Exploring Reconfigurable Architectures for Tree-Based Option Pricing Models

QIWEI JIN, DAVID B. THOMAS, WAYNE LUK, and BENJAMIN COPE
Imperial College London

This article explores the application of reconfigurable hardware to the acceleration of financial computation using tree-based pricing models. Two parallel pipelined architectures have been developed for option valuation using binomial trees and trinomial trees, with support for concurrent evaluation of independent options to achieve high pricing throughput. Our results show that the tree-based models executing on a Virtex 4 field programmable gate array (FPGA) at 82.7 MHz with fixed-point arithmetic can run over 160 times faster than a Core2 Duo processor at 2.2 GHz. The FPGA implementation is two times faster than the nVidia Geforce 7900GTX processor with 24 pipelines at 650 MHz, and 27%–35% slower than the nVidia Geforce 8600GTS processor with 32 Pipelines at 1450 MHz. Our preliminary experiments also indicate that while an FPGA implementation can be slower than a GPU, it could be more efficient when power consumption is taken into account.

Categories and Subject Descriptors: B.8.2 [**Performance and Reliability**]: Performance Analysis and Design Aids

General Terms: Design, Performance

Additional Key Words and Phrases: FPGA, HyperStreams, GPU, binomial, lattice, trinomial

ACM Reference Format:
Jin, Q., Thomas, D. B., Luk, W., and Cope, B. 2009. Exploring reconfigurable architectures for tree-based option pricing models. ACM Trans. Reconfig. Techn. Syst. 2, 4, Article 21 (September 2009), 17 pages. DOI = 10.1145/1575779.1575781. http://doi.acm.org/10.1145/1575779.1575781.

1. INTRODUCTION

Tree-based pricing models are one way to value and analyze financial derivatives, such as options. A widely used tree pricing model in finance applications is the binomial model [Hull 2005], since it is simple, efficient, and can

The support of Agility, Celoxica, Xilinx, and UK Engineering and Physical Sciences Research Council grants EP/D062322, EP/D60569/1, and EP/C549481/1 is gratefully acknowledged.
Authors' addresses: Q. Jin, D. B. Thomas, and W. Luk, Department of Computing, Imperial College, 180 Queen's Gate London SW7 2AZ, UK; email: {qj04, dt10, wl}@doc.ic.ac.uk; B. Cope, Circuits and Systems Group, Department of Electrical and Electronic Engineering, Imperial College, 180 Queen's Gate London SW7 2AZ, UK; email: benjamin.cope@imperial.ac.uk.

handle certain types of options (such as American options) that are difficult to price using Monte-Carlo methods. The model is often used to provide prices to a trader, but increasingly is also used as a component of larger applications, where the application may use the model to value hundreds or thousands of options. The trinomial option pricing model is an alternative to the binomial model, but requires fewer tree nodes (computation steps) to achieve the same level of accuracy. As the trinomial model involves more complex computations in one step, it is used less often than the binomial model for simple option valuations. However trinomial models are more widely adopted to evaluate interest rate derivatives [Kramin et al. 2005], since it offers additional freedom in the model that cannot be achieved by binomial models.

Pricing a single option using tree-based models is fast, and can typically be performed in milliseconds on a modern general-purpose processor. However, when huge numbers of options need to be valued, for example if a tree-based pricing model is embedded in a Monte-Carlo simulation, or if a large basket of options is being revalued in real-time using live data-feeds, the valuation of the pricing model can become the main computational bottleneck. This article shows how field programmable gate arrays (FPGAs) can provide a viable method of accelerating tree-based pricing computation, and how the proposed approach can be effectively mapped onto reconfigurable hardware.

The main contributions of this article are:

—two parallel pipelined architectures based on binomial tree and trinomial tree models that are capable of processing multiple trees simultaneously to support concurrent requests for option valuations;

—implementation of the architecture in reconfigurable hardware; exploiting on-chip RAM resources to avoid recomputing costly calculations;

—evaluation of the proposed approach and comparison with alternative implementations based on general-purpose Intel processors and nVidia GPUs (Graphics Processing Units).

In the following, Section 2 states the motivation of this article. Section 3 introduces the binomial option pricing model. Section 4 explains the trinomial option pricing model and compares it with the binomial model. Section 5 suggests an approach to developing hardware architectures for such models. Section 6 explains how the core evaluation computation of the tree-based option pricing model can be implemented in reconfigurable hardware. Section 7 contains results and comparison of the proposed approach and other implementations in general-purpose processors and GPUs, followed by the conclusion in Section 8.

2. MOTIVATION

Previous work on hardware acceleration of financial simulation has focused on Monte Carlo methods. Three examples are given in the following. First, a stream-oriented FPGA-based accelerator with higher performance than GPUs and Cell processors has been proposed for evaluating European options [Morris and Aubury 2007]. Second, an automated methodology has been

developed that targets high-level mathematical descriptions of financial simulation to produce optimized pipelined designs with thread-level parallelism [Thomas et al. 2007]. Third, 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 [Zhang et al. 2005]. All three approaches result in designs based on Monte Carlo methods. However, many financial simulations have alternative solutions, for which techniques such as binomial and trinomial trees will be more effective.

The binomial model can be seen as a discrete-time approximation to the Black-Scholes continuous-time model [Black and Scholes 1973], and the trinomial model can be considered as a variant of the finite difference method [Hull 2005]. We briefly explain the concept in terms of an *American put option*. A *put option* is a contract that gives party A the right to sell some asset S to party B at a fixed price K (called the strike price). The important factor is that the option provides a right, not an obligation: party A can choose whether or not to exercise that right (to sell asset S at price K).

