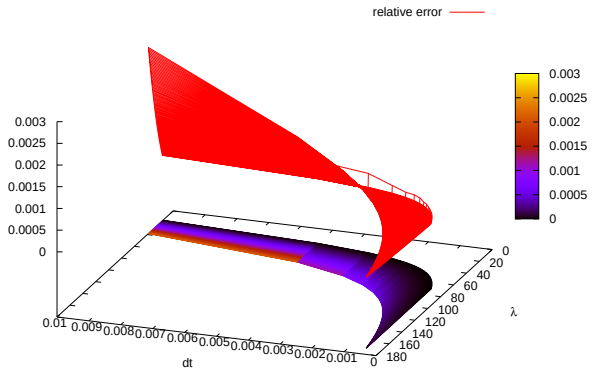

Fig. 4. Relative error against dt and λ, where $\lambda = \Delta Z / \Delta t$.

However, with given error tolerance, the tree-equivalent EFD grid might not be the optimal setting.

Figure 4 shows a typical graph of relative error between the Black Scholes formula result and the EFD result for the same at-the-money European option as in Figure 5 using double precision arithmetic. It can been seen that, the absolute error decreases monotonically as Δt and $\Delta Z = \Delta t \times \lambda$ becomes smaller. This means, as long as the result accuracy meets the accuracy requirement, it is possible to reduce the number of computational steps by increasing both Δt and λ. In addition, since it has been shown that the continuous $(\Delta t, \lambda)$ space has an analytical mapping to the stencil coefficients space in Figure 5, we have defined a valid search space for step two without affecting result accuracy. It is worth mentioning that, similar to Figure 5, the boundary error becomes significant when ΔZ becomes small enough and Equations 7 no longer hold.

4. OPTIMISATION APPROACHES

In this section we discuss two approaches for steps two and three of our optimisation process described in Section 3.

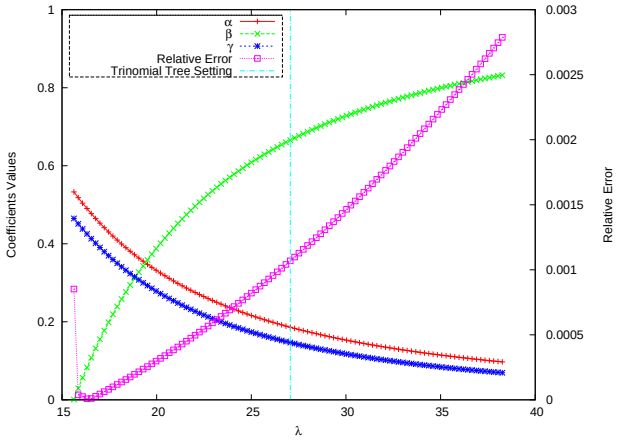

Fig. 5. The relationship between values of the coefficients $\{\alpha, \beta, \gamma\}$ and λ (left y-axis), where $\lambda = \Delta Z/\Delta t$; and the corresponding relative error between the EFD result and the Black Scholes formula result (right y-axis). Note that in this case the European option has the following parameters: $S = 70, K = 70, t = 3.0, r = 0.05, \sigma = 0.9, \Delta t = 0.01$.

The first approach tries to map directly from hardware to the EFD algorithm. Knowing a set of constants $(c_\alpha, c_\beta, c_\gamma)$ which yield small constant multipliers, this approach tries to find appropriate Δt and ΔZ which makes the stencil coefficients as close to these constants as possible, so as to reduce hardware resource usage by a dynamic kernel. A multidimensional minimisation problem can be formed, where variables Δt and ΔZ are the search space and r and σ are problem dependent variables. The target function of the minimisation problem is shown in Algorithm 1 which returns a utility value u; we minimise u to find appropriate stencil coefficients for the EFD grid. This algorithm implicitly assumes that the amount of hardware resource consumed by the constant multiplier is closely correlated to the hamming weight of the constant. In addition, Algorithm 1 requires all the three coefficients to be close to a 'nicer' set of constants, which is not always possible since coefficients α, β and γ are not free variables; in fact, according to Figure 5 the three coefficients has a one-to-one mapping to each other. Therefore the valid search space is limited. Moreover, since the user has no control over Δt or ΔZ in this case, a valid set of coefficients might result in extremely small Δt and ΔZ, which may increase the problem size significantly and offset any benefit gain by this optimisation routine.

