Energy-Efficient FPGA Implementation for Binomial Option Pricing Using OpenCL

Valentin Mena Morales*†, Pierre-Henri Horrein*, Amer Baghdadi*, Erik Hochapfel†, Sandrine Vaton*
*Institut Mines-Telecom; Telecom Bretagne; CNRS Lab-STICC, IRISA, Brest, France
†ADACSYS, 7 rue de la Croix Martre, Palaiseau, France
Email: {valentin.menamorales,ph.horrein,sandrine.vaton,amer.baghdadi}@telecom-bretagne.eu {erik.hochapfel}@adacsys.com

***Abstract*—Energy efficiency of financial computations is a performance criterion that can no longer be dismissed, and is as crucial as raw acceleration and accuracy of the solution. In order to reduce the energy consumption of financial accelerators, FPGAs offer a good compromise with low power consumption and high parallelism. However, designing and prototyping an application on an FPGA-based platform are typically very time-consuming and requires significant skills in hardware design. This issue constitutes a major drawback with respect to software-centric acceleration platforms and approaches. A high-level approach has been chosen, using Altera's implementation of the OpenCL standard, to answer this issue. We present two FPGA implementations of the binomial option pricing model on American options. The results obtained on a Terasic DE4 - Stratix IV board form a solid basis to hold all the constraints necessary for a real world application. The best implementation can evaluate more than 2000 options/s with an average power of less than 20W.**

I. Introduction

FPGAs (Field Programmable Gate Arrays) can be used as accelerators for financial computations, provided that the algorithm to be implemented can be parallelized to fit optimally on the FPGA's hardware resources. FPGAs provide benefits in terms of power consumption and computation time over a pure software solution, but usually come at a cost in programmability. As opposed to classic HDL-based design (High Description Language: VHDL, verilog, etc.), some approaches [1] attempt to reduce development time by giving access to a high-level framework. For instance, the OpenCL standard (Open Computing Language) supports FPGA targets, through Altera's implementation of an OpenCL compiler. Altera's OpenCL compiler has been used in this work so as to speed up a financial computation (pricing of American options, as detailed in Section III) within a specific set of constraints.

This work aims at providing an architecture that can price 2000 option values under a second while being powered by the user's workstation. These constraints define a use case where a trader can use our work to estimate the implied volatility curve of an option [2], giving him information on the expected evolution of this financial product. When a volatility curve of an option with a specific set of parameters is known, a trader can replace the constant volatility used to model the evolution of this option with the computed volatility and reach more accurate levels of modelization. A second per volatility curve (2000 option values per volatility curve for accuracy considerations) is a duration short enough to fit within the user's timeframe when making a decision. So as to be certain that the accelerator can be integrated into any user's existing infrastructure, a design limit of 10W has been set for its power consumption. The market data and the reference prices from which the 2000 input values are generated are based on a binomial representation. This restricts our study to the pricing of American options by a binomial model [3].

In Section II we present other works related to the subject. Section III describes the financial application as well as the OpenCL standard, and Sections IV and V detail respectively an architecture study of OpenCL implementations and their performance results.

II. Related Work

The acceleration of financial algorithms has been the subject of many articles, but most works focused on the acceleration factor of the implemented solution, compared to a reference software. Still, the energy cost of acceleration is starting to be taken into account as a key performance criterion, along computing times and accuracy. Following this trend, de Schryver, et al. [4] have presented a benchmark to compare option pricing accelerators between each other, and not only between an accelerator and a reference software. They define an option pricing accelerator as:

- a problem, usually finding the price of a financial product,
- a mathematical model used to predict this product's behavior,
- a solution to price this product, meaning an algorithm and its implementation.

This benchmark includes energy consumption as a criterion of discrimination between solutions (J/option). They applied this methodology to a design space exploration for the pricing of barrier options in the Heston model which led to the selection of a Multi-Level Monte Carlo method as the best compromise between acceleration, accuracy and energy consumption.

The Monte Carlo method and its optimizations have been extensively studied due to its massive parallelism. Accelerating this method on parallel hardware such as GPU [5], [6] or FPGA [7], [8] is quite efficient. However, the acceleration factors that can be achieved are counterbalanced by the slow convergence rate of this method. The Monte Carlo method is best suited to complex model evaluation or to problems with high dimensionality. Complex models are usually difficult to price with other methods, while high dimensionality problems benefits from the Monte Carlo's linear increase in complexity method with dimensionality. Most other pricing methods see their complexity increase exponentially with dimensionality (quadrature methods, finite differences methods, etc.).

