Optimising Explicit Finite Difference Option Pricing For Dynamic Constant Reconfiguration

Qiwei Jin*, David Thomas^, Tobias Becker*, and Wayne Luk*

*Department of Computing,
^Department of Electrical and Electronic Engineering,
Imperial College London

29 August 2012

Contributions

1. Novel optimisation for Explicit Finite Difference (EFD):
   - **preserves** result accuracy
   - **reduces** hardware resource consumption
   - **reduces** number of computational steps
2. Two approaches to minimise:
   - **hardware resource** utilisation
   - **amount of computation** required in the algorithm
3. Evaluation: 40% reduction in area-time product
   - 50%+ faster than before optimisation
   - 7+ times faster than static FPGA implementation
   - 5 times more energy efficient than static FPGA implementation

Background

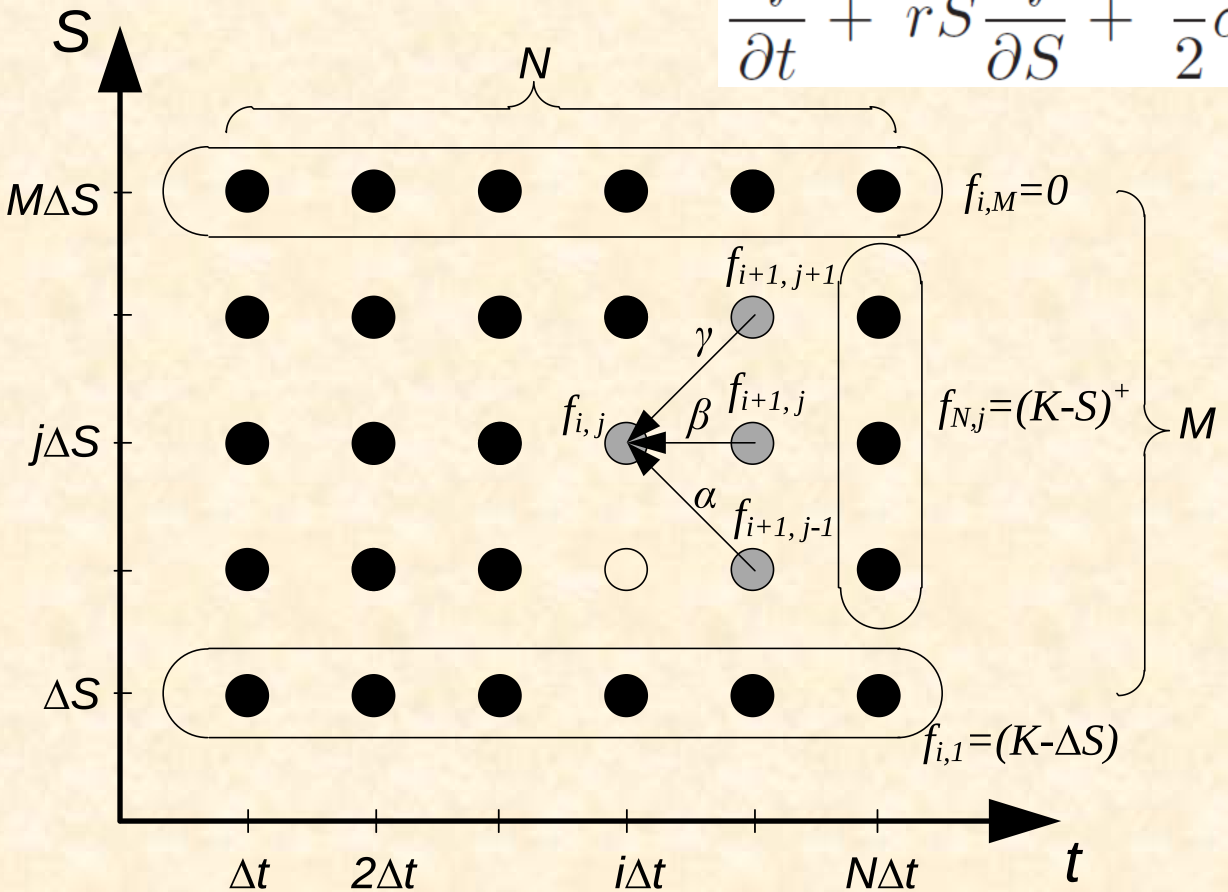
- Financial put option
  - gives the owner the right but not the obligation to sell an asset ***S*** to another party at a fixed price ***K*** at a particular time ***T***

- Explicit Finite Difference (EFD) Method
  - useful numerical technique to solve PDEs
  - used for options with no closed-form solution
  - Discretises over asset price space (***S***) and time (***t***), and maps them onto to a grid