In general the option will only be exercised if $K > S_t$. For example, when the strike price K is greater than the current price of the stock (S_t), party A can buy the asset from the market at a lower price and immediately sell the asset to realize a profit of $K - S_t$. If $K < S_t$ then party A will choose to leave the option to expire and will neither gain nor lose money. In contrast party B has no control over the option, so in the first case B will lose $K - S_t$, and in the second case B will neither gain nor lose. Because party A only stands to gain, and B only stands to lose, B must be offered some kind of compensation. The point of an option pricing model is to determine how much A should pay B in order to create the option contract, or equivalently how much A can charge a third party for the option at a later date.

An *American* option is one where party A can exercise the option at any time up until the option expires at time T. In contrast, a European option is one where the option can only be exercised at a particular time T. All else being equal, an American option must be worth more than a European option with the same parameters, since party A has more flexibility. With the flexibility come more opportunities for profit, which translate to greater possible losses for party B, so more compensation is required for the option contract.

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 very difficult to determine the option price using Monte-Carlo methods, another common method of option pricing mentioned earlier [Thomas et al. 2007]. In contrast, tree-based techniques are able to accurately price both European and American options.

3. THE BINOMIAL OPTION PRICING MODEL

The binomial model works by discretizing both time and the price of underlying asset S, and mapping both onto a binary tree. Each step from the root towards the leaves increases time by one step, and at each node one of the branches leads to an increase in S, while the other branch leads to a decrease in S. This

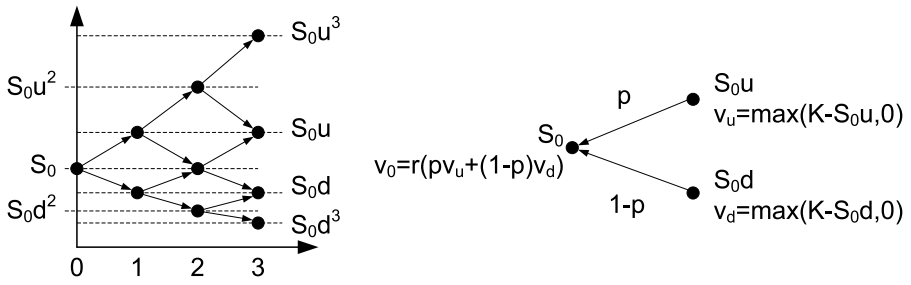


Fig. 1. The left-hand side shows the recombining binary tree of asset prices. The right-hand side shows the valuation of a put option over one time period, with each node showing the asset price on top, and the option price below.

is shown in Figure 1, with time along the horizontal axis, and asset price along the vertical axis.

At each node the upper branch increases the asset price by a factor u, while the lower branch decreases the price by a factor d. At the root of the tree the asset price is S_0, which is the current asset price. At the leaves of the tree are the possible asset prices at time T, which are defined by S_0 and the path through the tree to the leaf. For example, the highest price in Figure 1 is reached by taking only upper branches from the root, so the asset price at that node is S_0u^3. Note that the asset price can only take a fixed number of values, shown as horizontal dashed lines. The tree also recombines, so the leaf node with value S_0u can be reached through three paths (*uud*, *udu*, or *duu*).

The idea behind binomial tree techniques is that the put option is worth $\max(K - S_T, 0)$ at the leaves of the tree. Knowing the value at all the leaves of the tree enables us to work backwards to previous time steps, until eventually the root of the tree is reached. The right-hand side of Figure 1 gives a simplified example over just one time step. The node asset prices are already known (shown at the top of each node label), so the option values at the leaves (shown as v_u and v_d) can immediately be determined. To work back to v_0 we require another piece of information, which is the probability (p) that the asset price will move up. Given p, the expected value of the option at the first node can then be calculated.

Two further considerations are needed for practical use. The first is that interest rate evolution means that money earned in the future is worth less than money earned now. We handle this consideration by applying a discount factor r (where $r < 1$) to option values as we move backwards up the tree. The second is that at some nodes, early exercise may offer a better return than future exercise; so at each node we need to choose the higher of the discounted future payoff versus the payoff from early exercise.

From this discussion, the pricing model can be described as:

$$v_{T,i} = \max(K - S_{T,i}, 0) \tag{1}$$

$$v_{t,i} = \max(K - S_{t,i}, r(pv_{t+1,i+1} + (1-p)v_{t+1,i-1})) \tag{2}$$

$$S_{t,i} = \begin{cases} S_0u^i, & \text{if } i \geq 0 \\ S_0d^{-i}, & \text{otherwise} \end{cases}, \tag{3}$$

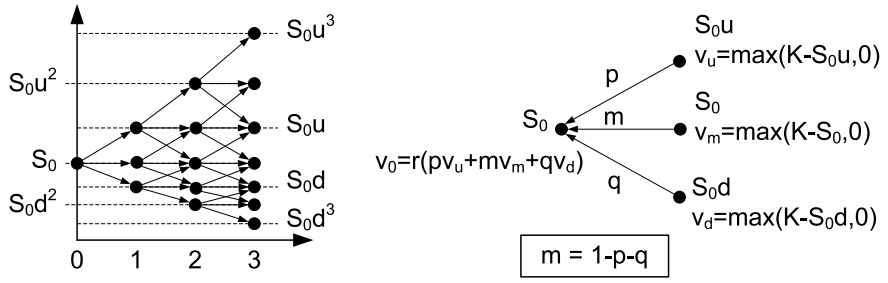


Fig. 2. The left-hand side shows the recombining trinomial tree of asset prices. The right-hand side shows the valuation of a put option over one time period, with each node showing the asset price on top, and the option price below.

where i is an integer indicating the number of steps up or down from the initial asset price, and t is an integer indicating the number of time steps away from the root of the tree, with the leaves at time $t = T$. All other values are real numbers. The inputs to the model are T, S_0, K, u, d, and r, and the output from the model is $v_{0,0}$, which is the estimated price for the option.

