EXPLORING RECONFIGURABLE ARCHITECTURES FOR EXPLICIT FINITE DIFFERENCE OPTION PRICING MODELS

Qiwei Jin, David B. Thomas and Wayne Luk

Department of Computing
Imperial College London
United Kingdom
email:qj04, dt10, wl@doc.ic.ac.uk

ABSTRACT

This paper explores the application of reconfigurable hardware and Graphics Processing Units (GPUs) to the acceleration of financial computation using the finite difference (FD) method. A parallel pipelined architecture has been developed to support concurrent valuation of independent options with high pricing throughput. Our FPGA implementation running at 106MHz on an xc4vlx160 device demonstrates a speed up of 12 times over a Pentium 4 processor at 3.6GHz in single-precision arithmetic; while the FPGA is 3.6 times slower than a Tesla C1060 240-Core GPU at 1.3GHz, it is 9 times more energy efficient.

1. INTRODUCTION

The finite difference (FD) method [1] is a basic numerical procedure used for financial option pricing, especially in situations when a closed-form solution does not exist (e.g. Barrier Options [2]). In particular, it can be used to handle certain types of options, such as American Options, that can not be easily modelled by Monte-Carlo methods.

The FD method solves Partial Differential Equations (P-DEs), by discretising the underlying variables in the PDE. Its application can be found in many scientific domains. For example the Laplace (Heat) Equation in thermodynamics [3]; Maxwell's (Wave) Equation in electromagnetism [4]; and the Black-Scholes equation in finance [1].

Option pricing using the FD method can typically be performed in milliseconds on a modern general purpose processor. However, if a high degree of accuracy is required, the amount of computation needed increases quadratically in the simplest case where there is only one random variable involved in the PDE. As financial derivatives become more sophisticated and complex, it is common to see two or more underlying random variables being used in a pricing model, which requires computational power to grow exponentially. In the finance industry the FD method is used in generating risks for large portfolios. Such generation can take many hours even on a large computer grid. Our aim is to explore acceleration technologies which can significantly reduce both execution time and grid size, without affecting solution quality. This paper shows how Field Programmable Gate Arrays (FPGAs) and Graphics Processing Units (GPUs) can provide viable methods of accelerating the FD methods to solve the standard Black-Scholes PDE. We mainly discuss designs based on FD method for European option valuation, but extensions to support American option valuation are straightforward. Our design is flexible enough to solve other types of PDEs such as the Heat Equation. The main contributions are the following:

- two pipelined architectures based on explicit FD option pricing model, that are capable of processing concurrent requests for option valuations (Section 4);
- implementation of the two architectures in reconfigurable hardware, exploiting on-chip resources by allowing parallelism in different dimensions (Section 5);
- evaluation and comparison of the reconfigurable approach and alternative implementations based on two nVidia GPUs and a general-purpose Intel processor (Section 6).

2. MOTIVATION

Hardware acceleration of financial instruments pricing using Monte Carlo methods has been studied intensively in the past few years. For instance, a platform independent domain specific language has been invented to produce optimised pipelined designs with thread level parallelism for Monte Carlo simulations from a high level abstraction [5]; an FPGA-based stream accelerator with higher performance than GPUs and Cell processors has been proposed for evaluating European options [6]; and an architecture with a pipelined datapath and an on-chip instruction processor has been reported for speeding up the Brace, Gatarek and Musiela (BGM) interest rate model for pricing derivatives [7].

Some recent studies have focused on pipelined tree based methods [8] and quadrature methods [9]. Tree based methods are efficient, and can handle certain types of options such as American options that cannot be handled easily by

Application	Implementation
The Heat Equation [10]	24 bit Fixed Point
Maxwell's Equation [11]	16 bit Fixed Point
Poisson Problem (Iterative Refinement) [12]	Mixed 32/64 bit Floating Point

Table 1: Existing PDE solvers

Monte Carlo methods, while quadrature methods provide more accurate result over tree based methods. However, the above methods cannot effectively address issues such as the effects of asset price on option price over time.