Background

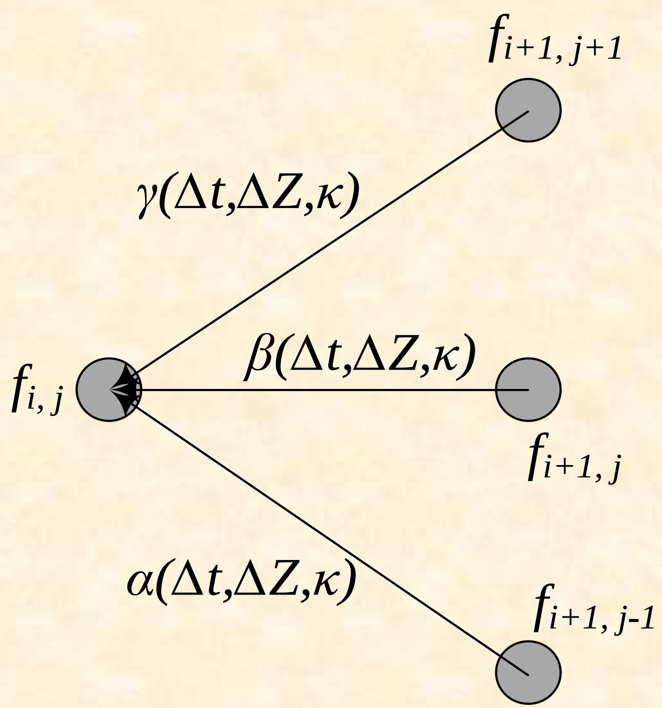
- The Black Scholes PDE and the EFD grid

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



Background

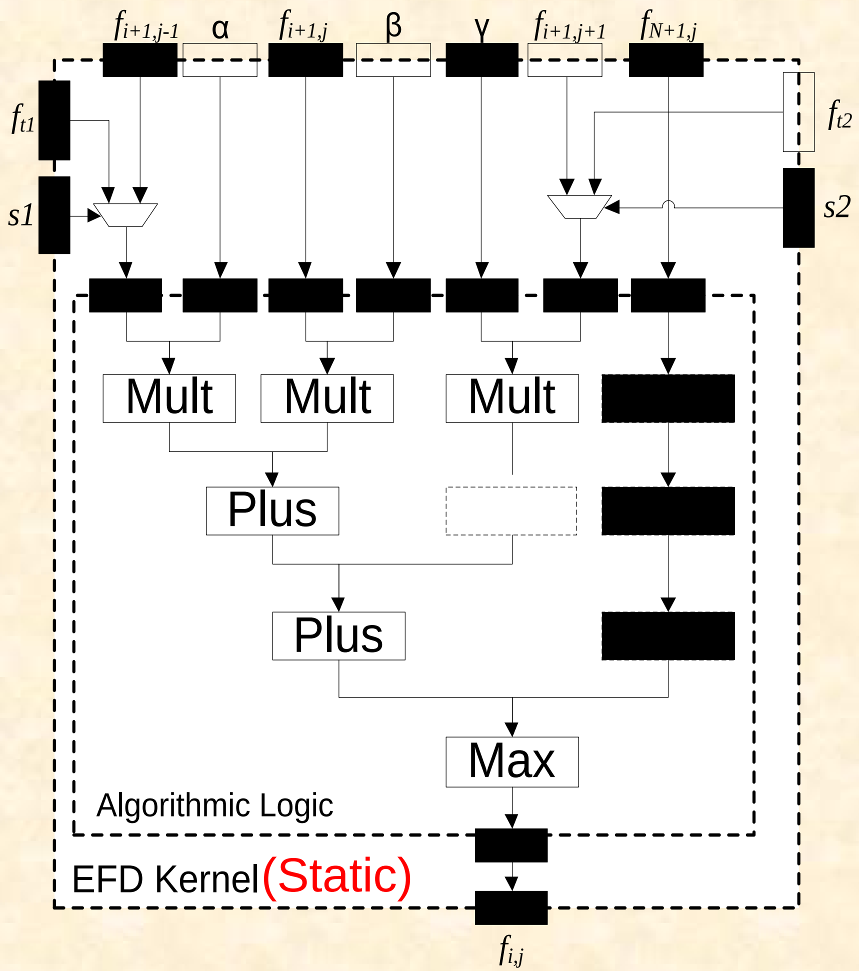
- The entire EFD grid
  - updated from right to left, backwards in time ***t***
  - updated by a stencil with coefficients α, β and γ



Z=lnS, κ≡(S, K , r, T, σ)