IEEE978-3-9815370-2-4/DATE14/©2014 EDAA

The binomial option pricing model has seen less interest, as they are harder to efficiently implement on classic acceleration hardware, despite the possible gains in computing times. Still, the work of Jin, et al. [9] can be mentioned, as well as [10] (both on reconfigurable targets), and [11] for an architecture based on a GPU, closer to the industry standards of CPU and GPU clusters. Although not explicited in those papers, one benefit of using FPGAs for acceleration is their low consumption, when compared with GPUs and CPUs. Indeed, FPGAs usually consume an order of magnitude less power than CPUs or GPUs ($\approx 10W$ for FPGA and $\approx 100W$ for GPUs and CPUs). Those implementations price slightly higher than 10^3 American options/s with high precision for the fastest of them [10], [11]. They can achieve better acceleration factors compared to a software reference in specific cases, when restrictions on accuracy are either alleviated (fixed precision implementations) or strengthened (higher time discretization steps). Unfortunately, the above referenced works do not consider the energy consumption criterion.

Jin, et al. have also carried out a survey of different methods of model evaluations [12]. They conclude that quadrature methods are the best compromise to price American options, while tree-based methods are optimal when time-to-solution is a key constraint. The results they described were obtained on floating-point and fixed precision.

III. Context

A. Option Pricing

The main aim of financial applications when used as decision aid tools is to provide the user with information on the expected evolution of financial products. Those applications can, for instance, provide an accurate estimation of a product value at a given time or the risk associated with an investment. Such applications either run on a server cluster to accelerate them, or on the user's personal computer when cost is a bigger constraint than computation time.

The algorithm targeted in this paper implements a binomial model to price American options. An option is a financial product that gives the right (but not the obligation) to buy (*call* option) or sell (*put* option) an asset before a given date. An option pricing model tries to estimate an option price when there exists no analytical solution to evaluate it. Many types of options exist, each corresponding to a slightly different version of this definition. For instance, European options give the right to buy (or sell) an asset at a predetermined date, whereas American options give the right to do so at any given time before its expiration date. The value of the latter thus depends on the asset price during the whole life of the option. This means that the optimal value of an American option is the maximum of all its possible values. The time dependancy introduced by this maximum renders it non trivial to compute (contrary to European options).

B. Binomial Model

The binomial model is a lattice-based method to price options, with a time discretization (*cf.* Figure 1). To simulate the evolution of the asset over time, the asset price can either increase or decrease by a fixed amount at each time step, with respective probabilities p and $q = 1 - p$. The tree is recombining: if an asset value increases once before decreasing once, it keeps the same value. This means that the number of possible values only increases by one at each step. If Δt is the time discretization step, at time $t = n\Delta t$, n values are possible for the underlying asset. The maximum time, which is the option expiration time, is noted T, and N values are possible at time T ($T = N\Delta t$). At time t, possible asset prices are noted $S_{t,k}$ with $k \in 1, \ldots, n$. The option value at time t, noted $V_{t,k}$, is computed as the maximum between its value if it were exercised right now and its potential value before expiry. In order to know both values at each node (i.e. at each tree coordinate (t, k)), the tree is computed backward, starting with the leaves, and ending with the option value $S_{0,0}$ at time $t = 0$. $S_{0,0}$ represents the option maximum value on the considered time frame. The values at the leaves ($S_{T,k}$, for $k \in 1, ..., N$) correspond to the pricing of European options and can be found analytically.

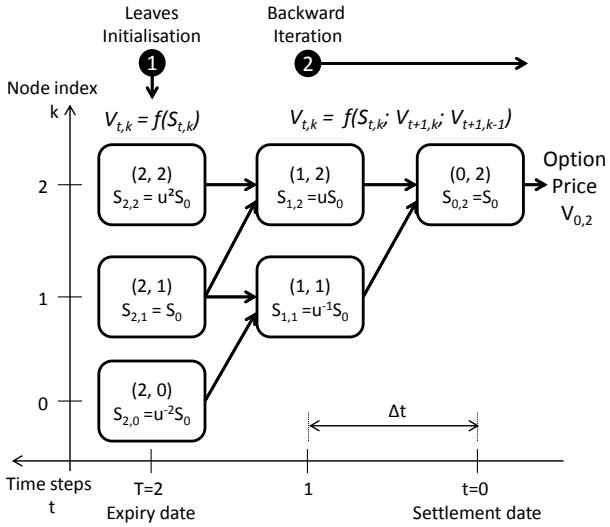

Figure 1. Binomial tree applied to the pricing of an American option

Each node is updated using the following recurrence formulas (for a call option) [2]:

$$\begin{aligned} S_{t,k} &= dS_{t+1,k} \\ V_{t,k} &= max(S_{t,k} - K; rpV_{t+1,k} + rqV_{t+1,k-1}) \end{aligned} \quad (1)$$

where d, K, r, p, q are option dependent parameters, where:

- $d = e^{(-\sigma\Delta t)}$, where σ is the volatility of the option,
- K is the strike price (the price at which the option can be bought),
- r is the risk free rate (assuming risk neutral valuation).

C. A High-Level Approach: OpenCL

OpenCL [13] is a framework for parallel programming on heterogeneous platforms. It is based on a runtime host library and C99 extensions for device programming, adapted to support vectorized data types, synchronization points, and other functionalities. OpenCL is a standard in parallel programming, which has been implemented on several hardware architectures by manufacturers (e.g.: GPUs, CPUs, FPGAs, etc.). An OpenCL program can be executed on any of those devices with only a handful of modifications, allowing portability. As shown on Figure 2, an OpenCL device is subdivided into compute units, which execute multiple copies (*work-items*) of a piece of code (*kernel*). However, performances can vary depending on the compatibility between the program and the architecture of the targeted device.

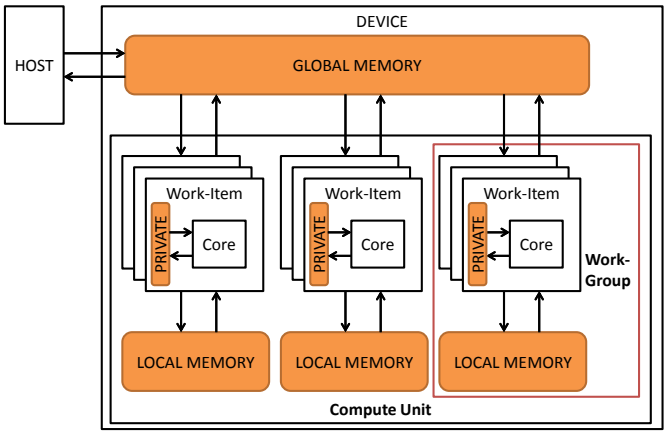

Figure 2. OpenCL platform

The master of an OpenCL architecture (the *host*) handles the application's data-flow through queues of orders to the devices it is connected to. Data movement is thus explicitly specified by the programmer. This data management relies on a relaxed memory consistency model, with three memory levels: global, local and private. The work-items on a device are organized into *work-groups* that share a common memory region (local memory) and synchronization points. This memory level acts as a cache-coherent memory for the programmer, which ease any data partitioning problem that may arise: every work-item within a work-group can access any data stored in the address space of the local memory. The global memory can be accessed by any work-group and by the host. Access to global memory can be coalesced to reduce latency, provided that accessed pieces of data are adjacent. Nevertheless, accesses to global data is always slower than accesses to local data (up to an order of magnitude in bandwidth). Finally, each work-item has access to a single private memory region.

Two OpenCL devices have been used as implementation targets: an FPGA board as the final target, and a GPU as an initial target. The use of a GPU board reduced development time through faster compilation and thus shorter debug and development iterations. It has also served as a reference to compare the results accuracy and computation times of the proposed acceleration solutions.

IV. Architecture Study

A. Straightforward Implementation

Compared with a classic HDL design methodology, work scheduling on hardware resources is delegated to the device in OpenCL. The developper simply enqueues the necessary number of work-items in the host program and lets the workload get dispatched on the device's resources as best as possible, with no need to program a layer of control over its operative core. Enqueueing far more work than what can be processed at once on the device can then help it optimize its use of hardware resources, and is actually necessary to reach an acceleration factor that balances the communication overhead between the host and the device.

