Optimising Performance of Quadrature Methods with Reduced Precision

Anson H.T. Tse[1], Gary C.T. Chow[1], Qiwei Jin[1],
David B. Thomas[2], and Wayne Luk[1]

[1] Department of Computing, Imperial College London, UK
{htt08,cchow,qj04,wl}@doc.ic.ac.uk
[2] Department of Electrical and Electronic Engineering, Imperial College London, UK
d.thomas1@imperial.ac.uk

Abstract. This paper presents a generic precision optimisation methodology for quadrature computation targeting reconfigurable hardware to maximise performance at a given error tolerance level. The proposed methodology optimises performance by considering integration grid density versus mantissa size of floating-point operators. The optimisation provides the number of integration points and mantissa size with maximised throughput while meeting given error tolerance requirement. Three case studies show that the proposed reduced precision designs on a Virtex-6 SX475T FPGA are up to 6 times faster than comparable FPGA designs with double precision arithmetic. They are up to 15.1 times faster and 234.9 times more energy efficient than an i7-870 quadcore CPU, and are 1.2 times faster and 42.2 times more energy efficient than a Tesla C2070 GPU.

1 Introduction

Quadrature methods have been applied in different areas including pricing options [1], modeling credit risk [5], solving electromagnetic problems [14] and calculating photon distribution [8]. Using quadrature methods to price a single simple option is fast and can typically be performed in milliseconds on desktop computers. However, quadrature methods can become a computational bottleneck, for example, when a huge number of complex options are being revalued overnight under many different scenarios for risk management. Computational complexity also scales exponentially with the number of underlying assets, so accelerating multi-asset quadrature computation is a significant problem. Moreover, energy consumption of computation is a major concern when the computation is performed 24 hours a day, 7 days a week.

The ability to support customisable precision is an important advantage of reconfigurable hardware. Reduced precision floating-point operators usually have higher clock frequencies, consume fewer resources and offer a higher degree of parallelism for a given amount of resources compared with double precision operators.

O.C.S. Choy et al. (Eds.): ARC 2012, LNCS 7199, pp. 251–263, 2012.

The use of reduced precision affects the accuracy of the numerical results. However, with a higher throughput capacity using reduced precision, the integration grid spacing could be reduced which might actually increase accuracy. This paper introduces a novel optimisation methodology for determining the optimal combination of operator precision and integration grid spacing in order to maximize the performance of quadrature method on reconfigurable hardware. The major contributions of this paper include:

- optimisation modeling based on a step-by-step accuracy analysis and performance model. A discrete moving barrier option pricer is used as an example to graphically illustrate the analysis and to provide empirical evidence for the model (Section 3);
- a methodology and algorithms to determine the optimal mantissa bit-width and the integration grid density for a given integration problem by finding the Pareto frontier satisfying a given error tolerance level (Section 4);
- case studies of two financial applications and one benchmark quadrature problem using the proposed methodology, namely a discrete moving barrier option pricer, a 3-dimensional European option pricer, and a discontinuous integration benchmark (Section 5);
- performance comparison of the optimised FPGA implementation versus GPU and CPU. Our results show that the proposed approach increases performance by around 4 times, resulting in a total speed-up over double-precision software of 15.1 times while maintaining the accuracy. The optimised FPGA designs are 1.2 times faster and 42.2 times more energy efficient than comparable GPU designs (Section 6).

2 Background

There has been much interest in the use of accelerators such as FPGAs and GPUs for high performance computing. GPUs use the same type of floating-point number representation and operation as CPUs, namely IEEE-754 double precision and IEEE-754 single precision. Double precision has 53 bits of mantissa while single precision has 24 bits of mantissa. GPUs are shown to provide significant speedup over CPUs for many applications, especially when single precision is used [11]. For quadrature methods in option pricing, a GPU design using single precision running on NVIDIA Tesla C1060 demonstrated a speedup of 8.4 times while a Virtex-4 FPGA demonstrated a speed-up of 4.6 times over a CPU [20].