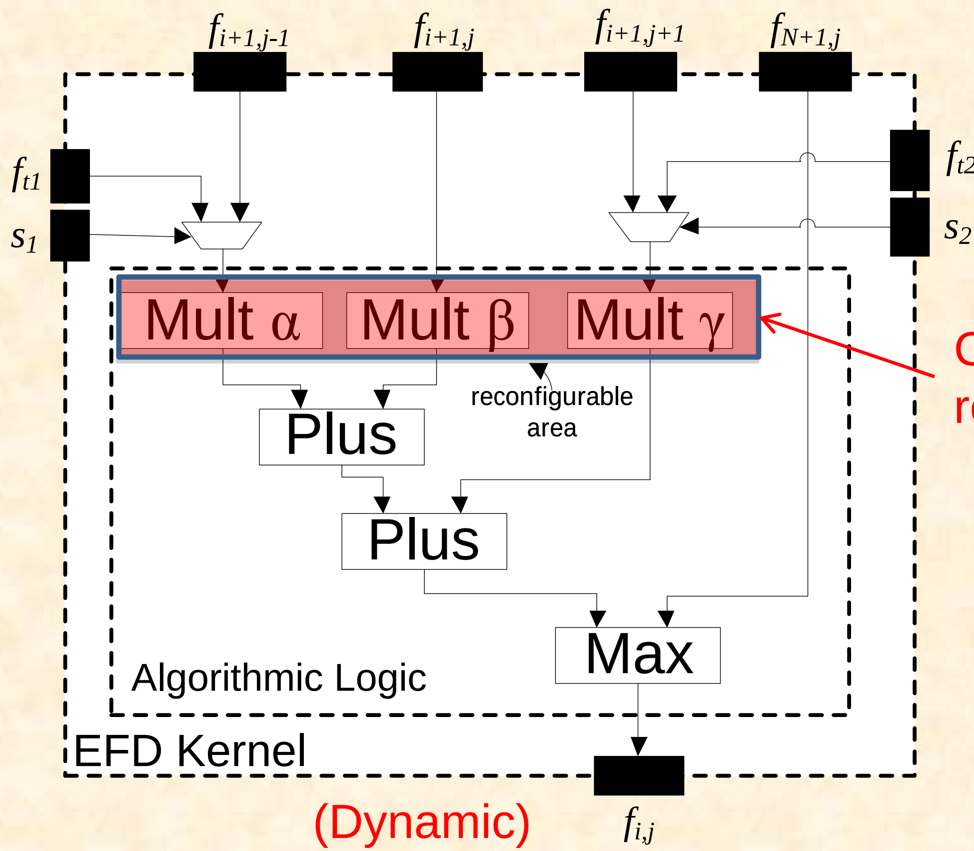
Background

- Stencil computation mapped to FPGA



Background

- Dynamic constant reconfiguration[$]
  - **reduce** area, power and **improve** performance



Constant multipliers, reconfigure to update

(Dynamic)

$ “Dynamic Constant Reconfiguration for Explicit Finite Difference Option Pricing”, Becker et, all

Two Optimisation Methods

Two methods to optimise high performance designs

1. Use custom data formats*
   - to reduce area
   - preserve sufficient result accuracy
2. Use constant specialisation and reconfiguration $
   - further reduce area and energy consumption
3. **Is there a third method?**

* “A mixed precision Monte Carlo methodology for reconfigurable accelerator systems”, Chow et. all
$ “Dynamic Constant Reconfiguration for Explicit Finite Difference Option Pricing”, Becker et. all

1. Novel Optimisation Method

- To obtain the best trade-off:
  - **A. reduce hardware resource consumption**
    - set the EFD grid carefully
    - minimise resource usage of constant multipliers
  - **B. reduce the number of computational steps required in the EFD grid**
    - ensure the result meets the accuracy requirement
    - avoid unnecessary computation

Novel Optimisation Method: Approach

- Normalising the option coefficients
  - fixed point datapaths used instead of floating point ones
  - less hardware resources consumed
- Identifying efficient fixed point constants
  - yield smaller constant multipliers
  - adjust the EFD algorithm to make use of them
- Reducing number of computational steps
  - make sure it is smaller than the original and
  - preserve result accuracy

Normalising Option Coefficients

- Option descriptor $\kappa \equiv (S, K, r, T, \sigma)$
- Option price f is unbounded
  - Since $0 < K < \infty$ and $0 < S < \infty$
- Bits are wasted in fixed point
  - not all integer bits utilised all the time
- κ can be normalised

  $\kappa' \equiv (S/K, 1, rT, 1, \sigma\sqrt{T})$ and $f = Kf'$
- $0 \leq f' \leq 1$, f' is bounded
  - bits are fully utilised

2. Efficient Fixed Point Constants

This will be discussed in two parts

1. **Possibility of finding efficient constants**

2. Two approaches to scan the search space
   A. minimise hardware resource utilisation
   B. minimise the amount of computation

Possibility of Finding Efficient Constants

- Assuming three B-bit fixed point constant multipliers are generated
  - based on coefficients α, β and γ
  - each use $N_\alpha, N_\beta, N_\gamma$ units of resource (i.e. in LUTs)
  - $N_\alpha \in L, N_\beta \in L, N_\gamma \in L$
  - L is a set containing all possible outcomes of resource consumption of B-bit constant multipliers
  - N_L, the size of L depends on the implementation details of the fixed point library
  - $P(N_\alpha = x) = P(N_\beta = y) = P(N_\gamma = z) = 1/N_L$ where $x,y,z \in L$
    - due to complex optimisation techniques applied
    - assuming $N_\alpha, N_\beta, N_\gamma$ are i.i.d. uniform random numbers in set L

Possibility of Finding Efficient Constants

- The probability of getting any N_α, N_β, N_γ combination is $(1/N_L)^3$
- The probability of finding a set of constants which halves the hardware resource is therefore 12.5% or $(1/2)^3$
- The probability to find 20% size reduction is 51.2% or $(4/5)^3$
- It is quite possible to find efficient constant combinations in the search space