The second approach is designed to reduce the number of computational steps needed with given accuracy requirement. Algorithm 2 shows a target function of a multidimensional minimisation process to match the EFD stencil stencil skewness and kurtosis to the theoretical values.

Algorithm 1 Simple Target Function

Input: Δt = size of time step, ΔZ = size of asset price step
Constant: r and σ
Output: u = utility value for the optimisation routine

1: Calculate (α, β, γ): use Equation 2
2: Obtain the numerical differences ε_α, ε_β and ε_γ from ideal constants (c_α, c_β, c_γ): $\varepsilon_{\alpha/\beta/\gamma} = \alpha - c_{\alpha/\beta/\gamma}$
3: Accumulate penalty p_1: to minimise ε_α, ε_β and ε_γ
4: Accumulate penalty p_2: to ensure the constraints (Equations 7) are met
5: Calculate utility value u: $u = p_1 + p_2$

$$Skew' = \frac{\alpha \times (dZ - \mu)^3 + \beta \times (-\mu)^3 + \gamma \times (dZ + \mu)^3}{Var'^{3/4}} \tag{13}$$

$$Kurt' = \frac{\alpha \times (dZ - \mu)^4 + \beta \times (-\mu)^4 + \gamma \times (dZ + \mu)^4}{Var'^2} \tag{14}$$

Algorithm 2 Moment Matching Target Function

Input: Δt = size of time step, ΔZ = size of asset price step
Constant: r and σ
Output: u = utility value for the optimisation routine

1: Calculate $(\mu, Var, Skew, Kurt)$: obtain mean, variance, skewness and kurtosis using the moment-generating function of a standard normal distribution [9], e.g. μ and Var as in Equations 9, $Skew = 0$ and $Kurt = 3$
2: Calculate (α, β, γ): by solving Equation 10
3: Calculate $Skew', Kurt'$: the skewness and kurtosis of the stencil using the standard statistical approach based on data points $(f_{i+1,j-1}, f_{i+1,j}, f_{i+1,j+1})$ and their corresponding probabilities (α, β, γ), an example is shown shown in Equation 13 and Equation 14
4: Calculate the difference between theoretical and stencil third and fourth order moments $\varepsilon_{Skew}, \varepsilon_{Kurt}$: $\varepsilon_{Skew} = |Skew - Skew'|$, $\varepsilon_{Kurt} = |Kurt - Kurt'|$
5: Calculate u: based on $\varepsilon_{Skew}, \varepsilon_{Kurt}$ and α, β, γ

The EFD grid setting found by Algorithm 2 guarantees fast convergence of the final result, since the setting makes the grid equivalent to a trinomial tree. However, the algorithm fails to address the hardware resource consumption of the constant multipliers. To overcome this limitation, we define a resource estimation function $res(\mathbb{R}) \rightarrow \mathbb{N}$, which, given a constant coefficient, returns a positive integer representing the amount of hardware resource used by the corresponding constant multiplier. Since each $(\Delta t, \Delta Z)$ combination can be mapped to the stencil coefficients analytically, it is possible to search in a limited $(\Delta t, \Delta Z)$ space

for nicer constants such that $dt \in [\Delta t - \varepsilon_1, \Delta t]$ and $dZ \in [\Delta Z - \varepsilon_2, \Delta Z]$, where ε_1 and ε_2 are small constants such that both the number of time steps N and the number of asset price steps M in the EFD grid stays the same as the original tree-equivalent grid setting.