For this reason, the first approach implements a dataflow paradigm, supported by kernels independent from each other and forming a highly scalable application. A single tree node of the binomial tree (Figure 1) is computed by a kernel, which is enqueued enough times so that a full tree datapath is computed at once. This amounts to $N(N+1)/2$ work-items, with N the total number of discretization steps. Each stage of the tree (i.e., each time-step) corresponds to a different pipelined option being computed. This efficiently pipelines $N+1$ options on the FPGA device, one option entering the tree and N options at each step within the tree. An option is computed by calling this network of kernels N times, until it exits the pipeline. Those iterations are controlled by the host program, which schedules the memory transfers between the host and the device and the sequential execution of the kernel batches. Four instructions are executed by the host during each batch: initializing the data necessary to fill the first addresses of the input buffer, writing this data to the device global memory, enqueueing the kernels and reading a result from the global memory. For instance, let us consider the pricing of 2000 options with 1024 steps. The tree for one option will roughly contain 5.10^5 tree nodes. The whole tree must be processed for each option, which means that approximately 1.10^9 tree nodes (and thus kernels) must be processed. The results of each batch is stored in global buffers, containing all the internal data. The use of global buffers enables data to be stored between two batches, and enables this data to be accessed by any kernel. To avoid any memory conflict, ping-pong buffering is used (one buffer is read while the other one is written). Those buffers are switched between each batch to let data flow through the network. Data updated during each batch are stored in these buffers (i.e. $S_{t,k}$, $V_{t,k}$, and indexes). Option-dependent data, yet that is constant during an option pricing, is stored in another global buffer. Figure 3 describes this data-flow, applied to the tree shown in Subsection III-B. The tree is displayed flattened here, with the

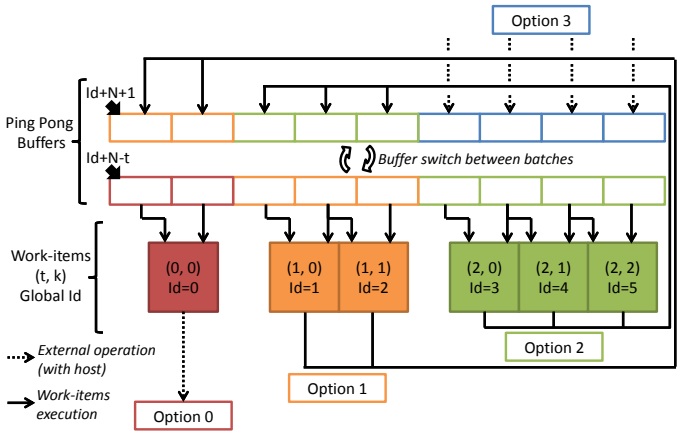

Figure 3. Straightforward implementation of the binomial tree illustrated in Figure 1

last tree node on the left side (at $(t,k) = (0,0)$ and the first encountered nodes on the right side (from $(t,k) = (2,0)$ to $(t,k) = (2,2)$. Option 0 is being computed by the work-item $(0,0)$ and will be read right before the next batch of kernels is enqueued, while three other options are in the pipeline, options 1 and 2 being processed (respectively) by the work-items $(1,0)$ and $(1,1)$ and $(2,0), (2,1)$ and $(2,2)$ while option 3 is being written in the buffer by the host. Two buffer addresses are read by each work-item to compute a tree node value (following recurrence formulas defined in Equation 1). The *id* indicated in each work-item stands for their respective global index, starting at 0 for the first encountered work-item, at the $(2,2)$ position in the tree.

Work-item indexing is a key point in OpenCL programming. Each work-item has several indexes associated to it:

- a global index, unique to each work-item and ranging from 0 to the number of enqueued kernels ($N(N+1)/2-1$ here),
- the index of its work-group,
- and its individual (local) index inside this work-group.

In this implementation, the address of a work-item input is a function of its time discretization step t within the binomial tree. However, work-items in a work-group do not necessarily correspond to tree nodes from the same time discretization step, as work-group size is identical for each work-group. Work-items can only be identified by their global index, and their input addresses are then $(id+N-t)$. Computing time steps within the work-item would be too costly in terms of computing resources. They are stored in a constant buffer, allowing work-items to determine their Read addresses at the beginnning of each batch ($Id+N-t$). Write addresses are less complex and depend only of the global index of the work-items ($Id+N+1$).