The model can be implemented in computational form as a recursive function; however a direct implementation of this function is inefficient unless memoization is used. An efficient solution can be formulated in an iterative form, with an outer loop stepping t backwards from T to 0, and an inner loop calculating the price for each i at level t in the tree. A temporary array holds the intermediate values, and can be updated in place.

4. THE TRINOMIAL OPTION PRICING MODEL

The trinomial option pricing model can be viewed as an alternative to the binomial model. It was initially proposed by Boyle [1986] and later proved by Brennan and Schwartz [1978] to be equivalent to the explicit finite difference method [Hull 2005], another method for American option evaluation. The trinomial model extends the binomial model, by allowing the price to increase or decrease as before, but by also allowing the price to stay the same. The main advantage of a trinomial tree is that it provides an extra level of freedom, making it easier for the tree to represent features of the interest rate process such as mean reversion [Hull 2005]. This extra level of freedom is very useful for modelling interest rate derivatives such as bond options [Kramin et al. 2005]. Generally a trinomial tree will have more nodes than a binomial tree with the same number of steps; therefore it is considered more accurate and will give the same result as a binomial model in fewer steps [Silva 2003]. Typically, an n step binomial tree has $(n+1)(n+2)/2$ nodes whereas an n step trinomial tree has $(n+1)^2$ nodes.

A three-step trinomial tree is shown in the left-hand side of Figure 2, with time along the horizontal axis and asset price along the vertical axis. The right-hand side of Figure 2 is a simple trinomial tree over one time step, where p and q indicate the probability that the asset price will go up and go down

respectively, and m is the probability for the asset price to remain unchanged. Given p and q, the expected option price can be calculated.

The trinomial pricing model for American put option can be described as:

$$v_{T,i} = \max(K - S_{T,i}, 0) \tag{4}$$

$$v_{t,i} = \max(K - S_{t,i}, r(pv_{t+1,i+1} + mv_{t+1,i} + qv_{t+1,i-1})) \tag{5}$$

$$S_{t,i} = \begin{cases} S_0 u^i, & \text{if } i > 0 \\ S_0, & \text{if } i = 0 \\ S_0 d^{-i}, & \text{otherwise} \end{cases}. \tag{6}$$

It can be observed that Equation 5 requires about 33% more computations than Equation 2. However, we are able to implement the trinomial model in a similar way to the binomial one by iterating over an array with nested for loops.

5. MAPPING TREE-BASED MODELS TO HARDWARE

In mapping the binomial model described in Section 3 into hardware, we make two central assumptions.

—The trees use a nontrivial number of time-steps, so the amount of I/O per tree is small compared to the number of nodes that must be evaluated.

—Requests for option valuations are received concurrently, so many individual trees can be valued in parallel.

The first assumption means that we only need to consider evaluation when it is computationally bound, so we can largely ignore the performance of any software to hardware communications channels. If x is the number of time-steps, the number of parameters needed for transfer will be of order x, which mainly comprise the Asset Price Lookup Table and six tree parameters. The Asset Price Lookup Table will have an exact size of $2x + 1$ in both binomial and trinomial valuations. The transfer overhead is insignificant when compared with the number of computations, which varies quadratically with x. For example, if a binomial model of 500 steps is being evaluated, we can expect an Asset Price Lookup Table of 1001 words. Within the model, 1.3×10^5 nodes are needed to be evaluated. Assuming the PCI bus has a bandwidth of 500 MBytes/sec, which is equivalent to 125 MWords/sec if single precision representation is adopted, and the FPGA implementation has a processing speed of 80 MNodes/sec, it is easy to calculate that it takes 8×10^{-6} seconds to finish the transmission and 1.63×10^{-3} seconds to finish the computation. In our case I/O can be pipelined to take place concurrently with computation, hence further reducing the overhead. A further improvement is to compute the lookup table on the fly so that we can process more trees in a batch, reducing the effect of start-up overheads of I/O. We discuss this in more detail in the trinomial example.

The second assumption allows us to use high-latency pipelined functional units to achieve high clock rates while still achieving high throughput, by using the C-Slow approach [Weaver et al. 2003].

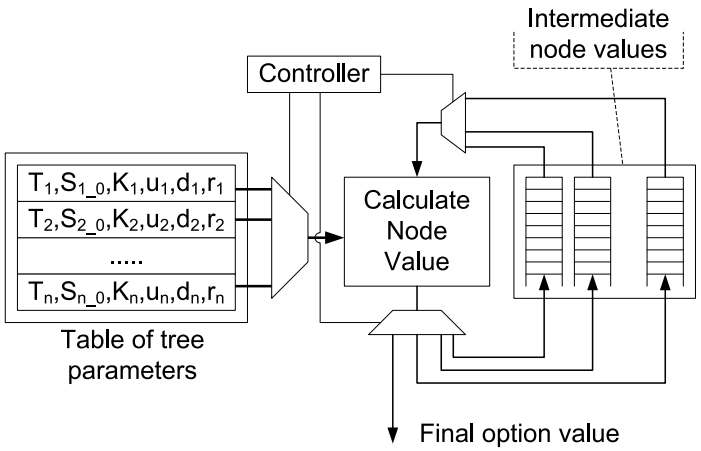


Fig. 3. System architecture for computing the binomial tree model.

