FPGA Acceleration of Multi-Factor CDO Pricing

ALEXANDER KAGANOV
University of Toronto
ASIF LAKHANY
Algorithmics Incorporated
and
PAUL CHOW
University of Toronto

The last decade has seen a significant growth in the financial industry. The recent widespread use of Internet technology has increased the accessibility of the general population to financial data, thereby increasing the average portfolio size. This increase, compounded by the need for accurate real-time results, has led to a rising demand for faster risk simulations. Often accurately pricing widespread instruments, such as Collateralized Debt Obligations (CDOs), can take excessively long due to their Multi-Factor assets dependency. We present a hardware implementation for a Multi-Factor Gaussian Copula (MFGC) CDO pricing algorithm. Through a detailed benchmark exploration we demonstrate how reconfigurable hardware could be used to exploit fine-grain parallelism. Our results show that our implementation mapped onto a Xilinx Virtex 5 (XC5VSX50T) FPGA is over 71 times faster than corresponding software running on a single core 3.4 GHz Intel Xeon processor.

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

General Terms: Algorithms, Economics, Performance

Additional Key Words and Phrases: FPGA, Financial Modeling

1. INTRODUCTION

Over the past few years, the financial industry has seen a sharp increase in computational demand [Dorfman and D.Canning 2007]. Financial engineers typically attribute this increase to three distinct factors:

(1) Increasing model/contract complexity. Financial engineering is a relatively new and fast growing field. As the field evolves, newer and more complex models are developed to better represent real world scenarios [Haugh and Loe 2001]. Many of these models do not have a closed-form solution, where the entire problem could be represented in a single deterministic equation. In other cases the dataset may be too general to allow the application of more restrictive analytical methods that make oversimplifying assumptions about the portfolio characteristics. These cases

Author's address: A.Kaganov and P. Chow, Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON, Canada M5S 3G4 email:{kaganova,pc}@eecg.toronto.edu
A.Lakhany, Quantitative Research, Algorithmics Incorporated, Toronto, ON, Canada M5T 2C6 email: Asif@algorithmics.com

typically necessitate the use of Monte-Carlo (MC) methods. While MC methods are very generic and can be used to solve a large variety of equations, they are typically slow due to their $O(\frac{1}{\sqrt{N}})$ (where N is the number of MC paths) convergence rate [Glasserman 2004].

(2) Increasing portfolio size. As the financial market grows, more contracts are issued and a larger number of instruments are created. A prominent example of this increase can be seen in the derivative market, which is a subset of financial instruments that derive their value based on the market behavior of a other instruments. The amount of over-the-counter derivatives in G10 countries has almost doubled from $370 trillion in June 2006 to $683 trillion in June 2008 [Bank of International Settlements (BIS) 2008]. Consequently, even when using the exact same model, more than twice as much data needs to be processed.

(3) The need to make real-time decisions. The increases in model/contract complexity and portfolio size by themselves would not cause significant problems, if financial institutes were willing to wait longer to make decisions. However, in the financial world timing is critical. Market trends can change quickly and with most instruments traded electronically a small computational delay can result in a significant monetary loss.

While typical financial MC simulations are computationally intensive, they also tend to be highly parallel, with a large portion of the computation time spent in a small portion of the code. This has traditionally made them good candidates for hardware acceleration. Some groups have accelerated single option pricing [Baxter et al. 2007; McCool et al. 2006; Morris and Aubury 2007; Thomas et al. 2007], focused on an interest rate model [Zhang et al. 2005], Value at Risk (VaR) calculations [Thomas and Luk 2007], and presented an optimal asset allocation acceleration [Irturk et al. 2008]. To our knowledge we are the first to deal with structured instrument pricing. Structured instruments are typically backed by a whole portfolio of underlying assets and are created to transfer the risk associated with these assets using a different risk-profile than the portfolio itself.