We now analyse the statistical possibility of a smaller dynamic kernel by searching in the limited $(\Delta t, \Delta Z)$ space. To begin with, we assume that, due to complex optimisation techniques applied, the output (N_{LUT}) of the resource estimation function of constant multipliers is a uniformly distributed random variable with L possible outcomes, where L depends on (a) the number of adders used to implement the constant multiplier which is directly related to the number of bits in the chosen number representing format (b) the implementation detail of the constant multiplier library in use. For simplicity it is assumed that $N_{LUT} \in \{1...\mathrm{L}\}$, therefore the probability of getting a particular reconfigurable setting in a kernel with N_{mul} multipliers is $C(N_{LUT}) = \prod_{i=1}^{N_{\mathrm{mul}}} P(i)$, where $i \in \{1...\mathrm{N_{mul}}\}$ is the index of a multiplier, and $P(i)$ is the probability of the compiler generating a multiplier i which consumes N_{LUT} resource. Since N_{LUT} is uniformly distributed, $P(i) = 1/L$. As a result, the possibility to get a dynamic kernel with three coefficients each consuming N_{LUT} or less hardware resource is $(1/L)^3$. Therefore, if Δt and ΔZ are free variables, on average we can find a set of constants which halves the hardware resource consumption (with probability $(1/2)^3$) by trying eight different combinations. Although in our case Δt and ΔZ have a limited range, a local optimal can still be found. We therefore propose an improved version of the second approach, as shown in Algorithm 3.

To make the best use of the two approaches, we propose a workflow to find small constant multipliers for EFD dynamic kernels, while reducing the number of computational steps required with given accuracy requirement. We assume that an option pricing problem $o \in \kappa$, the true value of the option $v \in \mathbb{R}$, an error tolerance level $e \in \mathbb{R}$, and a set of stencil coefficients $c \in (\mathbb{R}^3)$ that yields small constant multipliers are known before hand. The workflow comprises the following steps:

(1) Label the first approach with index 1. Based on o and c, use Algorithm 1 to find ΔZ_1 and Δt_1; then derive the stencil coefficients $(\alpha_1, \beta_1, \gamma_1)$, the number of time steps N_1, and the number of asset steps M_1 for the EFD grid. Calculate the number of computational steps $c_1 = M_1 \times N_1$ in the grid.

(2) Based on results from (1), calculate the EFD option price v_1, and find the error $e_1 = v - v_1$. If $e_1 < e$, add index 1 to a set s which stores indices of valid approaches.

(3) If $e_1 < e$, use the stencil coefficients from (1) and the resource estimation function, to estimate the resource consumption per EFD kernel $r_1 = res(\alpha_1) + res(\beta_1) + res(\gamma_1)$. We have now obtained indicators c_1, e_1 and r_1 for the first approach.

(4) Repeat the first three steps but label the base case with index 0 and use Algorithm 2, to obtain indicators c_0, e_0 and r_0 for the tree-equivalent EFD grid, add index 0 to s if $e_0 < e$.

(5) Similarly, use index 2 and Algorithm 3 to obtain indicators c_2, e_2 and r_2 for the second approach, then add index 2 to s if $e_2 < e$.

(6) Obtain area-computation-time product $d_i = c_i \times r_i$, $i \in s$, choose approach i with $d_i = \min(d_j | j \in s)$.