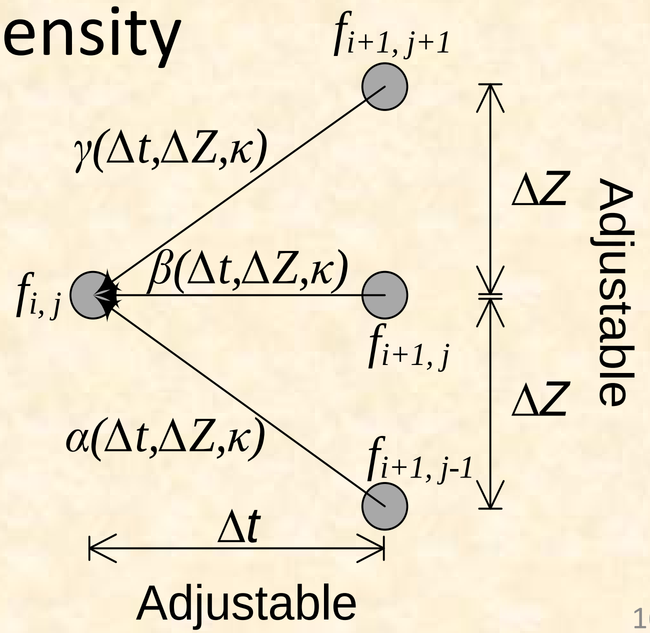
Efficient Fixed Point Constants

1. **Possibility of finding efficient constants**

2. Two approaches to scan the search space
   A. **minimise hardware resource utilisation**
   B. minimise the amount of computation

Scanning the search space

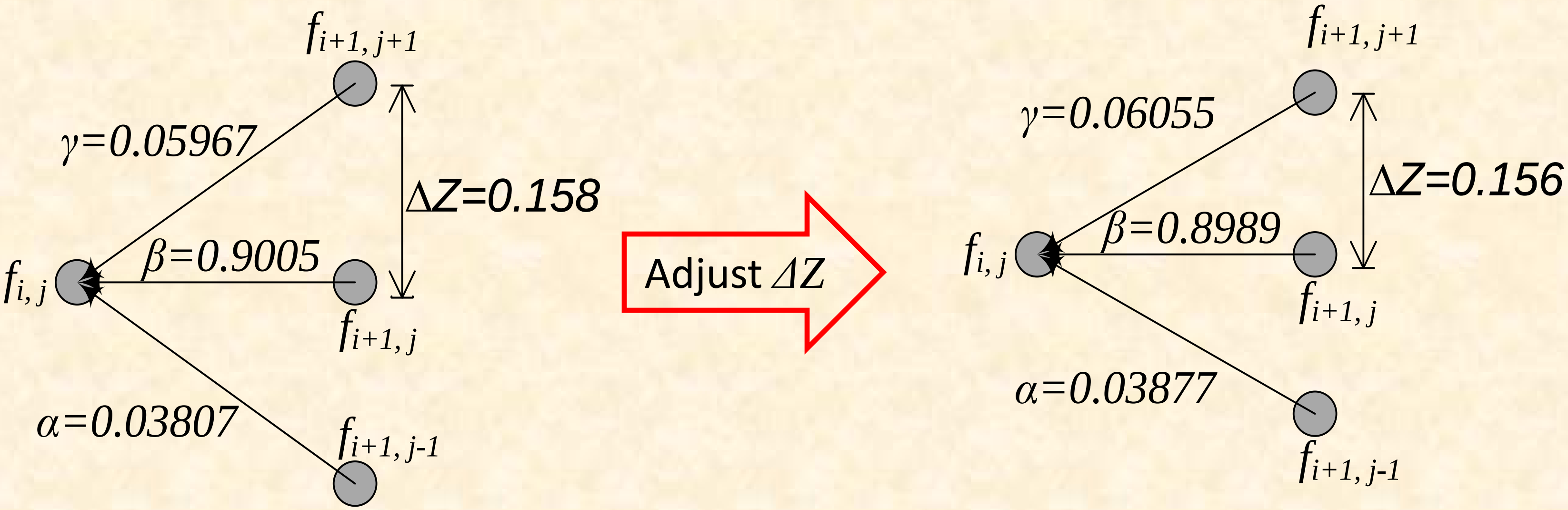
- Coefficients α, β and γ are flexible
  - certain properties must be met
  - ΔZ and Δt determines the coefficients
- Grid density can be adjusted
  - ΔZ and Δt determines grid density