Recently, the fastest growing structured instruments have been Collateralized Debt Obligations (CDO). In 2006 and 2007 CDO total global issuance reached $521 and $482 billion, respectively, which is over three times the 2004 issuance of $157 Billion [SIFMA 2008]. For a financial institute, a CDO provides a means of transferring the risk of owning volatile assets, such as mortgage loans, to the investors. To form a CDO, these assets are combined into a monetary Collateral Pool, which in turn is repackaged into different tranches (portions), each tranche covering a certain percentage of the monetary amount within the original pool. These tranches are then sold to investors in return for interest payments. The investor keeps receiving interest payments as long as there are no losses within the pool. However if a loss occurs, i.e. one of the loans defaults, the owners of the riskiest tranche start losing their invested principal. When the losses exceed the amount covered by the current tranche, the next tranche starts being affected.

A key component in CDO pricing is capturing the default dependence among the underlying assets, which could range from being independent to complete mutual dependence. A common way to model this dependence is through a number of mutual external stimuli, in such a way that highly correlated assets will depend on the same stimuli. For example, the price of Ford and Honda stock, major automotive manufacturing companies, might both depend on the global oil industry, the metal manufacturing industry, etc. For CDO pricing the two most popular models are the One-Factor Gaussian Copula (OFGC), which models the dependencies using only one common external stimuli, and the Multi-Factor Gaussian Copula (MFGC), a more general model with an arbitrary number of stimuli. Traditionally, the OGFC model is more common due to it's simplicity. However, it cannot fully capture the dependencies that exist among the portfolio instruments, often forcing financial engineers to resort to using the MFGC model [Glasserman and Suchintabandid 2007].

In this paper we extend the OFGC pricing core introduced in [Kaganov et al. 2008] to an MFGC implementation. Our contributions are:

— An optimized hardware implementation of the MFGC model, which exploits both coarse- and fine-grain parallelism inherit in the algorithm.

— A detailed benchmark exploration, highlighting the cases where the hardware could significantly outperform the software, and cases where the hardware advantage is small.

— A comparison against an optimized software implementation running on a 3.4 GHz Pentium Xeon single-core processor resulting in over 71-fold speedup using five hardware cores.

The rest of the paper is structured as follows. Section 2 describes the CDO structure and Li's [Li 2000] Gaussian Copula mathematical model. Section 3 presents the overall architecture and the MFGC core hardware implementation. Section 4 describes the test methodology. Section 5 reports the results of our implementation. Finally, section 6 summarizes the findings.

2. COLLATERALIZED DEBT OBLIGATION (CDO)

This section provides a brief overview of key CDO concepts: Subsection 2.1 presents an introduction to the CDO structure, and Subsection 2.2 describes the key equations in MC CDO pricing.

2.1 Structure

A Collateralized Debt Obligation usually consists of highly volatile obligation-based assets, such as: loans, bonds, and even other CDO tranches. Initially, a financial institute, termed *sponsor*, sells the obligation-based portfolio to an external organization, termed Special Purpose Vehicle (SPV), to isolate the investors from its own credit risk. The SPV combines all of the debt obligations into a single collateral pool. This pool is repackaged into different risk/profit CDO tranches, with each tranche covering a certain percentage of the monetary amount within the initial pool, as shown in Fig. 1.

Each CDO tranche is defined by an attachment and a detachment point. An attachment point is minimal monetary lose that can affect a tranche, and the detachment point is largest monetary lose still covered by a tranche. When the collateral pool loses exceed the attachment point, the investors in the tranche start to lose their principal. However, the investor continues to receive interest payments on the