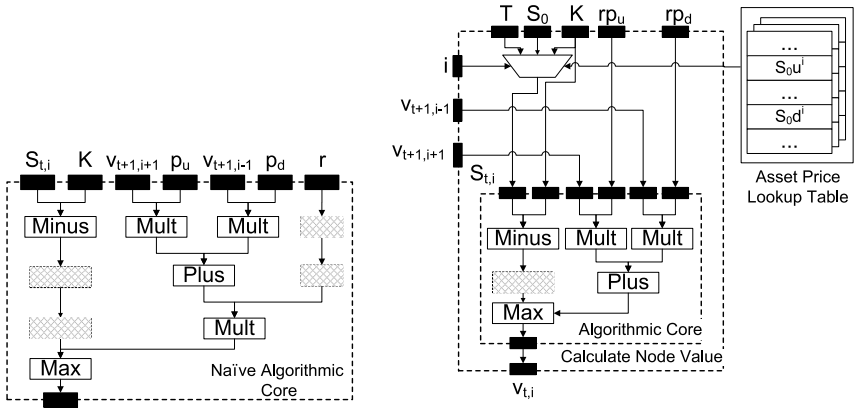
Figure 3 shows our proposed architecture for mapping the binomial tree model into hardware. On the left is a bank of parameter sets, each of which describes a binomial tree that is currently in the process of being evaluated. In the center is a large pipelined block that takes two previously calculated option values and calculates the value of the parent node. To manage temporary storage, a set of buffers (shown to the right) is used; ideally the buffers should be FIFO stream buffers that hold the option values until they are needed again.

FPGA embedded memory (in our case Xilinx Block Select RAMs) is used to implement a lookup table for $S_{t,i}$ (see Equation 2), which is initialized at the beginning of each tree-evaluation run, to remove the need for expensive exponential calculations in hardware.

C-Slow operation can be achieved by modeling multiple trees in parallel: we continuously provide parameters into the pipeline to evaluate other trees while we are waiting for the results required for the next iteration of the current tree. The stream buffers are carefully designed for this approach. A controller manages the overall timing of the system, ensuring that the intermediate values are stored and retrieved correctly, and that the correct parameter set is selected on each cycle.

Figure 4(a) illustrates a straightforward hardware implementation of the core evaluation pipeline. Two adders and three multipliers are required to implement Equation 2. If float or double data types are used in the implementation, multipliers can occupy a significant amount of on-chip resources. On a Virtex 4 xc4vsx55 device, a floating point multiplier consumes around 4% of total slices. By rearranging Equation 2 to Equation 7, it is possible to use one fewer multiplier if, instead of feeding in p_u, p_d, and r separately, rp_u and rp_d are used as two inputs. Note that p_u is the upward probability originally denoted by p in Equation 2 and p_d is the downward probability denoted by $1 - p$.

$$v_{t,i} = \max(K - S_{t,i}, rp_u v_{t+1,i+1} + rp_d v_{t+1,i-1}). \tag{7}$$



(a) A straightforward approach. (b) An improved design.

Fig. 4. Binomial Model: hardware design for the block *Calculate Node Value* in Figure 3. The solid black boxes denote registers and the dotted grey boxes denote pipeline balancing registers that are allocated automatically by *HyperStreams* (Section 6).

The last multiplier can be omitted since the discount factor r is taken into account in the first two multiplications. rp_u and rp_d can be transferred directly from software. Figure 4(b) shows an example of the improved hardware design of the evaluation core. For each tree it evaluates, it takes in a set of parameters provided by the controller from the tree parameters table in Figure 4(b). To optimise performance, a lookup table is initialized with all possible asset strike prices. The architecture takes from the stream buffers, three parameters: the two previous tree node values, and i, the price offset. Using the price offset, i, the current asset price, $S_{t,i}$, can be retrieved from the lookup table.

With all the parameters ready, the algorithmic core in Figure 4 computes the option price, $v_{t,i}$, for the current tree node. The result is then sent back to the stream buffers for later use. The C-Slow method can be implemented here as follows.

—The outside controller is designed to provide a set of correct parameters per clock cycle.

—The lookup tables are correctly initialized beforehand.

—The controller is able to store the result into the correct buffer.

The evaluation core, which calculates the next node value as specified by Equation 2, will require the most logic resources. In the asymptotic case, we would expect the overall performance to be dominated by the size and speed of this block, as the other components consist of a small amount of memories and selection logic.

The trinomial tree model in hardware shares the same main assumptions as the binomial model. It differs from the binomial model in the following aspects.

—The trinomial model is more computationally intensive as the number of computations compared to the binomial model is doubled.