The FD method, on the other hand, can generate a grid of option prices over time, based on a range of underlying asset price variations, which can then be easily plotted on a graph for further study if necessary.

There is also existing research explores the acceleration of FD and iterative refinement methods for solving PDEs, listed in Table 1. However, the efficiency and applicability of these methods is highly dependent on application domain, so in this paper we develop techniques specifically for the option pricing domain. On the other hand, our design is flexible enough to be generalised to solve PDEs such as the Heat Equation, with minor parameter settings.

The FD method solves the Black-Scholes equation by discretising time and the underlying asset price. We now briefly describe the concept of option pricing, in terms of European options, followed by American options.

A *put option* in general is a contract that gives party A the right to sell some asset S to party B at the strike price K. The important observation is that the option provides a right, not an obligation: party A can choose whether or not to exercise that right (i.e. to sell asset S at price K). In general the put option will only be exercised if $K > S_t$ where S_t is the price of S at time t.

A *European* option can only be exercised at a particular time T. The price of a European option is determined by the formula $max(K-S_T, 0)$. In contrast, an *American* option is one where party A can exercise the option at any time up until the option expires at time T. The American option is very common, but it presents some difficulties in pricing due to the freedom to exercise the option before the expiry date – in particular it becomes difficult to determine the option price using Monte-Carlo methods [5] due to its path dependence. In contrast, FD techniques are able to accurately price both European and American options.

3. THE FINITE DIFFERENCE MODEL

The FD model solves the Black Scholes PDE by discretising both time and the price of the underlying asset S, and mapping both onto a two-dimensional grid. There are three kinds of FD methods: implicit, explicit and Crank-Nicolson. We consider the explicit mechanism, as it is the most computationally efficient method amongst all three (other two involve solving linear equations at each iteration) [1].

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

$$\frac{\partial f}{\partial t} + (r-q)S\frac{\partial f}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 f}{\partial S^2} = rf \qquad (1)$$

where $f(S,t)$ denotes the price of the option, S denotes the value of the underlying asset, t denotes a particular time, r denotes risk-free interest rate, σ denotes the volatility of the underlying asset, and q denotes the dividend yield paid by the underlying asset.

Suppose the time to maturity for the option is T. We discretise T by dividing it into N equally spaced intervals, namely: $\Delta t = T/N$. As a result, a total of $N+1$ points in time are considered:

$$0,\ \Delta t,\ 2\Delta t,\ ...,\ (N-1)\Delta t,\ T$$

We then determine an asset price S_{max} such that $f(S,T) = 0$, and discretise it into M equally spaced intervals. As a result, $M+1$ underlying asset prices are obtained:

$$0,\ \Delta S,\ 2\Delta S,\ ...,\ (M-1)\Delta S,\ S_{max}$$

where $\Delta S = S/M$. A $(N+1)$ by $(M+1)$ grid is defined by the time and asset price points.

An efficient approximation for computation within this grid can be obtained by a change of variable [1], discretising over $\ln S$ instead of S. Suppose $Z = \ln S$, the following equations can be obtained:

$$\begin{aligned} f_{i,j} &= \alpha_j f_{i+1,j-1} + \beta_j f_{i+1,j} + \gamma_j f_{i+1,j+1} \qquad (2)\\ \alpha_j &= \frac{1}{1+r\Delta t}(-\frac{\Delta t}{2\Delta Z}(r-q-\sigma^2/2) + \frac{\Delta t}{2\Delta Z^2}\sigma^2)\\ \beta_j &= \frac{1}{1+r\Delta t}(1 - \frac{\Delta t}{\Delta Z^2}\sigma^2)\\ \gamma_j &= \frac{1}{1+r\Delta t}(\frac{\Delta t}{2\Delta Z}(r-q-\sigma^2/2) + \frac{\Delta t}{2\Delta Z^2}\sigma^2) \end{aligned}$$

where α_j, β_j and γ_j are independent of j and only need to be calculated once. Equation 2 can be implemented efficiently by two nested for-loops iterating round a one dimensional array: with an outer loop stepping t backwards from T to 0, and an inner loop calculating the price for each $f_{t,j}$ at level t in the grid. The array holds the intermediate values, and can be updated in place.