FPGAs provide customisable floating-point number operation which could be exploited to provide additional speedup. A mixed-precision methodology has shown to provide an additional performance gain of 7.3 times over an FPGA-accelerated collision detection algorithm [4]. Other works in bit-width optimisation aim to improve performance by using minimum precision in a data-path given a required output accuracy. One common approach is to develop an accuracy model which relates output accuracy to the precision of the data formats used in the data-path. The area and delay of data-paths with different precisions

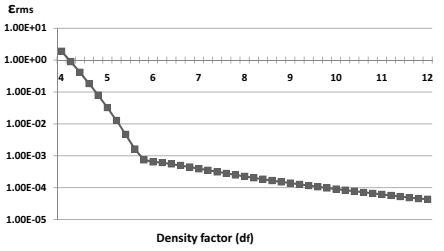

Fig. 1. The ϵ_{rms} for different d_f at m_w=53

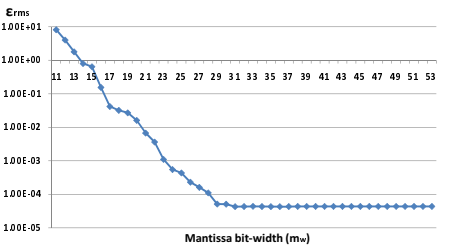

Fig. 2. The ϵ_{rms} for different m_w at d_f=12

are modelled based on the accuracy model. The design with a minimum area-delay product can be obtained from the models. Common accuracy modeling approaches include simulation [10], interval arithmetic [6], backward propagation analysis [7], affine arithmetic [12] [13] [16] [17], SAT-Modulo theory [9] and polynomial algebra [3].

Our novel precision methodology presented in this paper is specialised for solving quadrature problems. The optimal design is determined by optimising both the spacing between integration grid points and the precision of floating-point operators, instead of considering only the precision only in other common approaches.

Let us now briefly introduce quadrature methods: numerical methods for approximating an integral by evaluating at a finite set of integration points and using a weighted sum of these values. There are many different methods of numerical integral evaluation. Two of the most common methods are based on the trapezoidal rule and Simpson's rule [19]:

Trapezoidal Rule:

$$\int_a^b f(y)dy \approx \frac{\delta y}{2}\{f(a) + 2f(a+\delta y) + 2f(a+2\delta y)\cdots + 2f(b-\delta y) + f(b)\} \quad (1)$$

Simpson's Rule:

$$\int_a^b f(y)dy \approx \frac{\delta y}{3}\{f(a) + 4f(a+\delta y) + 2f(a+2\delta y) + \cdots + 4f(b-\delta y) + f(b)\} \quad (2)$$

3 Optimisation Modeling

The proposed optimisation objective function is based on a step-by-step accuracy analysis and performance modeling. A barrier option pricer is used as an example to illustrate the relationship between accuracy, throughput, integration grid density and the precision of floating-point operations.

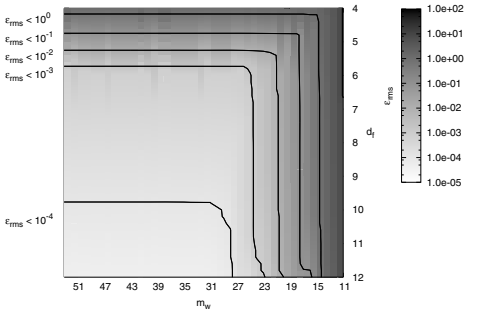

Fig. 3. The contour plot of ϵ_{rms} of barrier option pricer for different m_w and d_f

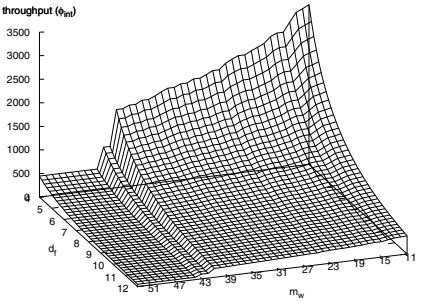

Fig. 4. The aggregated FPGA throughput

3.1 Accuracy Analysis

There are two sources of error affecting the "accuracy" of the integration result, namely integration error ϵ_{int} and finite precision error ϵ_{fin}. The total error ϵ_{total} is a function of both error sources. Integration error ϵ_{int} is the error due to having a finite number of integration points within an integration interval. Finite precision error ϵ_{fin} is the error due to non-exact floating-point arithmetic. Floating-point number representation in computer has a finite significant bit-width. The rounding of the intermediate or final result leads to precision loss. We define grid density factor $\boldsymbol{d_f}$ as a variable which is inversely proportional to the integration grid spacing and we define $\boldsymbol{m_w}$ as the number of bits in the mantissa. Therefore, we have $\epsilon_{int}(d_f)$ and $\epsilon_{fin}(m_w)$ respectively to represent their relationships.