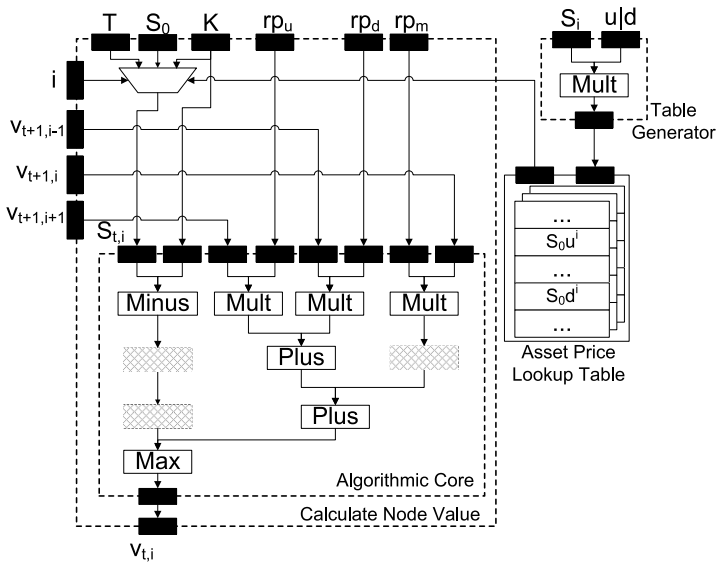


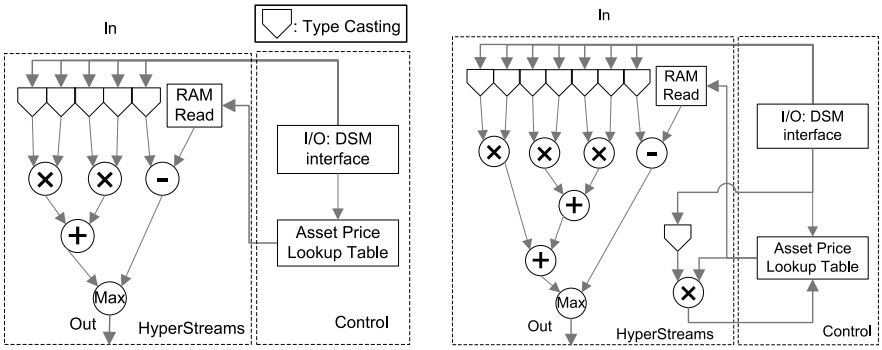
Fig. 5. Trinomial Model: Hardware design for the block *Calculate Node Value* in Figure 3. The solid black boxes denote registers and the dotted grey boxes denote pipeline balancing registers that are allocated automatically by *HyperStreams* (Section 6).

—It requires one extra multiplication and one extra addition within each step.
—It requires twice the memory space to store intermediate values.

The proposed architecture for mapping the trinomial tree model into hardware is similar to what is shown in Figure 3 except that the control logic and the evaluation core are redesigned.

Figure 5 shows a hardware implementation of Equation 5. One more adder and one more multiplier are used when compared with the design in Figure 4. However the pipeline depth only increases by one adder; it is therefore expected that there will be some rise in resource requirement and little increase in pipeline delay.

The box above the Asset Price Lookup Table in Figure 5 shows the logic to generate the lookup table on the fly. By using an extra multiplier, we are able to avoid using expensive exponential operators in hardware. The idea is to start from S_0 in the middle of the lookup table, and to repeatedly multiply it by u; writing the successive values into the lookup table until we reach one end of the price table. Then the same can be done to the other half of the look-up table. If the memory is dual-ported, the two procedures can be done simultaneously. This approach allows us to cache only tree parameters instead of caching large lookup tables, which therefore allows us to transfer trees in a batch from software. The tree parameters in the cache can be fetched by the control logic to generate lookup tables for later use. This reduces the communication overhead further. The table generator runs in parallel with the core evaluation logic to reduce the generator overhead. Extra memory cache is needed to store the generated lookup tables.



(a) Binomial Model (b) Trinomial Model

Fig. 6. The data flow of the hardware part of the tree-based models implemented on FPGA; note the separation of control and pipelined data flow.

In the next section we examine the implementation of the Calculate Node Value block in hardware.

6. THE OPTION UPDATE PIPELINE

Our FPGA implementation of the node evaluation logic to support the tree-based option pricing model is based on *HyperStreams* and the *Handel-C* programming language.

HyperStreams is a high-level abstraction based on the *Handel-C* language [Morris and Aubury 2007]. It supports automatic optimization of operator latency at compile time to produce a fully-pipelined hardware implementation. This feature is useful when implementing complex algorithmic calculations in FPGAs. In addition, *HyperStreams* also provides support for connecting to FPGA resources such as Block Select RAMs.

Figure 6(a) shows a fully pipelined FPGA implementation of the node evaluation logic indicated in Equation 2, while Figure 6(b) shows the implementation of Equation 5. Each symbol shown in a "*HyperStreams*" block in Figure 6(a) refers to a *HyperStreams* operator: for example, $\bigoplus$ for *HsAdd*, RAMRead for *HsRAMRead* and so on. Each arrow from *DSM* (Data Stream Manager), the interface used for hardware-software communication, indicates a stream data element received as an unsigned integer. 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 then cast back to the desired output format using the *HsCast* operator.

The control logic, which is used to send and retrieve data from pipelines, is written in the *Handel-C* language.

To fully utilize the pipeline, double buffering is used to get around the FPGA memory access limitation, since it has lower off-chip memory bandwidth than GPUs and CPUs [Cope 2008]. We give a simple example to illustrate.

Figure 7 shows some code feeding values to a *HyperStreams* pipeline. It waits for the result to come out from the pipeline and then feeds another value.

```
for(i=...){
   HsWrite...;
            ...
   HsRead...;
}
```

Fig. 7. Straightforward code.

```
par{
  for(i=...){
    HsWrite...;
    ...
  }
  for(j=...){
    HsRead...;
    ...
}}
```

Fig. 8. Pipelined code.

The pipeline is not fully utilized since only one pipeline stage will be effectively working at any time. Figure 8 shows an improved approach. Instead of waiting for the result to come out, inputs are read from one memory cache and constantly fed into the pipeline, while results are simultaneously written into another memory cache. Once the evaluation is finished, the result is sent back to software via the *DSM* interface.