B. Optimized Approach

The above described dataflow approach is scalable and easily implemented. However, it suffers from several drawbacks. First, the host iterates through each batch of kernels to continuously fill the pipeline, and in doing so results in an overhead in computation time. Memory operations and work-items executions are overlapped with one another and synchronized by the host, but they still incur a cost in computation time. Besides, the structure of the work-items is not kept between batches: private memories are emptied and local memories are unused, which results in new copies of previously processed data from the global memory to the work-items at the beginning of each batch. A way to solve these issues lies into a reordering of work-items into work-groups, where this division of work-items carries information meaningful for the computation of a binomial tree. A task-based parallelism fits this work division, with each work-group assigned to an option pricing (in other words, a full binomial tree). To reduce the amount of data copied during a computation, a work-item can be asssigned to the computation of a *row* in a tree, where a row can be defined as every tree node (t,k) with k constant. Following the terminology previously used in this work, a work-item would be assigned to the computation of the tree nodes (t,k), in a loop from $t=T$, down to $t=N-(k+1)$, at which point any following computation would be useless (i.e. outside of the tree). T work-items are then necessary at the beginning of the computation to initialize the leaves, and one less work-item is supposed to be computed between each time step. With such a repartition, options parameters, as well as $S_{t,k}$, can be kept in private memory with $S_{t,k}=dS_{t-1,k}$ updated by the work-item (t,k) at the beginning of each iteration. All shared data between work-items (i.e. $V_{t,k}$ for $k\in\{0...N\}$) are stored in local memory, which is possible as they are all in the same work-group. As local memory is scarcer than global memory, no ping pong buffers are used; they are replaced by a single local buffer, associated with local synchronization points (*barriers*) and temporary copies to avoid memory conflicts (*cf.* Figure 4). From the host point of view, three commands must be executed to run this computation:

1) copying all option parameters in global memory,
2) enqueueing enough kernels to process all the data,
3) and read back the final results from global memory.

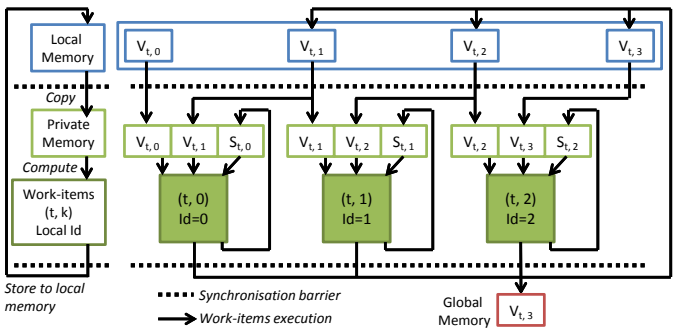

Figure 4. Data-flow during the computation of a binomial tree (Figure 1) with an optimized kernel

Option-dependent data is transfered once in global memory during the initialization step (1), and is then copied in private or local memory during the computation of the tree leaves (not displayed on Figure 4). Similarly, the results are read from global memory (3) only when the full workload has been processed, i.e. when all options have been computed. Compared with the straightforward implementation, host-device interaction is reduced to a minimum. The main restrictions on this implementation are linked to OpenCL constraints on the use of work-groups. The work-group size is a constant defined when enqueueing a kernel on a device, and cannot be modified on the fly. When a complete tree row has been processed, the corresponding work-item is either left idle or its results are ignored; hardware resources are unlikely to be reused. Besides, even if the kernel is more complex, only $N*N_{op}$ work-items have to be enqueued to compute N_{op} options: more options may have to be priced to reach device saturation, which is a requirement in order to reach maximum raw performance.

V. Results

In order to validate the approach, both proposed architectures have been implemented. In this section, the results of these implementations are presented.

A. Test Environment

Three target technologies were considered: a CPU on which the reference software ran, a GPU for development and comparison purposes, and an FPGA. The CPU is a quadcore Intel Xeon X5450 running at 3.0 GHz, the reference software being written in C. A single core of the Xeon was used during tests. The operating system running on the CPU is a Linux kernel 3.2 with 64 bits support. The targeted GPU is an NVIDIA GeForce GTX660 TI with 5 compute units. The board's global memory consists of a 2 GB GDDR5 memory with 144 GB/s of bandwidth and an access from the host through a PCIe 3.0 connection with x16 lanes (theorical throughput of 985 MB/s per lane) [14]. Local memory on the GPU is made of on-chip L1 caches of 48 kB per compute unit.