Algorithm 3 Improved Algorithm

Input: Δt = size of time step, ΔZ = size of asset price step
Constant: r, σ, ε_1 and ε_2
Output: α, β, γ = the coefficients that yield smallest constant multipliers, without affecting the number of computational steps required for a given accuracy

```
Calculate Δt, ΔZ: using Algorithm 2 to obtain Δt and ΔZ which yield a tree equivalent EFD scheme
Increasing both Δt and ΔZ to find an EDF grid setting with less computational steps that meets the given accuracy requirement
Set minimal hardware resource consumption u = ∞
for dt=Δt to Δt − ε1 do
  for dZ=ΔZ to ΔZ − ε2 do
    Calculate local α′, β′, γ′: using dt and dz
    Calculate local hardware resource consumption u′: u′ = res(α′) + res(β′) + res(γ′)
    if u′ < u then
      α = α′, β = β′, γ = γ′
    end if
  end for
end for
```

It is worth considering the validity condition for applying the optimising approaches, given that the optimised solution should not be slower than the original one. The execution time T_d of the original dynamic design can be calculated as:

$$T_d = \frac{c_d \times t_d}{p_d} + t_{r,d} \tag{15}$$

where c_d is the number of computational steps, t_d is the cycle time, p_d is the number of processing elements (kernels) in the dynamic design, and $t_{r,d}$ is the reconfiguration time. Similarly, the execution time T_{opt} of the corresponding optimised design can be calculated as:

$$T_{opt} = \frac{c_{opt} \times t_{opt}}{p_{opt}} + t_{r,opt} + t_{search} \tag{16}$$

where c_{opt}, t_{opt}, p_{opt} and $t_{r,opt}$ denote the same parameters as above for the optimised design, and t_{search} denotes the time it takes to search for the better constants. For simplicity we assume the optimisation process has negligible impact on cycle time, hence $t_d = t_{opt} = t_{clk}$. To achieve a performance benefit, the following condition is required:

$$T_{opt} < T_d$$

$$\frac{c_{opt} \times t_{clk}}{p_{opt}} + t_{r,opt} + t_{search} < \frac{c_d \times t_{clk}}{p_d} + t_{r,d}$$

$$t_{search} < t_{clk}\left(\frac{c_d}{p_d} - \frac{c_{opt}}{p_{opt}}\right) + (t_{r,d} - t_{r,opt}) \qquad (17)$$

In this experiment the search time t_{search} is negligible since it is possible to search for the better constants beforehand for each option and use table lookup to load designs at runtime, just like the original dynamic design. We therefore have:

$$0 < t_{clk}\left(\frac{c_d}{p_d} - \frac{c_{opt}}{p_{opt}}\right) + (t_{r,d} - t_{r,opt}) \qquad (18)$$

If $c_d > c_{opt}$, $p_d < p_{opt}$ and $t_{r,d} > t_{r,opt}$, the validity of the optimisation process is guaranteed. In our experiment, c depends on N and M, and p and t_r depends on the reconfigurable area. Therefore, if the optimisation process and reduce both the reconfigurable area and N and M, it will benefit the overall execution time.

5. RESULTS

In this section we examine the effectiveness of our workflow discussed in Section 4. We use the Nelder-Mead multivariable minimisation routine in GNU scientific library to implement our algorithms. The fixed point error analysis is based on the MPFR library [10]. The result is produced according to the error tolerant level used in industry [2], which requires the difference between the reduced precision result and the double precision result to be smaller than $2E-4$; however, the workflow can be tuned to accommodate arbitrary error tolerant level by adjusting the number format. The 23 bit fixed point number format with 1 bit for integer and 22 bit for fraction is used in our implementations, as it has been used in our previous work [4] and the finite precision error in the result is always below $2E-4$ compared to double precision floating point result for the EFD computation. The FloPoCo library [11] is used to generate dynamic kernel descriptions with different stencil coefficients in VHDL. The FloPoCo library is also used as the resource estimation function in our workflow. All VHDL kernel descriptions are placed and routed on a Xilinx Virtex-6 XC6VLX760 FPGA using ISE 13.2 implementation tools. All dynamic designs found by our workflow are compared to the 23 bit fixed point version of the unoptimised dynamic designs presented in [4].