To measure $\epsilon_{total}(m_w, d_f)$, the root-mean-squared error $\boldsymbol{\epsilon_{rms}(m_w, d_f)}$ comparing with a set of reference results is used. The set of reference results are computed using a large value of m_w and d_f.

We investigate the result of $\epsilon_{rms}(m_w, d_f)$ by computing a portfolio of 30 barrier options using different mantissa bit-width and density factor. The computed option values are compared with a set of reference values using $m_w = 53$ (double precision) and $d_f = 20$. Fig. 1 shows the graph of $\epsilon_{rms}(53, d_f)$. We can see that the total error is decreasing with respect to d_f. Fig. 2 shows the graph of $\epsilon_{rms}(m_w, 12)$. This figure shows that with a sufficient large density factor, the total error of the result decreases with increasing mantissa bit-width. In addition, this figure also indicates that at $d_f = 12$, increasing mantissa bit-width for more than 33 would not increase the accuracy significantly. It is because ϵ_{total} is dominated by ϵ_{int} but not ϵ_{fin} after m_w reached 33. Therefore, using more than 33 bits of mantissa is consuming unnecessary resources.

Fig. 3 shows the contour plot of $\epsilon_{rms}(m_w, d_f)$ at different error levels for the barrier option pricer and provides an overview of the total error using different m_w and d_f combinations.

3.2 Performance Modeling

The performance of the system is defined with the following equation:

$$\phi_{int}(m_w, d_f) = \frac{\phi_{pt}(m_w)}{N_{pt}(d_f)} \tag{3}$$

ϕ_{int} is the throughput in aggregated integrations per second per FPGA, ϕ_{pt} is the throughput in aggregated number of integration points per second per FPGA and N_{pt} is the number of integration points per integration. Furthermore, we define p_L as the degree of parallelism (number of replicated cores) and $freq$ as the clock frequency of the FPGA. With multiple replicated and fully pipelined integration cores running in parallel, ϕ_{pt} is defined as:

$$\phi_{pt}(m_w) = p_L(m_w) \cdot freq \tag{4}$$

because each core can process one integration point per clock cycle. ϕ_{pt} and p_L is monotonically decreasing with m_w. A higher m_w leads to a larger core, so fewer cores will fit in the FPGA, reducing degree of parallelism p_L and lower aggregated integration points throughput ϕ_{pt}. ϕ_{pt} is also monotonically decreasing with d_f, as a higher d_f leads to more integration points per integration N_{pt} and fewer integrations could be computed per second. Therefore, we have the following inequalities:

$$\phi_{int}(m_{w_x}, d_f) \geq \phi_{int}(m_{w_y}, d_f), \forall m_{w_x} < m_{w_y} \tag{5}$$

$$\phi_{int}(m_w, d_{f_x}) \geq \phi_{int}(m_w, d_{f_y}), \forall d_{f_x} < d_{f_y} \tag{6}$$

Fig. 4 shows the 3D graph of aggregated FPGA throughput $\phi_{int}(m_w, d_f)$ of the barrier option pricer which is consistent with the above inequalities.

3.3 Optimisation Objective Equation

Our objective is to determine the set of (m_w, d_f) which produces the design with optimal performance while maintaining the same level of accuracy. We define $\boldsymbol{\epsilon_{tol}}$ as error tolerance level. With the results from Equation 3 and 4, the following 2-dimensional optimisation problem can be formulated:

$$\max_{m_w, d_f} \left(\frac{p_L(m_w) \cdot freq}{N_{pt}(d_f)} \right), m_w \in \mathbb{Z}^+, d_f \in \mathbb{R}^+, \epsilon_{rms}(m_w, d_f) < \epsilon_{tol} \tag{7}$$

For example, Fig. 5 and Fig. 6 show the 3D plots of the optimisation result of barrier option pricer at $\epsilon_{tol} = 10^{-4}$ and $\epsilon_{tol} = 10^{-3}$ respectively by using the result of Fig. 3 and Fig. 4. We can see from the figures that the optimal aggregated throughputs are 350 and 1078 integrations per second. The corresponding (m_w, d_f) sets are (31,9.8) and (26,5.8).