The FPGA board is a Terasic DE4 based on a Stratix IV 4SGX530. Global memory is stored in two DDR2 memory banks, for a maximum theorical bandwidth of 12.75 GB/s to and from the FPGA (at a 400 MHz clock rate), and is accessible from the host through a PCIe gen2 4x connection. The PCIe connection has a maximum bandwidth of 500 MB/s per lane, meaning the DE4 board's maximum bandwidth is 2 GB/s. The

local memory is implemented through an interconnect structure that gives access to on-chip RAM blocks as simple dual port RAMs, running at 600 MHz. Specifically, M9K RAM blocks are used (RAM blocks of 256x36 bits). Private memory is implemented as flip-flops within the data flow and thus runs at the kernel's frequency. Each kernel implemented on the FPGA board was compiled with Quartus II 64 bits.

B. Hardware Resources Management and Utilization

All performance results were obtained according to the use case detailed in the Introduction (Section I), with a goal of less than 2000 options computed per second, and an available power of approximately 10W. The need for accuracy is met by representing all data in double precision and by choosing a discretization step of $T = 1024$. This provides a good compromise between speed, precision and hardware restrictions (in terms of memory resources). Further gain in efficiency could be achieved by manual fine tuning (i.e. custom data types), as seen in classic FPGA designs. We chose not to do so as it would not yield significant enough benefits compared with the necessary development time and would defeat the purpose of using the OpenCL standard.

Both kernels described in Section IV are detailed below. Table I shows compilation results for both kernels presented in Section IV, when implemented on the DE4 board. Those results were given by the Quartus II Fitter Summary as configured by default when running Altera's OpenCL Compiler, and Quartus Power Estimation tool (*quartus_pow*) to estimate the kernel's power consumption. In table I, the memory bits row covers M9K RAM blocks as well as M144K blocks (RAM block of 2048x72 bits). All results are given in a base 2 definition (i.e., $1K = 1024 = 2^{10}$).

Table I. RESOURCE USAGE

	Stratix IV EP4SGX530	
	Kernel IV.A	Kernel IV.B
Logic utilization	99 %	66 %
Registers	411 K/415 K	245 K/415 K
Memory bits	10,843 K/20,736 K (52 %)	7,990 K/20,736 K (39 %)
including M9K	1,250/1,250 (100 %)	1,118/1,280 (89 %)
DSP (18-bit)	586/1 K (59 %)	760/1 K (76 %)
Clock Frequency	98.27 MHz	162.62 MHz
Power consumption	15	17

Those kernels were parallelized using several options of Altera's OpenCL Compiler. First, compiler directives can be used to either replicate entire hardware pipelines or to vectorize the kernel execution. When replicating the pipeline, computations can be done independently from one another, while vectorization corresponds to an SIMD work division (Single Instruction, Multiple Data). From empiric observations, vectorization is usually a less resource-consuming optimization than replication. It also eases memory coalescing optimization. However, it is more constraining: vectorization can only be done by powers of two, and be a divider of the total work-group size. Besides, it is also possible to unroll any loop included in the kernel through *#pragma* directives. Loop unrolling uses less memory than a full replication, while giving another way to increase throughput and optimize resource consumption. Loop unrolling, replication and vectorization are 3 parameters that help reach the best compromise between resource utilization, latency and throughput. In our case, Kernel IV.A has been vectorized twice and replicated 3 times to use the maximum possible resources on the FPGA. Kernel IV.B contains an internal loop, which has been unrolled twice, coupled with a 4 times vectorization of the kernel. Both options of parallelization were chosen after several compilation iterations to find the best resource consumption rate.

It is interesting to note that, when optimized, both kernels use most of the M9K Block RAMs available, even though those blocks are used differently in the two kernels. Kernel IV.B implements its local memory as M9K blocks, while kernel IV.A uses those to coalesce its memory accesses to the global memory and store its inputs and outputs in shallow FIFOs. The kernels' power consumptions (15 and 17W) are upper bounds of the actual power consumption, and are resp. 50 and 70 % higher than the budget limit of 10W. They are an order of magnitude lower than the power consumption of the Xeon and the GTX660, with a respective Thermal Dissipation Power of 120 and 140W (see [14], [15], resp.). It is worth mentioning that the power consumption results of the FPGA designs do not encompass the DDR2 power consumption, nor any other part of the board but the FPGA. Still, the FPGA chip is responsible for most of the power consumption on the board, and those results represent a good approximation of the total power consumption of the DE4 board.