4. PARALLEL ARCHITECTURE EXPLORATION

Two levels of parallelism can be exploited for the FD model: coarse grained parallelism and fine grained parallelism. We define coarse granularity to be the ability to valuate different options at the same time; fine granularity to be the ability to valuate different nodes in a grid at the same time. The higher the coarse granularity, the more options can be priced

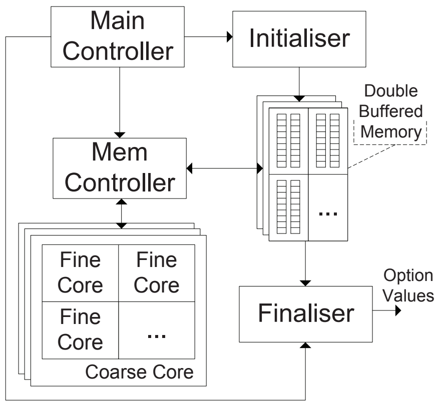

Fig. 1: System architecture for computing the FD model.

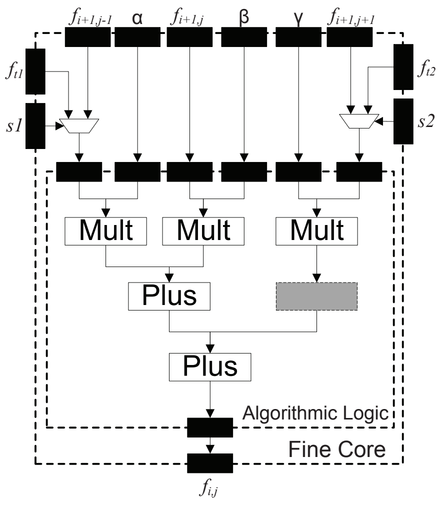

Fig. 2: Binomial Model: hardware design for the block *Fine Core* in Fig. 1. The solid black boxes denote registers and the grey boxes denote pipeline balancing registers that are allocated automatically by *HyperStreams* (Section 5).

at the same time; the higher the fine granularity, the faster the valuation speed per option.

As the amount of computational resources are limited, trade-offs between the coarse and fine granularities need to be made depending on the user requirements. Our design allows the user to choose the most appropriate granularity and configure the device on the fly.

Figure 1 demonstrates our proposed architecture for explicit finite difference option valuation. The architecture is comprised of the following components: (a) the Main Controller, (b) the Coarse/Fine Core, (c) the Memory Module, (d) the Memory Controller and (e) the Initialiser and Finaliser. The Main Controller controls the overall process and communicates to the software-end. The Coarse Core is the main processor. We assign one option to one Coarse Core. There can be many Coarse Cores in our architecture and each Coarse Core consists of one or more Fine Cores. The Fine Core is the basic computational unit. It is a fully pipelined block which takes three previously calculated option values and calculates the value of the present node. A Fine Core alone can form a one-core Coarse Core. Many Fine Cores can be wired up to assemble a more powerful Coarse Core. The Memory Module adopts double buffering to fully utilise the pipeline. FPGA embedded memory (in our case Xilinx Block Select RAMs) is used to implement the Memory Module. The Memory Controller couples the Memory Modules and Coarse Cores, makes sure that data are retrieved and stored correctly. It is capable of providing one set of parameters to a Fine Core per clock cycle. The Initialiser initialises the memory module by setting up the initial option prices, and the Finaliser finalises the process by getting data ready to be sent back to the software-end.

We adopt the C-Slow methodology [13] in our design. It allows us to use high-latency pipelined functional units to achieve high clock rates while still achieving high throughput. We continuously provide parameters into the Fine Core pipeline to valuate different nodes, and we load parameters for another option while we are waiting for the results required for the next iteration of the current option. The typical grid size used in industry is 6K by 30K, where 6K calculations are independent of each other at each iteration. As the total pipeline latency is usually tens of clock cycles depending on the actual device, two options are required to be valuated at the same time.