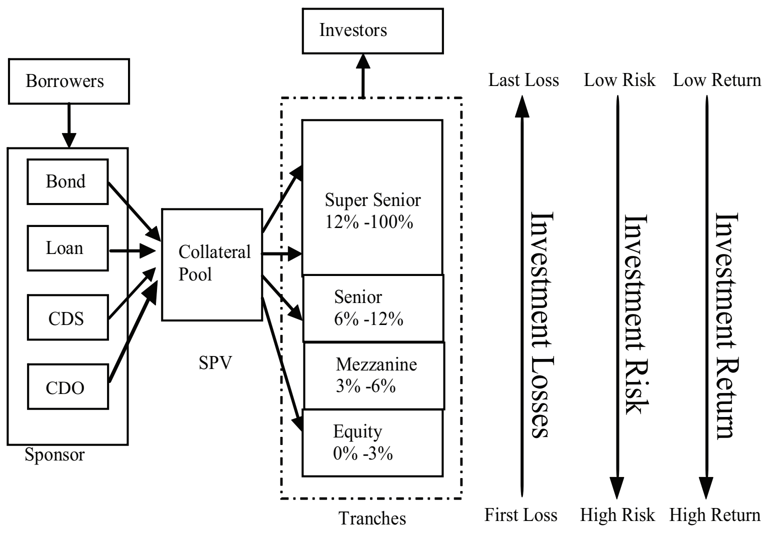


Fig. 1. Synthetic CDO Structure

remaining amount. For example, an investor invests \$900 in the Mezzanine, which has an attachment of 3% and a 6% detachment, and suddenly the pool losses reach 4% of the total pool. The investor will lose \$300, 1/3 of the original investment, but will keep receiving interest payments on the remaining \$600. When the cumulative losses reach the detachment point the tranche investors lose their entire investment and the tranche becomes inactive.

There are typically three motives for creating a CDO:

—Create market arbitrage: Generate money through the difference between tranche sales and premium payment on these tranches.

—Shrink balance sheet: Banks typically have regulations regarding the number of risky assets that they are allowed to hold.

—Risk Transfer: Sell the risk of owning a portfolio with debt assets to the investors [Goodman and Fabozzi 2002].

For a more complete CDO structure description refer to [J.Hall and White 2004].

2.2 Pricing

To issue tranches and establish the amount of interest an investor should receive, financial institutions have to determine each tranche's worth. To compute the value of a tranche one needs the following:

—A: Attachment point

—D: Detachment point

—s: Premium spread. Investment return rate.

—t_k: Premium dates. Dates at which investors receive interest payments, set relative to the initial CDO tranche issuance date. $0 < t_1 < t_2 < ... < t_K = T$, where T denotes the CDO maturity date

—d_k: Discount factor, present day worth of 1$ at time t_k
—$\hat{L}(t_k, \boldsymbol{x})$: Cumulative tranche losses up to the premium date t_k for a given market scenario $\mathbf{x}$.

In pricing a CDO tranche there are two important quantities termed Default Leg, Eqn. (1), and Premium Leg, Eqn. (2). The default leg measures the expected losses of the tranche over the life of the contract. In contrast, the premium leg measures the expected premium payments the tranche investor will receive over the life of the contract [K.Jackson et al. 2007].

$$\text{Default Leg} = E\left[\sum_{1}^{K}\left(\hat{L}(t_k) - \hat{L}(t_{k-1})\right)d_k\right], \tag{1}$$

$$\text{Premium Leg} = E\left[\sum_{1}^{K} s\left((D - A) - \hat{L}(t_k)\right)d_k\right], \tag{2}$$

By equating the two legs the break-even, "fair", premium spread s can be found. When evaluating Eqns. (1) and (2) the attachment/detachment points and the discount factors are fixed, which reduces the computationally intensive component to calculating $E[\hat{L}(t_k)]$, the expected tranche loss. For each of our datasets, refer to Section 4, expected tranche loss calculation takes over 99.8% of the total software runtime. Therefore, we choose to compute $E[\hat{L}(t_k)]$ using a Field Programmable Gate Array (FPGA), while the rest could be performed using a host processor.

2.3 Multi-Factor Gaussian Copula Model

In the financial world $E[\hat{L}(t_k)]$ is typically evaluated using Li's [Li 2000] MC based Gaussian Copula model for estimating collateral pool losses. This section highlights the major equations that correspond to key steps in the algorithm, numbered as *Steps*, which in turn will correspond to hardware blocks in Section 3.