C. Performance Comparison

Table II illustrates the performances for each kernel on GPU and FPGA, along with software reference results, and published results for comparison [9], [10]. All the presented results were sampled after device saturation (best use case possible). After device saturation, computation time is a linear function of the number of computed options. This saturation typically happens at 10^5 priced options, which represents 5 plotted volatility curve (2000 options per volatility curve), which seems to be a realistic scenario. Only the kernel IV.B implemented on the GTX660 has a saturation at a higher number of options (10^6 options in both double and single precision). Higher throughput could be reached by increasing the implementation latency, but it would effectively rise the device saturation rate and reduce the accelerator's efficiency at lower workloads. As we consider an accelerator used by a single trader and not a shared resource (e.g., a server component), latency at low workload is an issue and must be minimized. The first implemented kernel presents extremely poor computing times, in both GPU and FPGA versions. This is due to memory copy from global memory to the host: one of the two ping pong buffers is fully read between each batch (approximately 19 MB for $N = 1024$), effectively stalling the kernel to avoid overwriting the results. A modified version of this kernel on GPU, with a reduced number of read operations between host and device, has an acceleration factor 14 times better than the initial kernel version on the same hardware (840 options/s vs 58.4 options/s). Modifications of this new version of the kernel to run on the DE4 board are ongoing, but the same order of magnitude of acceleration can be expected.

The kernel IV.B shows more promising results. More than 2000 options can be computed in less than a second (5150 options/s). Considering energy efficiency, the FPGA implementation is more than 5 times more energy efficient than the software reference. Unfortunately, this kernel does not reach the accuracy levels required for this application, with a RMSE (Root Mean Square Error) of 10^{-3} only. The same kernel implemented on GPU has no accuracy issues. The source of this inacurracy has been isolated and is due to the use

Table II. PERFORMANCES

	Kernel IV.A		Kernel IV.B			Reference Software		[9]	[10]
Platform	FPGA	GPU	FPGA	GPU	GPU	Single Core Xeon X5450		Virtex 4 xc4vsx55	Stratix III EP3SE260
Precision	Double	Double	Double	Single	Double	Single	Double	Double	Double
options/s	25	53	2400	47000	8900	116	222	385	1152
RMSE	0	0	$\sim 10^{-3}$	$\sim 10^{-3}$	0	$\sim 10^{-3}$	0	0	0
options/J	1.7	0.4	140	340	64	1	1.85	N/A	N/A
Tree nodes/s	13 M	30 M	1.3 G	25 G	4.7 G	61 M	117 M	202 M	576 M

of the *Power* operator. This operator shows an RMSE of 10^{-3}, compared with a software reference. The Power operator is not used within the kernel IV.A as the tree leaves are computed by the host and then tranfered to the device, contrary to kernel IV.B where the tree leaves are initialized in the device (a work-item for each tree leaf).

The performance gap in computation time between FPGA and GPU kernels comes from the high number of streaming multiprocessors that are present on this GPU. The GTX660 boasts a total of 960 stream processors (*CUDA cores*), with 1 double precision Arithmetic Logic Unit (*ALU*) per 8 stream processors (120 DP-ALUs) and running at 980 MHz. This outranks the processing capabilities of the FPGA board, but also means that the GPU board needs a more important workload to reach optimal performances (ten times as many). Besides, this increase in latency does not lead to much gain in acceleration factor, as the number of options/s computed by the GTX660 and the FPGA version are within a factor 5 of each other. Both computation times fall between expected results. This is illustrated by the number of options that can be computed with a single joule: the implementation on the DE4 board is 2 times more energy-efficient than the GPU implementation. Provided that the 13.0 SP1 of Altera's OpenCL compiler generates an accurate Power operator, the kernel IV.B on the DE4 board answers most of the constraints of our problem: pricing American options with high accuracy, at 2000 options/s or higher, and with minimal energy consumption. In case this issue is not solved with version 13.0 SP1, the values at the leaves will have to be computed on the host and sent to global memory, to be then copied in local memory, to the detriment of speed. The energy consumption still remains an issue, as the best current solution still needs more power than what is available. Workarounds exist, however. The best kernel implemented shows faster computation times than necessary; either clock frequency or paralellism levels can be lowered to reduce energy consumption. Besides, a less power consuming FPGA board can be selected that would better fit our goal. Finally, the DDR memory size used to implement global memory can be drastically reduced with no other impact on general performances but energy savings, as kernel IV.B use at best less than 100 KB of global memory during computation.