The C-Slow approach allows parallelism in time, but we also try to exploit parallelism in space for the FD model. There are data dependencies between any two consecutive columns in the grid to be processed. However the data in the same column is independent of each other hence parallelisable. In the simplest case we arrange the original grid horizontally into 2 parts of the same size and process the upper half and lower half of the grid concurrently. In our design the array being iterated over is divided into two separate arrays and processed simultaneously by two Fine Cores. However the two array elements (the overlapped elements) adjacent to the dividing point need to be included in both of the arrays. At the end of each iteration the two overlapped elements are swapped in order to seal the cut. In cases when the grid has odd number of rows (e.g. it can not be cut into two equally sized parts), we can adjust the number of the overlapped elements in the array with fewer elements to rebalance the computational load.

Figure 2 shows our hardware design of the Fine Core, which essentially corresponds to Equation 2. For each $f_{i,j}$ it valuates, it takes a set of parameters provided by the Memory Controller: three previously valuated grid node values and three parameters associated to them. Note that the parameters remain the same throughout the option valuation process. Switches s1 and s2 are used to swap the overlapped elements with peer Fine Cores, the Memory Controller is designed to make sure the values are only swapped at the right time. In cases when there is only one Fine Core in a

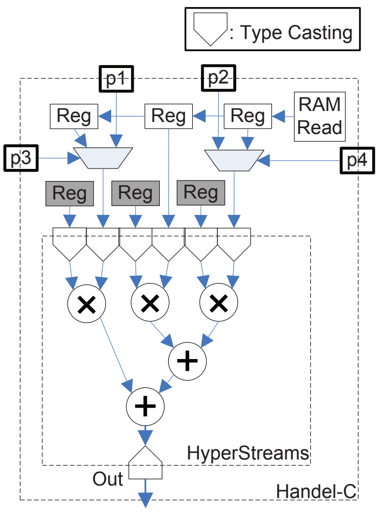

Fig. 3: The data flow of the hardware implementation on FPGA. Noting the registers in grey hold constant parameters for an option; the bold squares are ports connecting to peer Fine Cores to swap overlapped elements.

Coarse Core, these switches are not used.

Our implementation can be extended easily to support American option pricing by adding an extra comparison operator in the pipeline, as the price of the American option at a particular point in the grid can be obtained by a simple table lookup from an $M + 1$ element lookup table (given that the lookup table of possible option prices has been generated initially). The implementation can also be used to solve one-dimensional Heat Equation: the array need to be initialised with temperature values, and parameters α_j, β_j and γ_j will be set according to the Heat PDE.

So far the Fine Core is the most resource intensive basic component in our design. In the asymptotic case we would expect the overall performance to be dominated by the size and speed of this block, as other components consist of memories and a small amount of selection logic. We shall examine our implementation in more detail in the next section.

5. IMPLEMENTATION

Our FPGA implementation of the Fine Core logic for the FD option pricing model is based on *HyperStreams* and the *Handel-C* programming language. The GPU implementation is based on CUDA version 2.0 API provided by nVidia [14].

First we consider FPGA implementations. *HyperStreams* is a high-level abstraction based on the *Handel-C* language [6]. It supports automatic scheduling of pipelined operators at compile time to produce a fully-pipelined hardware implementation. This feature is useful when implementing complex algorithmic calculations in FPGAs.

Figure 3 shows a fully pipelined FPGA implementation of the Fine Core logic indicated in Equation 2. Each symbol shown in a "*HyperStreams*" block in Figure 3 refers to a *HyperStreams* operator (e.g. $\bigoplus$ stands for *HsAdd*). The inputs are cast to desired internal representation, for example *HS_DOUBLE*, at the top of the *HyperStreams* block. Once all the computations are finished, the output stream is cast back to the output format.