Simple Example

Before

After



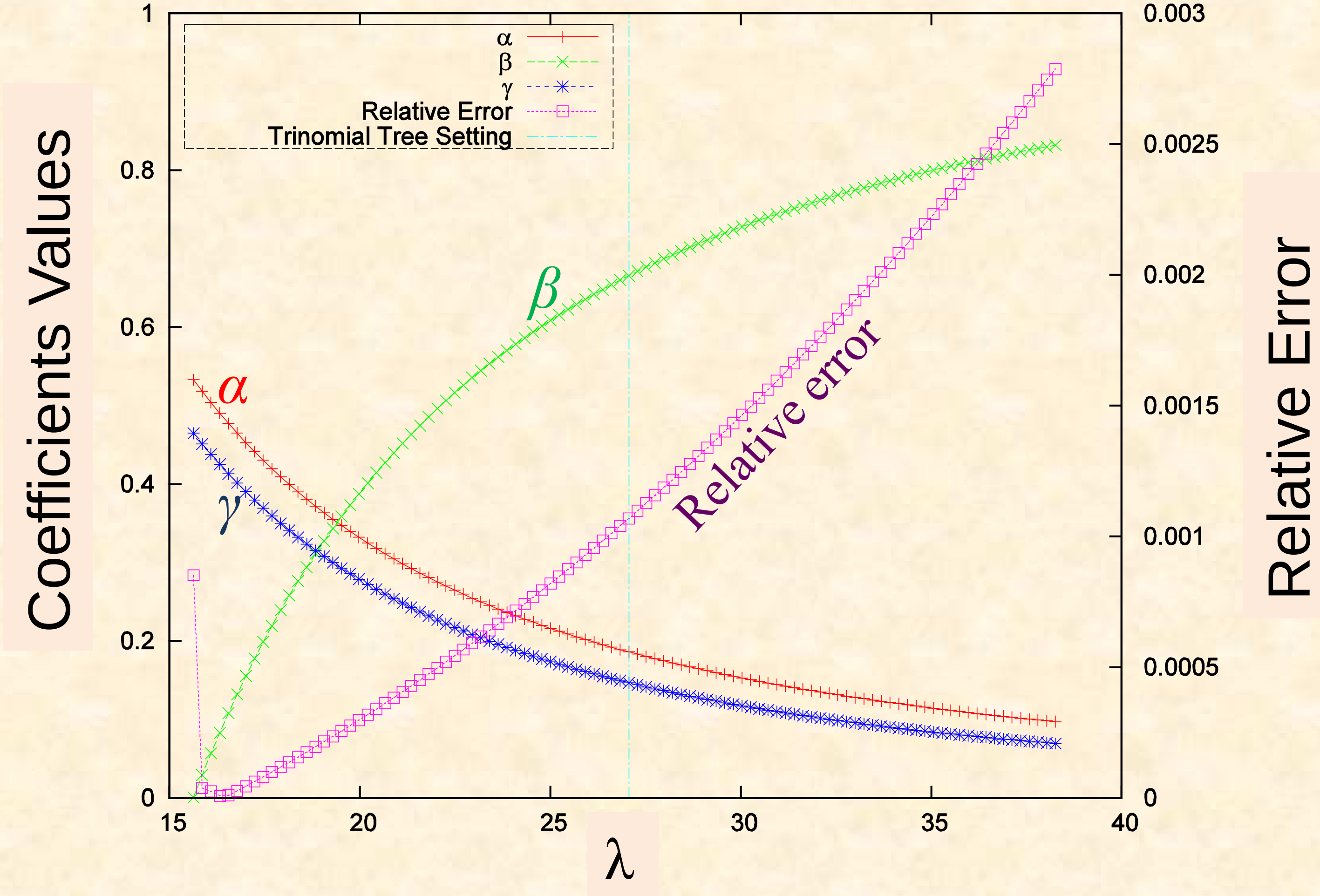
N_{LUT}: 533

N_{LUT}: 335

37% reduction in N_{LUT}

result based on flopoco p&r

Simple Example



$\lambda = \Delta Z / \Delta t$

Minimising Hardware Resource Utilisation

- Constants in the form of 2^E ($E \in Z$) leads to smaller multipliers
  - multipliers are implemented by shift operators in fixed point arithmetic
- Making α, β and γ close to form 2^E
  - corresponding multipliers **likely** to be smaller
  - assuming the hardware resource consumption is related to the hamming weight of the constant

Minimising Hardware Resource Utilisation

- The coefficients need to meet the following criteria:
  - $\alpha+\beta+\gamma=1,\ \alpha>0,\ \beta>0,\ \gamma>0$
  - $\mu(r, \sigma, \Delta t) = \mu'(\alpha, \beta, \gamma, \Delta Z)$
    - *mean, 1st moment*
  - $Var(\sigma, \Delta t) = Var'(\alpha, \beta, \gamma, \Delta Z)$
    - *variance, 2nd moment*
  - $Skew = Skew'(\alpha, \beta, \gamma, \Delta Z)$
    - *skewness, 3rd moment*
  - $Kurt = Kurt'(\alpha, \beta, \gamma, \Delta Z)$
    - *Kurtosis, 4th moment*

Statistical Moments should be equal to EFD Moments

Minimising Hardware Resource Utilisation

- Make the EFD scheme converge
  - the criteria must be met
- A minimisation routine
  - search around a efficient set of constants
    - (i.e. $\alpha=2^{E1}$, $\beta=2^{E2}$, $\gamma=2^{E3}$ or 0.25, 0.5, 0.25)
  - force the criteria to be met
    - first and second moments must match
    - differences in third and fourth moments minimised
    - grid convergence is guaranteed