The tool flow is as follows: *Handel-C* source code is synthesized to EDIF using the Celoxica DK5 suite, which supports *HyperStreams*. Xilinx ISE 9.2i project navigator is used to place and route the design.

The target device on our Celoxica ADMXRC4SX platform is an xc4vsx55 FPGA from the Xilinx Virtex 4 family, but it would be simple to retarget our design to other FPGAs supported by the Celoxica DK5 suite.

7. RESULTS

The American put option benchmark has been calculated using three different FPGA implementations in different numerical representations as well as a reference PC implementation. All the FPGA implementations are compared to software implementations on a PC that provides the reference. The reference Intel PC1 implementation is based on C++ code fully optimized for local hardware profile running on a 2.2 GHz Core2 Duo processor with 2 GB of RAM and Windows XP pro operating system. The reference Intel PC2 is a 3.2 GHz Pentium4 processor with 1 GB of RAM and Linux operating system; this implementation involves fully optimized C++ code with *Intel SSE3* enabled.

The FPGA device utilization figures are shown in Table I. The results indicate that less than half of the FPGA device is utilized in all three cases

Table I. Binomial/Trinomial Performance/Area Results for Xilinx xc4vsx55 FPGA. The Percentage Shows Utilization of a Specific FPGA Resource; Note that Acceleration is Compared with the Reference PC1

	FPGA Binomial			FPGA Trinomial		
	Double	Single	Fix	Double	Single	Fix
Slices	7,162 (29%)	3,805 (15%)	1,740 (7%)	11,113 (45%)	5,898 (24%)	2,773 (11%)
FFs	5,924 (12%)	2,914 (5%)	1,756 (3%)	9,182 (19%)	5,170 (10%)	2,608 (5%)
LUTs	6,020 (12%)	3,238 (6%)	782 (1%)	9,331 (19%)	6,113 (12%)	1,422 (2%)
BRAMs	20 (6%)	18 (5%)	18 (5%)	24 (7%)	20 (6%)	20 (6%)
DSPs	32 (6%)	8 (1%)	8 (1%)	48 (9%)	12 (2%)	12 (2%)
MHz	67.3	76.0	82.7	65.2	69.1	76.3
Replication (cores/chip)	3	6	14	2	4	9
Processing Speed (M nodes/sec)	67.3	76.0	82.7	65.2	69.1	76.3
Mean% Error	0%	0.03%	0.05%	0%	0.04%	0.07%
Acceleration (1 core)	9.3×	10.6×	11.5×	10.3×	10.6×	11.7×
Acceleration (replicated cores)	27.9×	63.6×	161×	20.6×	42.4×	105×

involving double-precision, single-precision, and fixed-point arithmetic for both binomial and trinomial models. Hence performance improvement can be achieved by replicating the evaluation core in a single device. Although only the nodes at the same step of a binomial tree can be computed simultaneously, acceleration can be achieved by evaluating several trees in parallel.

The lower part of Table I shows the space occupancy/acceleration results for different precision implementations, including core replication that can be done on a single device to gain further performance. The left-hand side of the table shows results for the binomial implementation and the right-hand side gives data for the trinomial implementation.

Graphics processing units (GPUs) are another alternative to CPUs for computational-intensive tasks, and have also been used for financial computation [Giles and Su 2007]. Our first GPU design is implemented on an nVidia Geforce 7900GTX device with 512 MB of onboard RAM. The second GPU design is based on nVidia's new CUDA platform [Hennessy and Patterson 2006] and running on a Geforce 8600GTS with 256 MB of onboard RAM. The stated GPU clock rates in Table II are the peak rates specified by nVidia. Double-precision floating-point arithmetic is unavailable on Geforce 7900GTX [Morris and Aubury 2007] and is not yet supported by CUDA 1.1 for Geforce 8600GTS.

Table II shows the data for two GPUs and two reference PCs. The speed benchmark for both reference PCs and GPUs is to evaluate a 1×10^3 step binomial tree for 2^{20} times.

First consider the acceleration of the FPGA over the Intel PC1 and PC2 benchmarks. From the results, it can be seen that, for the single core binomial implementation, the 32-bit 16.16 fixed-point implementation offers an 11.5 times acceleration, while the 32-bit single precision floating-point and

Table II. Binomial/Trinomial Performance Geforce 7900GTX (GPU1) and 8600GTS GPU (GPU2), Intel Core2 Duo (PC1) and Pentium4 (PC2) Processors; Note that Acceleration is Compared with the Reference PC1

	Binomial				Trinomial		
	GPU1	GPU2	PC1	PC2	GPU2	PC1	PC2
	Single	Single	Double	Double	Single	Double	Double
MHz	650	1450	2200	3200	1450	2200	3200
Replication (pipelines)	24	32	1	1	32	1	1
Processing Speed (M nodes/sec)	477	1476	7.2	4.3	927	6.5	3.1
Mean% Error	0.03%	0.03%	0%	0%	0.04%	0%	0%
Acceleration	–	–	1×	0.6×	–	1×	0.5×
Acceleration (replicated cores)	66×	205×	1×	0.6×	142×	1×	0.5×

the 64-bit double precision versions offer 10.6 times and 9.3 times speedup respectively.

Not surprisingly, fixed-point arithmetic is faster and smaller than floating-point arithmetic in an FPGA. For instance, three cores can be implemented in a single Xilinx xc4vsx55 device if double precision arithmetic is adopted, which leads to a 27.9 times speedup over optimized software running on a Core2 Duo processor. In contrast, 14 cores in fixed-point arithmetic can be implemented in the same Xilinx xc4vsx55 FPGA, indicating a 161 times acceleration for multiple binomial trees evaluated in parallel. It is worth noting that we choose the 16.16 fixed point representation because it provides sufficient accuracy (less than 0.1% error) for our application. It may still be possible to apply word length optimization to reduce the number of bits in the fixed point representation while retaining acceptable accuracy, but the details are beyond the scope of this article.