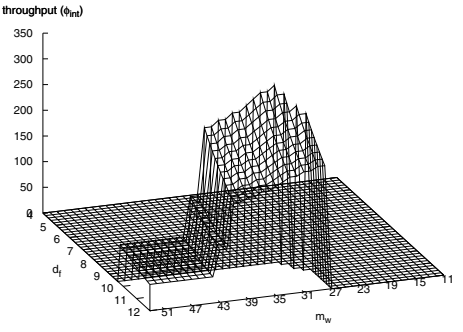

Fig. 5. The aggregated FPGA throughput satisfying $\epsilon_{rms}(m_w, d_f) < 10^{-4}$

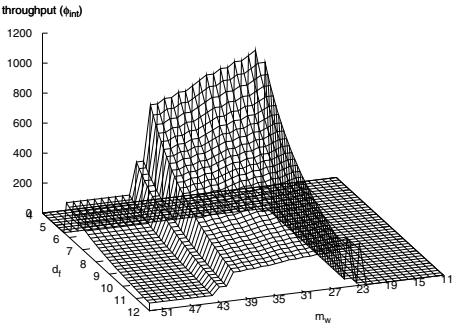

Fig. 6. The aggregated FPGA throughput satisfying $\epsilon_{rms}(m_w, d_f) < 10^{-3}$

4 Optimisation Algorithm and Methodology

This section provides the algorithms and a systematic way to apply the precision optimisation technique for a quadrature problem. The optimisation algorithm uses the property that the throughput of the integration decreases monotonically with respect to both m_w and d_f as shown in inequalities (5) and (6). The optimal throughput will only occur at the Pareto frontier points (Pareto set **S**) of (m_w, d_f) satisfying $\epsilon_{rms} < \epsilon_{tol}$ and, therefore, it is not necessary to obtain the ϵ_{rms} values for all (m_w, d_f) combinations. Fig. 7 shows the Pareto frontier of a barrier option pricer and the corresponding throughput as an illustration.

The detailed steps of our proposed one-pass optimisation process are:

1. Prepare a set of sample inputs.
2. Evaluate the results of the sample inputs as reference values.
3. Apply Algorithm 1 to obtain a Pareto set **S**.
4. Apply Algorithm 2 on **S** to obtain the optimal ϕ_{int}.

In step 2, we will typically use double precision (m_w=53) and a sufficiently large d_f to obtain the reference values such that the reference values are known to be accurate. In step 4, the algorithm requires the values of function $N_{pt}(d_f)$ and $pL(m_w)$ in order to compute ϕ_{int}. The function $N_{pt}(d_f)$ could easily be determined with the knowledge of the integration problem. The parameters pL can be either obtained directly after the full FPGA implementation, or estimated using the resource usage of a single-core FPGA design. Fig. 8 shows our estimation of pL and the resource usage of a single-core barrier option pricer.

The whole optimisation process is completely automated in our case studies by designing the hardware implementation as a parametric template, with m_w and pL as parameters. The hardware implementations for different values of m_w are generated, placed and routed automatically from the template for the use of step 3 and 4.

Algorithm 1. Algorithm for obtaining Pareto set **S**

```
S ← ∅
for m_w ∈ m_w^min..m_w^max do
    perform binary search for min d_f s.t ε_rms(m_w, d_f) < ε_tol
    if found then
        add tuple (m_w, d_f) to S
    end if
end for
```

Algorithm 2. Algorithm for determining the optimal precision and density factor

```
φ^max ← 0
for (m_w, d_f) ∈ S do
    if φ_int(m_w, d_f) > φ^max then
        φ^max ← φ_int(m_w, d_f)
        S_optimal ← (m_w, d_f)
    end if
end for
```

5 Case Studies

The hardware architectures of two financial applications and one benchmark integration problem are designed. Simpson's rule is used in all three case studies. The optimal combination of (m_w, d_f) is determined using the optimisation methodology and algorithms as described in the previous two sections with $\epsilon_{tol} = 10^{-3}$. As the range of the floating-point numbers is known to be small, the exponent size of the floating-point operators is set to 8. The accumulation is performed in double precision ($m_w = 53$) to minimise the loss of accuracy due to insufficient dynamic range in the accumulator.

5.1 Discrete Moving Barrier Option Pricer

The first case study is the pricing of discrete barrier options, which is a real-world pricing problem for which there is no closed-form solution. The pricing equation of an barrier option using quadrature methods is derived from the Black and Scholes partial differential equation [2]. For an option with an underlying asset following geometric Brownian motion:

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + (r - D_c)S\frac{\partial V}{\partial S} - rV = 0 \tag{8}$$

where $V(S,t)$ is the price of the option, S is the value of the underlying asset, t is time, r is risk-free interest rate, σ is volatility of the underlying asset, K is exercise price, and D_c is continuous dividend yield.