Minimising Hardware Resource Utilisation

- Pros:
  - possible to find very small constant multipliers
    - hardware consumption close to the 2^E multipliers
- Cons:
  - has no control over Δt or ΔZ
    - the EFD problem size can be unbounded
    - result accuracy is not guaranteed
  - requires a 2D search space ($\Delta t, \Delta Z$)
    - some times not possible in business
  - Works under assumption
    - hamming weight is related to hardware resource consumption

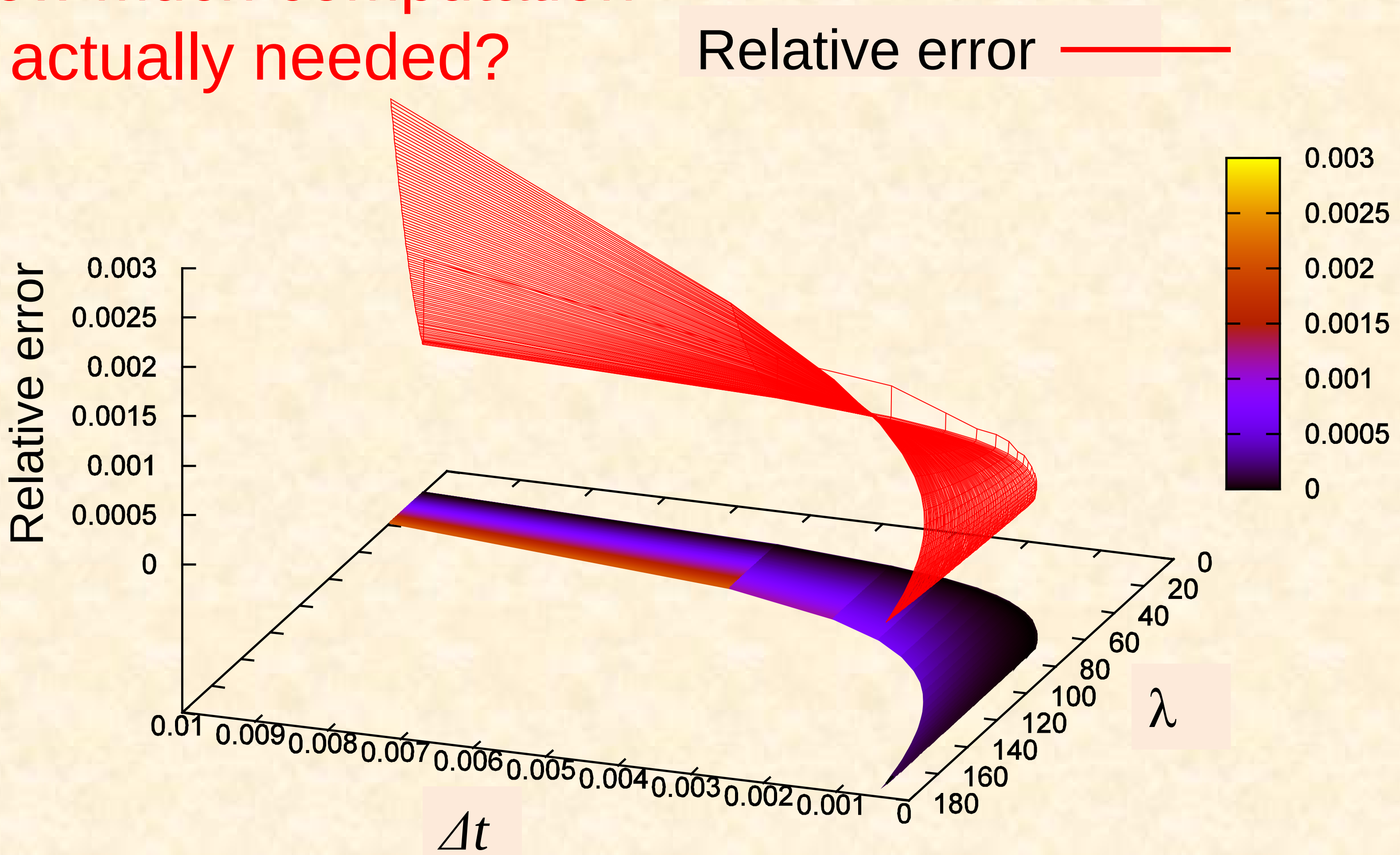
Efficient Fixed Point Constants

1. Possibility of finding efficient constants

2. Two approaches to scan the search space
    A. **minimise hardware resource utilisation**
    B. **minimise the amount of computation**