[2]Private correspondence with J.P. Morgan Chase

	Rel. Err.	LUTs	N	M	Comp. Steps
Double Static	1.55E-4	13759	365	420	1.36E5
Original 23bit Dynamic	1.52E-4	732	365	420	1.36E5
First Approach	1.2E-7	881	1146	1818	2.08E6
Second Approach	1.41E-4	603	365	324	1.05E5

Table 1. Comparison between the two approaches discussed in Section 4. The target European option has the following κ: $(70, 70, 0.05, 1.0, 0.3)$.

We first give an example for choosing between the two approaches in our workflow shown in Section 4, in terms of relative error, hardware resource consumption and number of computational steps. We assume that the number of timesteps (N) is provided by the user and therefore dt is fixed. The results and a static double precision EFD kernel implementation reference are presented in Table 1. The relative error is obtained by comparing the reduced precision results from the four designs to the double precision result from the Black Scholes formula. It can be seen that although the first approach has the smallest relative error, it requires over 10 times more computational steps compared to the second approach, since the first approach requires a two dimensional search space $(\Delta t, \Delta Z)$, we have no control over the total number of grid points or result precision. The LUTs result in Table 1 shows the place and route result of the kernels under consideration; it is clear that the hamming weight is not directly correlated to the amount of LUT resource per dynamic kernel, since the number of LUT consumed by the kernel generated using the first approach is larger than the original kernel. On the other hand, the second approach can reduce the amount of hardware resource consumption as well as the number of computational steps, compared to the original EFD kernel; it also provides the same level of accuracy compared to the original scheme. Therefore the dynamic kernel generated by our second approach is chosen for our workflow.

We now analyse the upper bound of reconfigurable area of the dynamic EFD kernels found by our workflow, and try to prove that it can reduce the required amount of preserved reconfiguration area, as well as reduce the number of computational steps. One thousand American options are randomly generated with different parameters covering the types of parameters observed in the market; and each is solved by an EFD grid with $N = 365$, which is a common industry setting for options with time-to-maturity of one year under daily observation.

In the original tree-equivalent EFD grid setting the optimal M is a constant equal to 420; and in our workflow M is determined by the error tolerance. All EFD results are compared to the standard double precision EFD result with $N = 2000$, which we consider to be the true result, to make sure the error tolerance is not exceeded. To minimise the noise from discretisation error, we assume that all options are at the money so that the strike price lies exactly on the

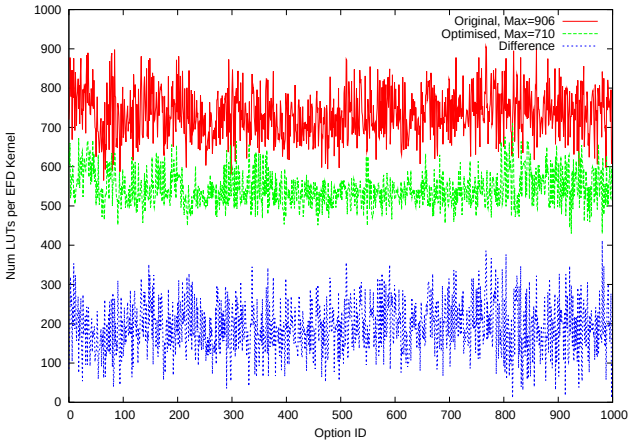

Fig. 6. Comparison of LUT usage per kernel between the standard and optimised EFD kernel, based on place and route result.

grid. Figure 6 shows the comparison of LUT usage per kernel between the original and optimised dynamic kernels. It can be seen that the optimises kernel always consume fewer LUTs than the unoptimised original kernels. The maximum LUT usage has been reduced from 906 to 710, indicating a reduction of 22%. Since N is fixed, the average number of computational steps depends only on M. Our result indicates that we have reduced the average of M from 420 to 324, which is a 23% reduction. As a result, the area-computation-time product is reduced by 40%.