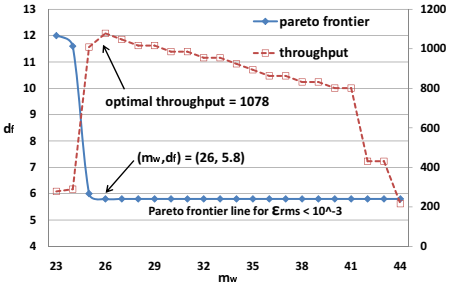

Fig. 7. The Pareto frontier line of barrier option pricer when $\epsilon_{tol} = 10^{-3}$

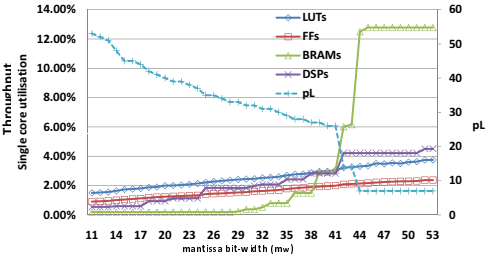

Fig. 8. pL estimation and the single core resource utilisaion of barrier option pricer

The following standard transformations

$$x = \log(S_t/K), \qquad y = \log(S_{t+\Delta t}/K)$$

give us the solution of $V(x,t)$ as:

$$V(x,t) = A(x) \int_{-\infty}^{+\infty} E(x,y)V(y,t+\Delta t)dy \tag{9}$$

where

$$A(x) = \frac{1}{\sqrt{2\sigma^2\pi\Delta t}} e^{(-kx/2)-(\sigma^2k^2\Delta t/8)-r\Delta t} \tag{10}$$

$$E(x,y) = e^{(yC_2-(x-y)^2C_1)}, \quad C_1 = \frac{1}{2\sigma^2\Delta t}, \quad C_2 = \frac{2(r-D_c)-\sigma^2}{2\sigma^2} \tag{11}$$

Since C_1 and C_2 will not change during the whole pricing process, they can be precomputed in software. Eq. (9) is the basic building block for quadrature option pricing. To price a down-and-out discrete moving barrier option with m time steps and B_m as the barrier price at time step m, we define the transformed position of b_m as:

$$b_m = log(B_m/K), \tag{12}$$

then the option price V_m at time step m can be computed using the equation:

$$V_m(x,t_m) \approx A(x) \int_{y_{min_m}}^{y_{max_m}} E(x,y)V_{m+1}(y,t_{m+1})dy, \tag{13}$$

where

$$y_{max_m} = x + 10\sigma\sqrt{t_{m+1}-t_m} \tag{14}$$

$$y_{min_m} = \max(b_m, x - 10\sigma\sqrt{t_{m+1}-t_m}) \tag{15}$$

The barrier option value is calculated iteratively backward from the expiry date to the present date as shown in Fig. 9. The main data-path for the hardware barrier core is shown in Fig. 10.

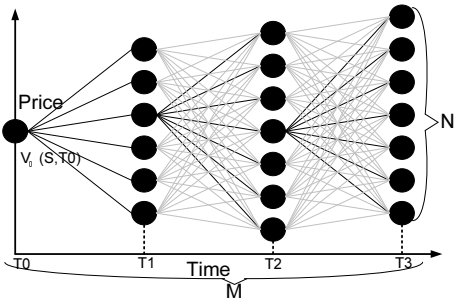

Fig. 9. The backward barrier option iteration process

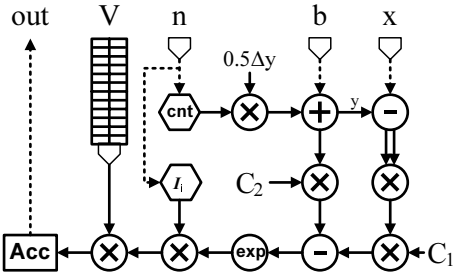

Fig. 10. The hardware barrier option pricing core

As the change of price exhibits a Brownian motion, the value of y fluctuates proportional to $\sqrt{\Delta t}$. Therefore, the size of δy should also be defined proportional to $\sqrt{\Delta t}$. We define the grid density factor d_f as $\frac{\sqrt{\Delta t}}{\delta y}$. Using the methods from Section 4, the optimal (m_w, d_f) is found to be (26,5.8).