The trinomial implementation is generally around 1.5% slower than the binomial implementation. The small timing overhead is introduced by a longer pipeline and the associated control logic. The trinomial implementation is also 40% larger than the binomial implementation. The significant increase in space is due to the additional multiplier and adder in the Algorithmic Core and the multiplier in the Table Generator. In Section 4 we pointed out that the trinomial model will produce a result with the same accuracy as a binomial model but in fewer steps. Up to $x/2$ fewer steps are required for the trinomial model compared to the binomial, where x is the number of steps required for the binomial model to get the same result. The trinomial model also provides an effective means of modelling interest rate derivatives [Kramin et al. 2005], such as American bond options [Hull 2005]. These properties compensate for the drawbacks of the trinomial implementation.

The floating-point implementations on GPUs are faster than the corresponding single precision implementations on FPGAs with replicated cores in both binomial and trinomial implementations. However it is worth noting that the difference between the FPGA and GPU1 is within a factor of 2. This is because both the GPU1 and FPGA approaches are based on straightforward implementations without including further parallelism and optimization.

On the other hand, the implementation on GPU2 is based on CUDA and seeks to exploit the full utilization of GPU resources by:

—Scheduling options to be evaluated concurrently on different multiprocessors.

—Allowing multiple threads to simultaneously evaluate a single step.

—Using double buffering in local shared memory to avoid highly delayed access to global memory [Podlozhnyuk 2008].

It is interesting to see that GPU2 is three times faster than the FPGA in both binomial and trinomial single-precision floating point implementations. We only had a 2% latency increase to evaluate a trinomial tree on the FPGA, and asymptotically doubled the computation time when modelling it on CPUs and GPUs having extra *multiply* and *add* operations. This is because in the FPGA case we trade space for speed. The logic for the trinomial model is asymptotically twice as large as the binomial one, hence we are not able to map as many cores on a single FPGA as in the binomial case. The other possible reason is that CUDA has instruction level parallelism that is able to reduce computation time for complex expressions.

It is not surprising to see that GPU2 is over 3 times faster than GPU1. GPU2 has more pipelines than GPU1 (32 stream scalar processors versus 24 pipelines, used here to implement scalar operations), which is 1.5 times better performance than GPU1; and the shader cores used for computational purposes on GPU2 runs two times faster than GPU1 (1450 MHz versus 650 MHz). The difference in onboard memory between GPU1 and GPU2 is irrelevant since our problem is not data-bound but computation-bound. We believe that the Virtex 4 FPGA and the Geforce 7900GTX GPU are broadly comparable, since both are based on 90 nm technology.

Additional performance improvement can be expected if the latest Geforce 8800 class GPU is adopted: a 4 times speedup can be gained from increased parallelism (128 versus 32 stream scalar processors) and some further speedup from clock speed increment (from 1450 MHz to 1600 MHz). However, the 8800 class GPUs adopt 80 nm technology, and they should be compared with the latest FPGA technology such as Virtex 5 from Xilinx and Stratix IV from Altera. Geforce 8600GTS is used as a benchmark to indicate how CUDA can be used to exploit resources for financial computations in similar areas.

If a larger Virtex 4 or even Virtex 5 device is used which has 4 times or more slices than that on our Xilinx xc4vsx55 device and a higher basic clock frequency, then at least 4 times speedup can be achieved without further optimization.

On the other hand, single precision operators in FPGAs can run at a clock rate of up to 322 MHz [Xilinx 2006]; our current implementation at 76 MHz has much scope for improvement. Furthermore, if we are able to reduce the size of our design to half of the original size by means of further optimization, it would enable us to put twice as many evaluation cores on the FPGA, producing a further 2 times speedup.

The speed benchmark is purely for the purpose of measuring maximum evaluation speed, hence we chose to measure 1024^2 trees with the same depth

to achieve maximum parallelism. In reality, it is rarely the case that 1024^2 options will be changing price at the same time and all of them, sharing the same depth. Our experiments show the GPU processing speed is halved when 1024 options are evaluated and nearly quartered when 512 options are evaluated. Other benchmarks, which involve valuation of 2048 options up to 1024^2 options, are also included in our experiments. We find the GPU processing speed goes up logarithmically with the number of options, while the processing speed of the FPGA remains almost constant in all our experiments. The requirement of varying tree depths for neighboring nodes would map inefficiently to the GPU SPMD (single program multiple data) programming model. It can be expected that the one which has the largest number of steps will become the bottleneck in GPU. An FPGA implementation can be designed with data path flexibility to alleviate this restriction.

From experience there is a tradeoff when using *HyperStreams*, between the development time and the amount of acceleration that can be achieved. Although we are able to easily implement complex algorithms in FPGAs with *HyperStreams*, the highest possible performance and utilization 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* implementation still gives a satisfactory result with significant acceleration over the software implementations. Hence *HyperStreams* is useful particularly for rapidly producing prototypes to explore the design space; once promising architectures are found, further optimizations can be applied.

The mean percentage error method is used to measure the accuracy of all the implementations. Random number generators are used to simulate all the option pricing scenarios and the results are compared to the corresponding results produced by the benchmark software version. The mean percentage error is calculated as:

$$\textit{Mean Percentage Error} = \frac{100}{n} \sum_{i=1}^{n} \frac{|v_i - v_{i^*}|}{v_{i^*}}, \tag{8}$$

where v_i is the option price produced by a particular implementation; v_{i^*} is the corresponding result produced by the reference software and n is the number of experiments done.