Li's model assumes that the default probability for an individual asset follows an exponential distribution, with a default parameter λ_i. Hence, the probability of the i^{th} asset defaulting prior to time t_k is given by:

$$P\left(\tau_i < t_k\right) = 1 - \exp\left(-\lambda_i t_k\right), \tag{3}$$

To model the inter-asset default dependencies, Li introduced a creditworthiness index, Y_i, for each instrument given by:

Stage 1

$$Y_i = \sum_{j=1}^{m} \alpha_{i,j} X_j + \beta_i Z_i, \tag{4}$$

Where $\alpha_{i,j}$ and β_i are real positive constants such that $\sum_{j=1}^{m} \alpha_{i,j}^2 + \beta_i^2 = 1$. Z_i is the idiosyncratic factor, unique to each asset. $X_1, X_2, ..., X_m$ are systemic risk factors, used to model inter-asset dependencies. They represent external market stimuli that affect the entire portfolio. Highly correlated assets will have very similar $\alpha_{i,j}$ and β_i values, therefore an increase/decrease in a given systemic factor will affect the assets in the same manner. In fact, the correlation between instrument i and j can be expressed as $\sum_{k=1}^{m} \alpha_{i,k}\alpha_{j,k}$ [K.Jackson et al. 2007]. By setting $m = 1$ in Eqn. (4) the MFGC is reduced to an OFGC model, where the asset dependency is modeled through a single systemic factor. In the Gaussian Copula model the X_j's and Z_i are assumed to be independent zero mean unit variant Gaussian random variables.

The individual probabilities of default and the creditworthiness indexes are related by:

$$P\left(\tau_i < t_k\right) = P\left(Y_i < H_i(t_k)\right), \tag{5}$$

In the financial world $H_i(t_k)$ is called the default barrier of the i^{th} instrument at time t_k [K.Jackson et al. 2007]. Since, both X_j and Z_i follow "standard" normal distributions, Y_i is also normally distributed, from which it follows:

$$H_i\left(t_k\right) = \Phi^{-1}\left[P\left(\tau_i < t_k\right)\right], \tag{6}$$

where Φ is the normal cumulative distribution function.

At each path the overall collateral pool losses for a given time instance t_k and market scenario $\mathbf{x}$ $(X_1, X_2, ..., X_m)$ are:

Stage 2/3

$$L(t_k, \boldsymbol{x}) = \sum_{i=1}^{N} R_i I(Y_i(\boldsymbol{x}), t_k), \tag{7}$$

where R_i is recovery adjusted notional, the amount lost when asset i defaults (i.e. notional times one minus the recovery rate), and $I(Y, t_k)$ is the indicator function:

$$I(Y, t_k) = \left\{ \begin{array}{lll} 1 & Y < H_i\left(t_k\right) & 0 \leq t_k \leq T \\ 0 & \text{Otherwise} & \end{array} \right\}, \tag{8}$$

The tranche losses, $\hat{L}(t_k, \mathbf{x})$, can be derived from the pool losses using the attachment and detachment points. If the pool losses are below the attachment point there are no tranche losses; if the pool losses are above the detachment point the tranche losses are constant at the tranche width; between detachment and attachment points tranche losses are linearly proportional to pool losses, for reference refer to Fig 2:

Stage 4

$$\hat{L}(t_k, \boldsymbol{x}) = \min\left(D - A, \max\left(L(t_k, \boldsymbol{x}) - A, 0\right)\right), \tag{9}$$

The final expected value for the actual tranche loss is the average over all MC paths:

Stage 5

$$E[\hat{L}(t_k)] = E[\hat{L}(t_k, \boldsymbol{x})] \approx \frac{1}{\text{\# of Paths}} \sum_{j=1}^{\text{\# of Paths}} \hat{L}(t_k, \boldsymbol{x}), \tag{10}$$

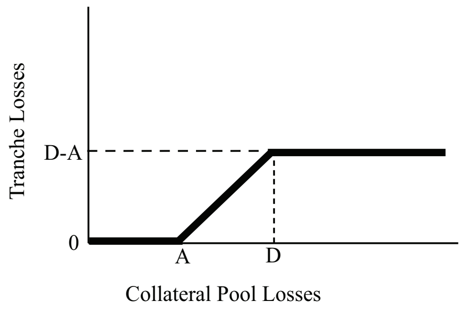


Fig. 2. Tranche Losses vs. Collateral Pool Losses

3. HARDWARE IMPLEMENTATION

This section presents the hardware implementation for a MFGC CDO pricing. Subsection 3.1 presents the overall multi-core pricing architecture. Subsection 3.2 goes into an in-depth description of the MFGC core design and presents the modifications made to the OFGC architecture [Kaganov et al. 2008] to allow multi-factor pricing.