5.2 Multi-dimensional European Option pricer

The option pricing equation with multiple underlying assets using quadrature methods is based on the multi-asset version of Black and Scholes partial differential equation [1]:

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sum_{i=1}^{d}\sum_{j=1}^{d}\sigma_i\sigma_j\rho_{ij}S_iS_j\frac{\partial^2 V}{\partial S_i \partial S_j} + \sum_{i=1}^{d}(r - D_i)S_i\frac{\partial V}{\partial S_i} - rV = 0 \qquad (16)$$

where r is the risk-free interest rate, d is the number of underlying assets, S_i are the underlying asset values, σ_i and D_i are the corresponding volatilities and dividend yields, and ρ_{ij} is the correlation coefficient between underlying asset values S_i and S_j. Note that $|\rho_{ij}| \leq 1$, $\rho_{ii} = 1$ and $\rho_{ij} = \rho_{ji}$. We make the logarithmic transformations

$$x_i = \log(S_i), \quad y_i = \log(S_i)$$

to be the chosen nodes at t and $t + \Delta t$. Let R be the matrix such that element $R(i,j) = \rho_{ij}$.The solution is:

$$V(x_1, \ldots, x_d, t) = C\int_{-\infty}^{+\infty}\ldots\int_{-\infty}^{+\infty} V(y_1, \ldots, y_d, t + \Delta t) \\ E(x_1, \ldots, x_d, y_i, \ldots, y_d)dy_1 \ldots dy_d \qquad (17)$$

$$C = e^{-r\Delta t}(2\pi\Delta t)^{-n/2}(|R|)^{-1/2}(\sigma_1\sigma_2\ldots\sigma_d)^{-1}, \qquad (18)$$

$$E(x_1, \ldots, x_d, y_i, \ldots, y_d) = exp(-\frac{1}{2}\alpha^T R^{-1}\alpha), \tag{19}$$

$$\alpha_i = \frac{x_i - y_i + (r - D_i - \frac{\sigma_i^2}{2})\Delta t}{\sigma_i(\Delta t)^{1/2}} \tag{20}$$

We define the grid density factor d_f as $\frac{\sqrt{\Delta t}}{\delta y}$ such that it is inversely proportional to the grid spacing. The optimal (m_w, d_f) is found to be (20,23).

5.3 Genz's "Discontinuous" Benchmark Integral

Our last case study is Genz's "Discontinuous" benchmark multi-dimensional integral(21). It is a common test integral being used in evaluation of different numerical integration methods. In our tests we use $n = 4$ as the dimension and an integration domain of $[0, 1)^4$. Fully parallelised designs are used in our FPGA implementations and the data-paths can compute a single sample point per clock cycle, with constants c_i and w_i:

$$I = \int\int\cdots\int f_{dis}(x_1, x_2, \cdots x_n)dx_1dx_2\cdots dx_n \tag{21}$$

$$f_{dis} = \begin{cases} 0 & \text{if } x_0 > w_0 \text{ or } x_1 > w_1 \\ exp(\sum_{i=1}^{n}(c_i \times x_i)) & \text{otherwise} \end{cases} \tag{22}$$

In this problem, we define $d_f = N$ since its grid density should depends on the number of grid points only. The optimal (m_w, d_f) is found to be (11,96).

6 Result and Evaluation

We use the MaxWorkstation reconfigurable accelerator system from Maxeler Technologies for our evaluation. It has a MAX3424A card with a Xilinx Virtex-6 SX475T (XC6VSX475T) FPGA. The XC6VSX475T FPGA has a total of 297,600 LUTs, 595,200 FFs, 1,064 DSPs and 2,016 BRAMs. We set the target clock frequency at 100MHz ($freq$). The card is connected to an Intel i7-870 CPU through a PCI express link with a measured bandwidth of 2 GB/s. The Intel CPU has 4 physical cores.

The Intel Compiler (ICC) is used in our software implementations with optimisation flag `-fast` and SSE4.2 enabled. The software implementation is manually optimised in order to achieve the maximum throughput. Multiple processes are launched simultaneously in order to utilise all 4 physical cores of the quad-core i7-870 CPU.

For the FPGA implementations, we use the MaxCompiler as our development system, which adopts a streaming programming model similar to [15] and supports customisable data formats so that floating-point calculations can be performed with different mantissa bit-widths. The hardware implementations are synthesized, placed and routed using Xilinx 13.1 ISE.