While the FPGA designs are slower than those on GPU2, they are more efficient when power consumption is taken into account. For instance, using Xilinx XPower Estimator, we find that a 3-core design on an xc4vlx160 FPGA at 160MHz takes 5.7 Watts, while the Geforce 8600GTS requires around 71 Watts [Vorobiev and Else 2007]. Hence a 500-step calculation would run at 18 million nodes per second per watt on the xc4vlx160, which is around 6 times more efficient than the Geforce 8600GTS.

8. CONCLUSION

This article describes a new architecture for accelerating option pricing models based on both binomial and trinomial trees. The proposed design involves

a highly pipelined datapath capable of supporting multiple tree calculations in parallel, which can deal with concurrent requests for option valuations. We have implemented our design onto an xc4vsx55 FPGA, and demonstrate that our implementations can generally run more than 100 times faster than a Core2 Duo processor. They are more than 2 times faster than an nVidia GPU that does not support CUDA, and are 27%-35% slower than a CUDA supported nVidia GPU.

Further work is planned to carry out complete hardware implementation of the binomial and trinomial tree models, with various speed and area optimizations based on hardware cores with the highest performance. Task scheduling can also be studied to find the best ways of utilizing the pipelines in FPGAs and GPUs in order to further enhance performance.

Additional improvements in power consumption can also be obtained for our FPGA designs. Since both pipelining [Wilton et al. 2004] and word-length optimization [Constantinides 2006] can improve performance and reduce power consumption, it would be worthwhile to investigate how these two techniques can be used in automating domain-specific strategies for producing tree-based designs that best meet user requirements in speed, area, and power consumption. More sophisticated comparisons with the latest GPUs from nVidia and AMD/ATI, Altera FPGAs, and the Cell Broadband Engine [Agarwal et al. 2008] are also planned.

REFERENCES

AGARWAL, V., LIU, L.-K., AND BADER, D. 2008. Financial modeling on the Cell broadband engine. In *Proceedings of the IEEE International Symposium on Parallel and Distributed Processing (IPDPS'08)*. 1–12.

BLACK, F. AND SCHOLES, M. 1973. The pricing of options and corporate liabilities. *J. Polit. Econ. 81*, 637–659.

BOYLE, P. P. 1986. Option valuation using a three jump process. *Inter. Opt. J. 3*, 7–12.

BRENNAN, M. J. AND SCHWARTZ, E. S. 1978. Finite difference methods and jump processes arising in the pricing of contingent claims: a synthesis. *J. Financ. Quant. Anal. 13*, 3, 461–474.

CONSTANTINIDES, G. A. 2006. Word-length optimization for differentiable nonlinear systems. *ACM Trans. Des. Autom. Elect. Syst. 11*, 1, 26–43.

COPE, B. T. 2008. Video processing acceleration using reconfigurable logic and graphics processors. Ph.D. thesis, Imperial College London.

GILES, M. AND SU, X. 2007. Notes on using the nVidia 8800 GTX graphics card. Oxford University.

HENNESSY, J. L. AND PATTERSON, D. A. 2006. *Computer Architecture* 4th Ed: *A Quantitative Approach*. Morgan Kaufmann.

HULL, J. 2005. *Options, Futures, and Other Derivatives* 6th Ed. Prentice Hall.

KRAMIN, M. V., KRAMIN, T. V., YOUNG, S. D., AND DHARAN, V. G. 2005. A simple induction approach and an efficient trinomial lattice for multi-state variable interest rate derivatives models. *Rev. Quant. Finan. Acc. 24*, 2, 199–226.

MORRIS, G. AND AUBURY, M. 2007. Design space exploration of the European option benchmark using Hyperstreams. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL'07)*. 5–10.

PODLOZHNYUK, V. 2008. Binomial option pricing model. http://www.nvidia.com.

SILVA, A. F. C. 2003. *The Trinomial Option Pricing Model: An Improvement Over The Binomial Lattice?*

THOMAS, D., BOWER, J., AND LUK, W. 2007. Automatic generation and optimisation of reconfigurable financial Monte-Carlo simulations. In *Proceedings of the International Conference on Application-Specific Systems, Architectures and Processors*. IEEE.

VOROBIEV, A. AND ELSE, A. B. A. S. 2007. nVidia GeForce 8600 GTS Spec. http://www.digit-life.com/.

WEAVER, N., MARKOVSKIY, Y., PATEL, Y., AND WAWRZYNEK, J. 2003. Post-placement C-slow retiming for the Xilinx Virtex FPGA. In *Proceedings of the ACM/SIGDA 11th International Symposium on Field Programmable Gate Arrays (FPGA'03)*. ACM, 185–194.

WILTON, S. J., ANG, S.-S., AND LUK, W. 2004. The impact of pipelining on energy per operation in Field-Programmable Gate Arrays. In *Proceedings of the International Conference on Field-Programmable Logic and its Applications*. Springer-Verlag, 719–728.

XILINX. 2006. Floating-point operator v3.0 manual. http://www.xilinx.com.

ZHANG, G., LEONG, P., HO, C., TSOI, K., CHEUNG, C., LEE, D.-U., CHEUNG, R., AND LUK, W. 2005. Reconfigurable acceleration for Monte-Carlo based financial simulation. In *Proceedings of the IEEE International Conference on Field-Programmable Technology*. 215–224.

Received June 2008; revised November 2008; accepted December 2008