Minimising Amount of Computation

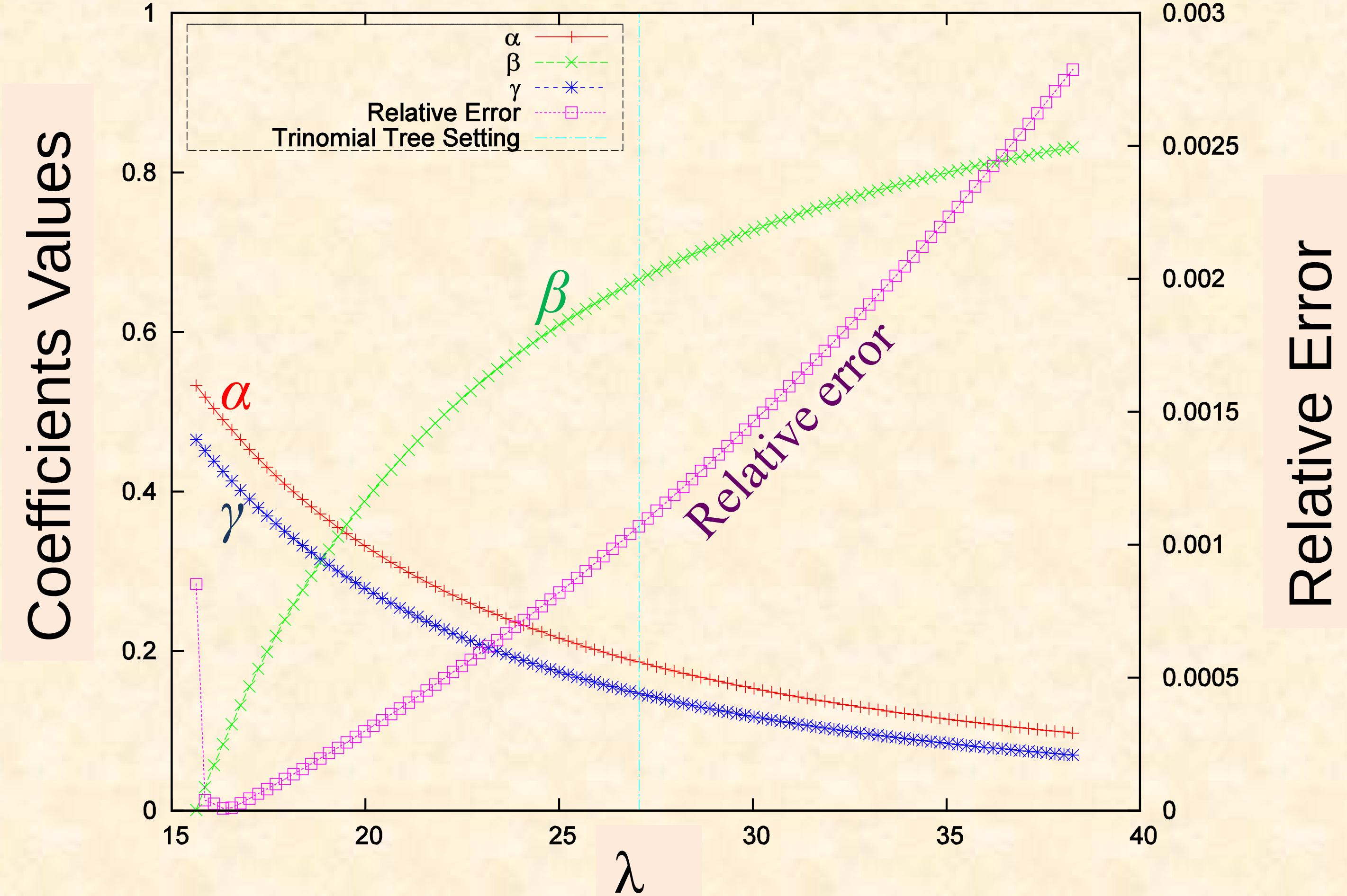
How much computation
is actually needed?



Relative error against Δt and λ, where $\lambda = \Delta Z / \Delta t$

Minimising Amount of Computation

- Δt is usually determined by business fact (fixed)
  - i.e. if options are exercised on a daily basis, $\Delta t = 1\ day$



Minimising Amount of Computation

- With Δt fixed, find a λ (i.e. ΔZ, since $\lambda = \Delta Z / \Delta t$)
  - must meet the result accuracy requirement
  - $\Delta Z > \Delta Z_{tree}$, amount of computation is reduced
- Search in $[\lambda - \varepsilon, \lambda]$
  - ε is a small number
  - find a set of coefficients with **local optimal hardware consumption**
  - **result accuracy is guaranteed** since relative error grows with λ

Minimising Amount of Computation

- Pros
  - amount of computation in the result EFD grid is minimised
  - Result accuracy is guaranteed
  - fast search under simple search space (1D), no minimisation routine required
- Cons
  - global optimal is NOT guaranteed

3. Evaluation: Implementation

- Xilinx Virtex-6 XC6VLX760 FPGA, ISE 13.2
- FloPoCo library for fixed point datapath
- MPFR library for fixed point error analysis
- Nelder-Mead multivariable routine in GNU scientific library for minimisation
- Place and route results are reported