Table 1. Comparison of different applications using i7-870 quad-core CPU, NVIDIA Tesla C2070 GPU, double precision XC6VSX475T FPGA and reduced precision optimised XC6VSX475T FPGA

	Discrete barrier option			3D European option				Genz's benchmark		
	CPU	FPGA		CPU	GPU	FPGA		CPU	FPGA	
arithmetic	double	double	optimised	double	double	double	optimised	double	double	optimised
clk freq. (GHz)	2.93	0.1	0.1	2.93	1.15	0.1	0.1	2.93	0.1	0.1
num. of cores	4	7	35	4	448	5	18	4	6	36
exec. time (sec.)	313	86.3	22.3	145	11.45	34.5	9.6	328.56	169.98	28.33
norm. speedup	1x	3.6x	**14.0x**	1x	12.7x	4.2x	**15.1x**	1x	1.9x	**11.6x**
opti. gain	-	1x	**3.9x**	-	-	1x	**3.6x**	-	1x	**6.0x**
APCC(W) [1,2]	89	13	16	69	117	4	5	81	4	4
AECC(J) [3]	27857.0	1121.9	356.8	10005.0	2026.7	138.1	48.0	26613.4	679.9	113.3
norm. energy	**78.1x**	3.1x	1x	**208.4x**	**42.2x**	2.9x	1x	**234.9x**	**6x**	1x

[1] APCC = run-time power consumption - idle power consumption.
[2] The idle power is $80W$ for FPGA and CPU system, and $154W$ for GPU system.
[3] AECC = APCC $\times$ execution time.
[4] In all applications, $\epsilon_{tol} = 10^{-3}$.

For the GPU performance result, we use NVIDIA Tesla C2070 GPU to measure the performance of our 3-dimensional European option pricer. The GPU has 448 cores running at 1.15 GHz and has a peak double precision performance of 515 GFlops.

The experiments are performed to compute a portfolio of 100 barrier options, a portfolio of 576 3D-European options, and a set of 1120 Genz's benchmark integrals.

6.1 Performance Comparison

Table 1 shows comparisons of the implementations running on a CPU with double precision arithmetic, an FPGA with double precision arithmetic, and an FPGA with optimised precision using our proposed methodology. The GPU result of the multi-dimensional European option pricer is also presented. The computed results of all designs are all optimised for $\epsilon_{tol} = 10^{-3}$ and have the same accuracy level. The measured execution time includes the data transfer time, which means the speedup figures are measured end-to-end.

Using the reduced precision optimisation techniques with XC6VSX475T FPGA, we achieve 3.6 to 6.0 times speedup gain over the original double precision FPGA designs. These optimised FPGA designs running on XC6VSX475T are 11.6 to 15.1 times faster than multi-threaded software designs running on a **quad-core** Intel i7-870, and 1.2 times faster than a GPU design running on a Tesla C2070.

6.2 Energy Comparison

We also compare the energy efficiency of the three applications on different devices. The average power consumption is measured using a remote power measuring socket from Oslon® electronics with an measuring interval of 1 second. Additional power consumption for computation (APCC) is defined as the power usage during the computation time (run-time power) minus the power usage at

idle time (static power). In other words, APCC is the dynamic power consumption for that particular computation. Since the dynamic power consumption fluctuates a little, we take the average value of dynamic power to be the APCC. The additional energy consumption for computation (AECC) is defined by the following equation:

$$AECC = APCC \times \textit{Total Computational Time}. \tag{23}$$

Therefore, AECC measures the actual additional energy consumed for that particular computation.

As shown in Table 1, the precision optimised FPGA designs demonstrate the greatest energy efficiency over both CPU and GPU. It is 78.1 - 234.9 times more energy efficient than an Intel i7-870 quad-core CPU, and 42.2 times more energy efficient than a Tesla C2070 GPU.

7 Conclusion

We presented a precision optimisation methodology for the generic quadrature method using reconfigurable hardware. Our novel methodology optimises the performance by considering both integration grid density and mantissa bit-width of the floating-point operators. Increasing the integration grid density reduces integration error but increases the required amount of computation, while increasing the mantissa bit-width improves precision but decreases the computation speed, due to reduced parallelism. Our proposed algorithm allows us to identify the optimal balance between the number of integration points and the precision of the floating-point operator, such that the throughput is maximised while the accuracy remains in a given error tolerance level.

Our three case studies demonstrate that using our proposed optimisation methodology, the reduced precision FPGA designs are up to 6 times faster than comparable FPGA designs with double precision arithmetic. They are up to 15.1 times faster and 234.9 times more energy efficient than an i7-870 quad-core CPU, and are 1.2 times faster and 42.2 times more energy efficient than a Tesla C2070 GPU.