Fixed point arithmetic is best performed on FPGAs with well established optimisations such as word-length optimisation [15]. However in the FD case the numerical values can theoretically range from infinity to infinitely close to zero, hence we use floating point arithmetic instead. The control logic, which is used to send and retrieve data from pipelines, is written in the *Handel-C* language. To fully utilise the pipeline, double buffering is used to get around the FPGA memory port limitation, memory reads are pipelined so that there is only one read from the memory per clock cycle instead of three. The results are simultaneously written to another memory bank.

The ports p3 and p4 shown in Figure 3 are used to swap overlapped elements with peer Fine Cores. Once the option valuation is finished, the result is sent back to software via the *DSM* interface, which is a platform-independent library for hardware-software communication.

The tool flow is as follows: *Handel-C* source code is synthesised to EDIF using the Celoxica DK5 suite which supports *HyperStreams*. Xilinx ISE 10.1 is used to place and route the design. The target device on our Celoxica RCHTX platform is an xc4vlx160 FPGA from the Xilinx Virtex 4 family, the description is platform and architecture independent, so could be retargeted for any platform or device supported by DK.

We then consider the implementation on Graphics Processing Units (GPUs). GPUs are another alternative to CPUs for computational intensive tasks, and have also been used for financial computation [16]. Our implementation on GPUs is based on the CUDA 2.0 programming API provided by nVidia. Double-precision floating-point arithmetic is supported by CUDA 2.0 on supported GPU platforms.

To exploit the CUDA single instruction multiple thread (SIMT) model [14], we assign one option to one thread block and use double buffering with low latency on-chip shared memory to avoid frequent access to the GPU's high latency global memory.

6. RESULTS

In this section we study the performance of our implementations in FPGA and on GPU. The testing case is the common industrial level European option pricing problem based on 6K$\times$30K grids. We make sure that each implementation has sufficient amount of tasks to fully utilise its resource. For example, we price 70 options simultaneously on Tesla C1060 to make full use of its multiprocessors. The best performance of each implementation is recorded and shown in Table 2. For FPGA, we show the performance of the single

Coarse-Core implementation with multiple Fine Cores; for GPU we use 70 blocks with 256 threads per block. The performance of both 32-bit single-precision and 64-bit double-precision floating point implementations are considered.

All the FPGA and GPU implementations are compared to the software implementation on a reference Intel PC which is a 3.6GHz Pentium 4 processor with 4GB of RAM and Linux operating system. This implementation involves C++ code compiled with maximum speed optimisation options. The targeted FPGA device is a Virtex 4 xc4vlx160 on an RCHTX board. The targeted GPU devices are an nVidia Geforce 8600GT with 256MB of on-board RAM, and an nVidia Tesla C1060 with 4GB of on-board RAM. Table 2 shows a summary of our performance comparison results.

The top part of Table 2 shows the FPGA device utilisation figures. The results indicate that less than one third of the FPGA device is utilised in both cases involving double-precision and single-precision arithmetic. We can improve performance by replicating the valuation core in a single device. In our experiments replications over both the coarse and fine dimensions are considered. We find the peak performance occurs when we aggregate all the Fine Cores into one Coarse Core, as no logic is wasted to glue the Coarse Cores. Another reason is that the bandwidth for hardware-software communication is limited; fewer Coarse Cores means more bandwidth for each option, hence less communication overhead. In the actual implementation we allow the user to choose the Fine-Coarse granularity and configure the FPGA device on the fly.

The acceleration results can be found in the middle part of Table 2 for different precision implementations, including core replication that can be done on a single device to gain further performance. It can be seen that, the 32-bit single-precision FPGA implementation offers a 1.5 times acceleration over the software, while the 64-bit double precision version offers 1.2 times speedup. Eight single-precision Fine Cores can be replicated on the xc4vlx160, and is estimated to achieve 12.2 times acceleration.

Table 2 also shows the data for two GPUs. A speedup of 9.7 times and 43.9 times can be achieved by Geforce 8600GT and Tesla C1060 respectively, both under single precision. Regarding to the double-precision implementation, Tesla C1060 can outperform 26.6 times over the reference PC. Double precision is not supported by Geforce 8600GT.