VI. CONCLUSION & FUTURE WORK

This paper presents two implementations of a binomial option pricing model applied to American options. The hardware target is a system comprising a CPU as host station, as well as an FPGA board to accelerate computations with minimal power consumption. The best OpenCL kernel is close to the performance levels desired, with more than 2000 options priced per second. The power that is used to achieve this computation time, 7W more than available, can be lowered to acceptable levels with a more appropriate target and by reducing the kernel frequency. Future work will focus on other hardware architectures supporting the OpenCL standard [16], [17], so as to compare their performances to the FPGA device and study the portability of the OpenCL kernel.

REFERENCES

[1] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang, "High-level synthesis for FPGAs: from prototyping to deployment," vol. 30, no. 4, pp. 473–491, Apr. 2011.

[2] J. Hull, *Options, Futures, & Other Derivatives*, ser. Prentice Hall finance series. Prentice Hall, 2009.

[3] J. C. Cox, S. A. Ross, and M. Rubinstein, "Option pricing: A simplified approach," *Journal of Financial Economics*, vol. 7, no. 3, pp. 229–263, Sep. 1979.

[4] C. de Schryver, I. Shcherbakov, F. Kienle, N. Wehn, H. Marxen, A. Kostiuk, and R. Korn, "An energy efficient FPGA accelerator for monte carlo option pricing with the heston model," in *2011 International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, Dec. 2011, pp. 468–474.

[5] E. Atanassov, S. Ivanovska, and D. Dimitrov, "Parallel implementation of option pricing methods on multiple GPUs," in *2012 Proceedings of the 35th International Convention MIPRO*, May 2012, pp. 368–373.

[6] D. Murakowski, W. Brouwer, and V. Natoli, "CUDA implementation of barrier option valuation with jump-diffusion process and brownian bridge," in *2010 IEEE Workshop on High Performance Computational Finance (WHPCF)*, Nov. 2010, pp. 1–4.

[7] R. Sridharan, G. Cooke, K. Hill, H. Lam, and A. George, "FPGA-Based reconfigurable computing for pricing multi-asset barrier options," in *2012 Symposium on Application Accelerators in High Performance Computing (SAAHPC)*, Jul. 2012, pp. 34–43.

[8] A. Tse, D. Thomas, K. Tsoi, and W. Luk, "Reconfigurable control variate monte-carlo designs for pricing exotic options," in *2010 International Conference on Field Programmable Logic and Applications (FPL)*, Sep. 2010, pp. 364–367.

[9] Q. Jin, D. B. Thomas, W. Luk, and B. Cope, "Exploring reconfigurable architectures for binomial-tree pricing models," in *Reconfigurable Computing: Architectures, Tools and Applications*. Springer, 2008, pp. 245–255.

[10] C. Wynnyk and M. Magdon-Ismail, "Pricing the american option using reconfigurable hardware." IEEE, 2009, pp. 532–536.

[11] M. Pharr and R. Fernando, *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (Gpu Gems)*. Addison-Wesley Professional, 2005.

[12] Q. Jin, W. Luk, and D. Thomas, "On comparing financial option price solvers on FPGA," in *2011 IEEE 19th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, May 2011, pp. 89–92.

[13] K. O. W. Group, "The opencl specification," 2011. [Online]. Available: http://www.khronos.org/registry/cl/specs/opencl-1.1.pdf

[14] "GeForce GTX 660 | Specifications | GeForce." [Online]. Available: http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-660/specifications

[15] "ARK | Intel®Xeon®Processor X5450 (12M Cache, 3.00GHz, 1333 MHz FSB)." [Online]. Available: http://ark.intel.com/products/34446/

[16] T. Flanagan, Z. Lin, and S. Narnakaje, "Accelerate multicore application development with keystone software."

[17] "Software Development Kit OpenCL™on ARM Linux." [Online]. Available: http://malideveloper.arm.com/develop-for-mali/sdks/mali-opencl-sdk/