Current and future work includes exploring the adaptive quadrature method on reconfigurable hardware [18]. We are also interested in extending the mantissa bit-width optimisation technique for other numerical methods.

Acknowledgment. The support of the Croucher Foundation, the UK Engineering and Physical Sciences Research Council, Maxeler, and Xilinx is gratefully acknowledged. The research leading to these results has received funding from the European Union Seventh Framework Programme under grant agreement number 248976 and 257906.

References

1. Andricopoulos, A.D., Widdicks, M., Newton, D.P., Duck, P.W.: Extending quadrature methods to value multi-asset and complex path dependent options. Journal of Financial Economics 83(2), 471–499 (2007)

2. Black, F., Scholes, M.S.: The pricing of options and corporate liabilities. Journal of Political Economy 81(3), 637–654 (1973)
3. Boland, D., Constantinides, G.: Automated precision analysis: A polynomial algebraic approach. In: Proc. IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM), pp. 157–164 (2010)
4. Chow, G., Kwok, K., Luk, W., Leong, P.: Mixed precision processing in reconfigurable systems. In: Proc. IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM), pp. 17–24 (May 2011)
5. Davis, M.H.A., Esparragoza-Rodriguez, J.C.: Large portfolio credit risk modeling. International Journal of Theoretical and Applied Finance 10(04), 653–678 (2007)
6. Fang, C.F., Rutenbar, R.A., Chen, T.: Fast, accurate static analysis for fixed-point finite-precision effects in DSP designs. In: IEEE/ACM international Conference on Computer-Aided Design, pp. 275–282 (2003)
7. Gaffar, A.A., Mencer, O., Luk, W., Cheung, P.Y.K.: Unifying bit-width optimisation for fixed-point and floating-point designs. In: FCCM, pp. 79–88 (2004)
8. Humphries, T., Celler, A., Trammer, M.: Improved numerical integration for analytical photon distribution calculation in spect. In: IEEE Symposium Conference on Nuclear Science, vol. 5, pp. 3548–3554 (2007)
9. Kinsman, A., Nicolici, N.: Finite precision bit-width allocation using SAT-Modulo theory. In: Proc. Design Automation and Test in Europe (DATE), pp. 1106–1111 (2009)
10. Kum, K.I., Sung, W.: Combined word-length optimization and high-level synthesis of digital signal processing systems, vol. 20(8), pp. 921–930 (2001)
11. Lee, A., Yau, C., Giles, M.B., Doucet, A., Holmes, C.C.: On the utility of graphics cards to perform massively parallel simulation of advanced Monte Carlo methods. Journal of Computational and Graphical Statistics, 769–789 (2010)
12. Lee, D.U., Gaffar, A.A., Cheung, R.C.C., Mencer, O., Luk, W., Constantinides, G.A.: Accuracy-guaranteed bit-width optimization. IEEE Trans. on CAD of Integrated Circuits and Systems 25(10), 1990–2000 (2006)
13. Lee, D.U., Gaffar, A.A., Mencer, O., Luk, W.: Minibit: bit-width optimization via affine arithmetic. In: DAC, pp. 837–840 (2005)
14. Masserey, A., Rappaz, J., Rozsnyo, R., Swierkosz, M.: Numerical integration of the three-dimensional green kernel for an electromagnetic problem. Journal of Computational Physics 205(1), 48–71 (2005)
15. Mencer, O.: ASC: a stream compiler for computing with FPGAs, vol. 25(9), pp. 1603–1617 (2006)
16. Osborne, W., Coutinho, J., Cheung, R., Luk, W., Mencer, O.: Instrumented multistage word-length optimization. In: Proc. International Conference on Field Programmable Technology (FPT), pp. 89–96 (2007)
17. Osborne, W.G., Cheung, R.C.C., Coutinho, J.G.F., Luk, W., Mencer, O.: Automatic accuracy-guaranteed bit-width optimization for fixed and floating-point systems. In: Proc. International Conference on Field Programmable Logic and Applications (FPL), pp. 617–620 (2007)
18. Rice, J.R.: A metalgorithm for adaptive quadrature. Journal of the ACM 22, 61–82 (1975)
19. Sueli, E., Mayers, D.F.: An Introduction to Numerical Analysis. Cambridge University Press (2006)
20. Tse, A.H.T., Thomas, D., Luk, W.: Design exploration of quadrature methods in option pricing. IEEE Transactions on Very Large Scale Integration (VLSI) Systems (2011) (accepted for publication)