It can be seen that the FPGA is 1.3 times faster than Geforce 8600GT, but 3.6 times slower than Tesla C1060 in the single-precision case. The result is not surprising as Tesla C1060 is based on the latest 65nm technology, while the Virtex 4 family is based on 90nm technology. To be fair, the Tesla series should be compared to the latest FPGA technology such as Virtex 5 from Xilinx and Stratix IV from Altera. If a larger Virtex 5 device is used with 2 times or more slices than that on our Xilinx xc4vlx160 device and with higher basic clock frequency, then potentially 2 times speedup can be achieved without further optimisation. Single-precision operators in FPGAs can run at a clock rate of up to 322MHz [17]; our current implementation at 106MHz has much scope for improvement.

From our experience there is a tradeoff when using *HyperStreams* between the development time and the acceleration achieved. Although we are able to implement complex algorithms easily in FPGAs with *HyperStreams*, the highest possible performance and utilisation of FPGA resources is not guaranteed. The balance between development time and performance needs to be explored with further research and experiment. However our *HyperStreams* implementations still give satisfactory result with significant acceleration over the software implementation. Hence *HyperStreams* is useful particularly for producing prototypes rap-idly to explore the design space; once promising architectures are found, further optimisations can be applied.

If energy consumption is taken into consideration, we find that the FPGA implementations are much more energy efficient than the GPU implementations. For instance, using Xilinx XPower Estimator, an 8-core design on an xc4vlx160 FPGA at 106MHz is 9 times more energy efficient than the two GPUs, based on single-precision arithmetic. If the energy efficiency of double-precision arithmetic is considered, the FPGA implementation outperforms the Tesla GPU and PC by 2.8 times and 46.2 times respectively. Note that both GPUs have similar energy efficiencies for this application, regardless of the fabrication technologies on which they are based. The GPU power consumption data come from [18] and [19].

7. CONCLUSION

This paper describes a new architecture for accelerating option pricing models based on FD method. The proposed design involves a highly pipelined datapath capable of processing multiple option valuations in parallel, which supports concurrent option pricing requests. We have implemented our design onto an xc4vlx160 FPGA, and demonstrate that our implementations can generally run more than 12.2 times faster than a Pentium 4 processor. They are 1.3 times faster than the GeForce 8600GT GPU in comparable technology, and are 3.6 times slower than the Tesla C1060 GPU. If energy consumption is taken into consideration, the FPGA is up to 9.4 times more energy efficient than a GPU in single-precision arithmetic, and 46 times more power efficient than a CPU in double-precision arithmetic.

Further work is planned to study the speed and area optimisations on hardware cores based on the latest Virtex 5 and Stratix IV FPGAs. FPGA-GPU collaboration is also on our agenda: we wish to find out the most efficient way to make FPGA and GPU work collaboratively. It would be worthwhile to investigate how FPGA and GPU techniques can be used in automating domain-specific strategies for producing designs which best meet user requirements in speed, area and energy consumption.

	FPGA		GPU			CPU
	Virtex 4 xc4vlx160		Geforce 8600GT	Tesla C1060		Intel Pentium 4
Number Format	single	double	single	single	double	double
Slices	5228 (7%)	8460 (12%)	-	-	-	-
FFs	4253 (3%)	6271 (4%)	-	-	-	-
LUTs	5780 (4%)	9891 (7%)	-	-	-	-
BRAMs	37 (12%)	69 (23%)	-	-	-	-
DSPs	12 (12%)	48 (50%)	-	-	-	-
Clock Rate	106MHz	82.9MHz	1.35 GHz	1.3GHz		3.6GHz
Processing Speed (M values/sec)	106	82.9	673	3057	1851	69.7
Replication (cores/chip)	8	3	32	240		1
Acceleration (1 core)	1.5×	1.2×	-	-	-	1×
Acceleration (replicated cores)	**12.0×**	**3.6×**	**9.7×**	**43.9×**	**26.6×**	**1×**
Max Power (Watt)	5.8	9.1	43	187		115
Energy Efficiency (M values/Joule)	**146**	**27.3**	**15.6**	**16.3**	**9.9**	**0.6**

Table 2: Finite difference Performance/area results for Xilinx xc4vlx160 FPGA, Geforce 8600GT and Tesla C1060 GPU and Intel Pentium 4 CPU; note that the percentage shows utilisation of a specific FPGA resource, and acceleration is compared with the Intel CPU.