6. CONCLUSION

This paper demonstrates a novel optimisation methodology to adjust stencil based numerical procedures from the algorithm level, so as to reduce both the hardware resource consumption per kernel and the amount of computation needed with given accuracy requirement, when mapping the algorithm to reconfigurable hardware using dynamic constant reconfiguration. As a result, less area is needed to support run-time reconfiguration, and less computational steps are required in the numerical procedure to obtain a result with given error tolerance.

In a case study on a Virtex-6 XC6VLX760 FPGA, by comparing one thousand randomly generated 23 bit fixed point implementations after optimisation to those before optimisation, the upper bound of reconfiguration area is reduced by 22%; the average number of computational steps is reduced by 23%; and the area-computation-time product is reduced by 40%; while the numerical errors of the results are kept below the error tolerant level used in industry.

Current and future work involves using more sophisticated hardware estimation tools such as [12] and [13] to improve the resource estimation function. Our long term objective is to extend the proposed optimisations to address trade-offs in speed, area, numerical accuracy, power and energy efficiency for a variety of applications and devices.

Acknowledgement. The research leading to these results has received funding from European Union Seventh Framework Programme under grant agreement number 287804, 248976 and 257906. The support by UK EPSRC, the HiPEAC NoE, J.P. Morgan Chase, Maxeler and Xilinx is gratefully acknowledged.

7. REFERENCES

[1] J. Hull, "Options, futures and other derivatives," Prentice Hall, 2005.

[2] Q. Jin, D. Thomas, and W. Luk, "Exploring reconfigurable architectures for explicit finite difference option pricing models," in *Proc. Int. Conf. on Field Programmable Logic and Applications*, 2009, pp. 73–78.

[3] T. Becker, W. Luk, and P. Cheung, "Energy-aware optimisation for run-time reconfiguration," in *Proc. Int. Symp. on Field-Programmable Custom Computing Machines (FCCM)*, 2010, pp. 55 –62.

[4] T. Becker, Q. Jin, W. Luk, and S. Weston, "Dynamic constant reconfiguration for explicit finite difference option pricing," in *Proc. Int. Conf. on ReConFigurable Computing and FPGAs*, 2011, pp. 176–181.

[5] M. J. Brennan and E. S. Schwartz, "Finite difference methods and jump processes arising in the pricing of contingent claims: A synthesis," *Journal of Financial and Quantitative Analysis*, vol. 13, pp. 461–474, 1978.

[6] D. J. Duffy, *Finite Difference Methods in Financial Engineering*. Wiley, 2006.

[7] A. H. Tse, G. C. Chow, Q. Jin, D. Thomas, and W. Luk, "Optimising performance of quadrature methods with reduced precision," in *Proc. Int. Workshop on Reconfigurable Computing: Architectures, Tools and Applications*, 2012.

[8] G. M. Jabbour, M. V. Kramin, T. V. Kramin, and S. D. Young, "Multinomial lattices and derivatives pricing," World Scientific Publishing, 2005.

[9] C. M. Grinstead and J. L. Snell, "Introduction to probability," American Mathematical Society, 1997.

[10] L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier, and P. Zimmermann, "Mpfr: A multiple-precision binary floating-point library with correct rounding," *ACM Trans. Math. Softw.*, vol. 33, 2007.

[11] F. De Dinechin, J. Detrey, C. Octavian, and R. Tudoran, "When FPGAs are better at floating-point than microprocessors," Laboratoire de lInformatique du Paralllisme research report RR2007-40, 2007.

[12] C. Shi, J. Hwang, S. Mcmillan, A. Root, and V. Singh, "A system level resource estimation tool for FPGAs," in *Proc. Int. Conf. on Field Programmable Logic and its applications (FPL)*, 2004.

[13] P. Schumacher and P. Jha, "Fast and accurate resource estimation of RTL-based designs targeting FPGAs," in *Proc. Int. Conf. on Field Programmable Logic and Applications (FPL)*, 2008, pp. 59 –64.