Experiment Setting

- 23 bit fixed point number format is used
  - with 1 bit for integer and 22 bits for fraction
- Relative error tolerance is $2E-4$
  - compared to double precision result
  - same level of accuracy is used in industry
- Δt is set as $1/365$
  - Assuming daily observation in a 365-day year
- Result compared to our previous work
  - 23 bit fixed point dynamic designs – unoptimised

A Typical Case

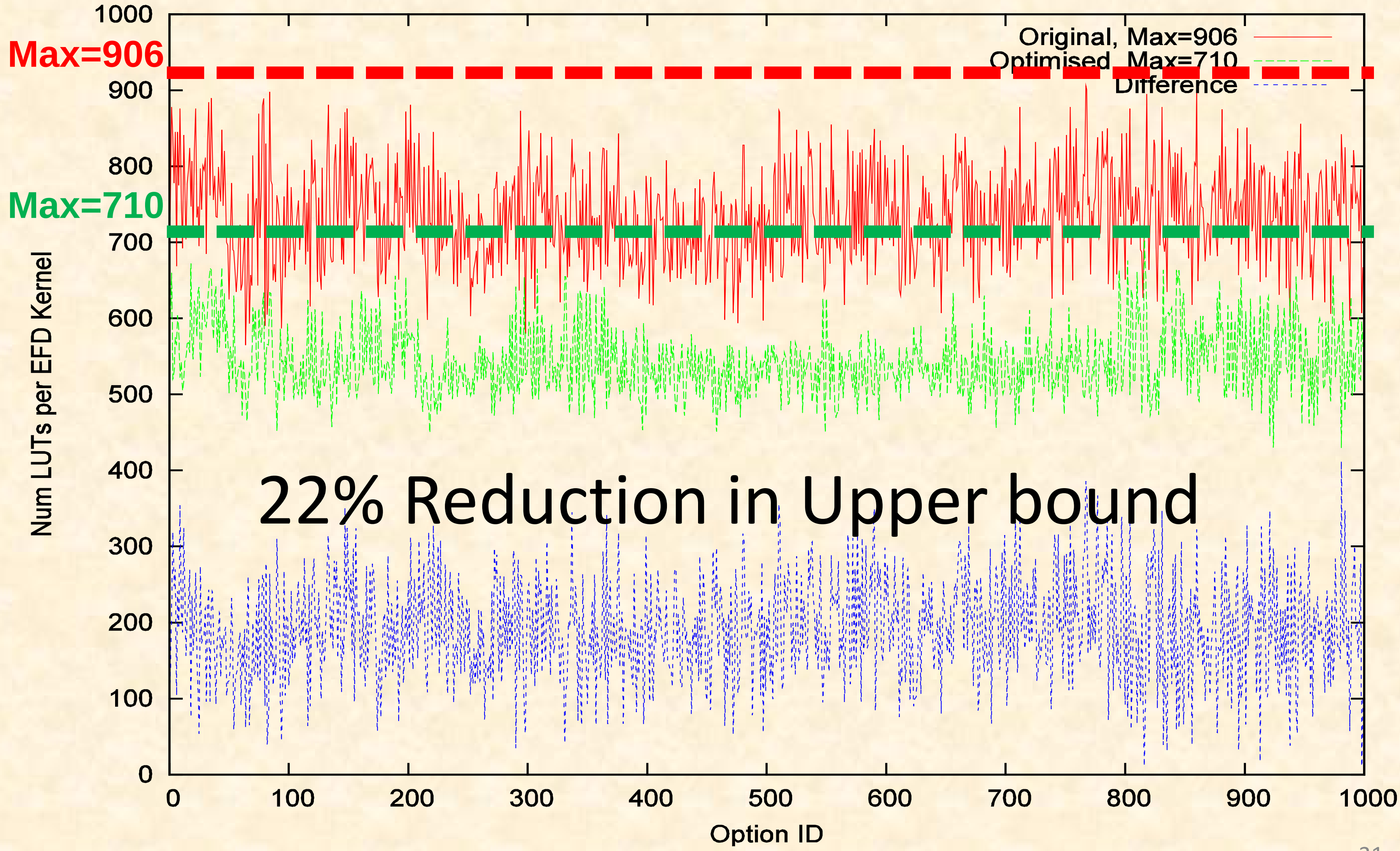
European option

	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

Result from implementations is compared to the Black Scholes formula result

$$\kappa \equiv (S, K, r, T, \sigma) = (70, 70, 0.05, 1.0, 0.3).$$

P&R Result: 1000 Typical Options



Future Work

- Use more sophisticated hardware estimation tools for better result
- Generalise the work to support more types of EFD schemes
- Long term: address trade-offs
  - in speed, area, numerical accuracy, power and energy efficiency
  - for a variety of applications and devices

Summary

1. Novel optimisation for Explicit Finite Difference (EFD):
   - **preserves** result accuracy
   - **reduces** hardware resource consumption
   - **reduces** number of computational steps
2. Two approaches to minimise:
   - **hardware resource** utilisation
   - **amount of computation** required in the algorithm
3. Evaluation: 40% reduction in area-time product
   - 50%+ faster than before optimisation
   - 7+ times faster than static FPGA implementation
   - 5 times more energy efficient than static FPGA implementation