Acknowledgement. The support of J.P. Morgan Securities Limited, particularly Abib Bocresion, Tony Chau, Eric Fournie and Mark Gibbons, is gratefully acknowledged.

8. REFERENCES

[1] J. Hull, *Options, Futures and Other Derivatives*, 6th ed. Prentice Hall, 2005.

[2] P. P. B. Y. Tian, "An explicit finite difference approach to the pricing of barrier options," *Applied Mathematical Finance*, vol. 5, no. 1, pp. 17–43, 1998.

[3] L. Elden, "Numerical solution of the sideways heat equation by difference approximation in time," *Inverse Problems*, vol. 11, no. 4, pp. 913–923, 1995.

[4] L. Gao, B. Zhang, and D. Liang, "The splitting finite-difference time-domain methods for Maxwell's equations in two dimensions," *J. Comput. Appl. Math.*, vol. 205, no. 1, pp. 207–230, 2007.

[5] D. Thomas, J. Bower, and W. Luk, "Automatic generation and optimisation of reconfigurable financial Monte-Carlo simulations," in *Proc. Int. Conf. on Application-Specific Systems, Architectures and Processors*. IEEE, 2007.

[6] G. Morris and M. Aubury, "Design space exploration of the European option benchmark using Hyperstreams," in *Proc. Int. Conf. on Field Programmable Logic and Applications*, 2007, pp. 5–10.

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

[8] Q. Jin, D. B. Thomas, W. Luk, and B. Cope, "Exploring reconfigurable architectures for binomial-tree pricing models," in *Proc. Int. workshop on Applied Reconfigurable Computing*, 2008, pp. 245–255.

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

[10] J. E. Vuillemin, P. Bertin, D. Roncin, M. Shand, H. H. Touati, and P. Boucard, "Programmable active memories: reconfigurable systems come of age," *IEEE Trans. on VLSI*, vol. 4, no. 1, pp. 56–69, 1996.

[11] E. Motuk, R. Woods, and S. Bilbao, "Implementation of finite difference schemes for the wave equation on FPGA," in *Proc. IEEE Int. Conf. on ASSP*, vol. 3, 2005, pp. 237–240.

[12] R. Strzodka and D. Göddeke, "Pipelined mixed precision algorithms on FPGAs for fast and accurate PDE solvers from low precision components," in *Proc. IEEE Symp. on Field-Programmable Custom Computing Machines*, 2006, pp. 259–268.

[13] N. Weaver, Y. Markovskiy, Y. Patel, and J. Wawrzynek, "Post-placement C-slow retiming for the Xilinx Virtex FPGA," in *Proc. ACM/SIGDA Int. Symp. on Field Programmable Gate Arrays*. ACM, 2003, pp. 185–194.

[14] NVIDIA, "nVidia CUDA programming guide," nVidia website, 2008.

[15] G. A. Constantinides, "Word-length optimization for differentiable nonlinear systems," *ACM Trans. on Design Automation of Elect. Sys.*, vol. 11, no. 1, pp. 26–43, 2006.

[16] M. Giles and X. Su, "Notes on using the nVidia 8800 GTX graphics card," 2007, Oxford University.

[17] Xilinx, "Floating-point operator v3.0 manual," Xilinx website, 2006.

[18] A. Vorobiev and A. B. A. S. Else, "nVidia GeForce 8600 GTS Spec," http://www.digit-life.com/, 2007.

[19] NVIDIA, "Tesla C1060 Datasheet," NVIDIA website, 2009.