3.1 Multi-Core Simulation Architecture

The hardware setup consists of a single FPGA (Virtex 5 SX50T) connected to a host processor through a single-lane PCI Express card, 250 MBytes/s bandwidth in both directions. The host processor is mostly idle, only controlling data transfer. In an actual financial system, the processor could be used for other pricing calculations, such as the Default and Premium Legs, Eqns. (1),(2). The FPGA itself performs the computationally intensive portion of computing $E[\hat{L}(t_k)]$.

The top level architecture plays three distinct roles: Input/Output (IO) management, data distribution and final data collection. The architecture, shown in Fig. 3, is designed to overlap data transfer and computation, which is achieved using memory double buffering by taking advantage of the dual-ported nature of Block RAMs (BRAMs). As discussed in Section 5.3, double buffering can completely hide transfer latency for our design given a data transfer bandwidth greater than 11.3MBytes/s.

The Distributor receives data from the host processor and loads local memory blocks within each CDO pricing core. Input data required for every simulation is:

—# of scenarios,

—# of instruments, N

—# of time steps, T

—# of default barriers, B

—# of system factors, F

—$\alpha_{i,j}$ (NxF matrix) and β_i (Nx1 vector)

—R_i (Nx1 vector), recovery adjusted notional

—$H_i(t_k)$ (BxT matrix), default barriers

—Ind_i (Nx1 vector), index vector

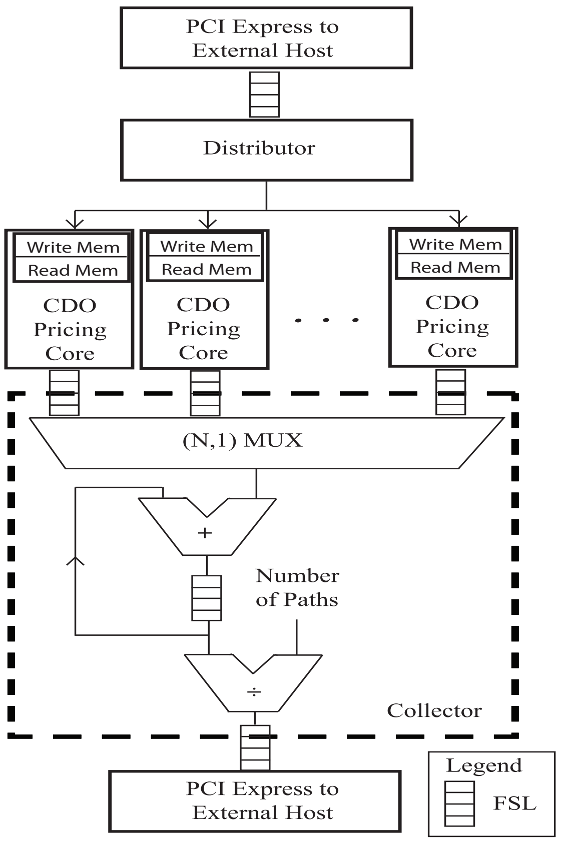


Fig. 3. Multi-Core Simulation Architecture.

The index vector,Ind_i, provides a mapping between instrument i and its default barrier. Many of the assets involved in the collateral pool have the same barrier curve, on average 4-5 curves per dataset [CDX 2008]. Therefore, providing an index vector, rather than replicating default barrier curve for each instrument, allows us to transfer significantly less data per simulation.

Since all MC paths are independent of each other [Glasserman 2004], the Distributor equally divides the paths amongst the CDO pricing cores. All CDO pricing cores to loaded simultaneously with the same data. The difference in the outputs stems from the different random numbers generated at each core. The number of pricing cores instantiated depends only on chip resources. The architecture supports an arbitrary number of cores. The performance and resource utilization is linearly proportional to the number of cores instantiated.

The final stage of the design is the Collector module. It combines the output from all the individual pricing cores, divides by the number of MC paths, and returns the results to the host processor. The Collector is decoupled from the pricing cores using Fast Simplex Links (FSLs) [Xilinx Inc. 2009], allowing the cores to start a new simulation, while the Collector is either still processing data or waiting for a

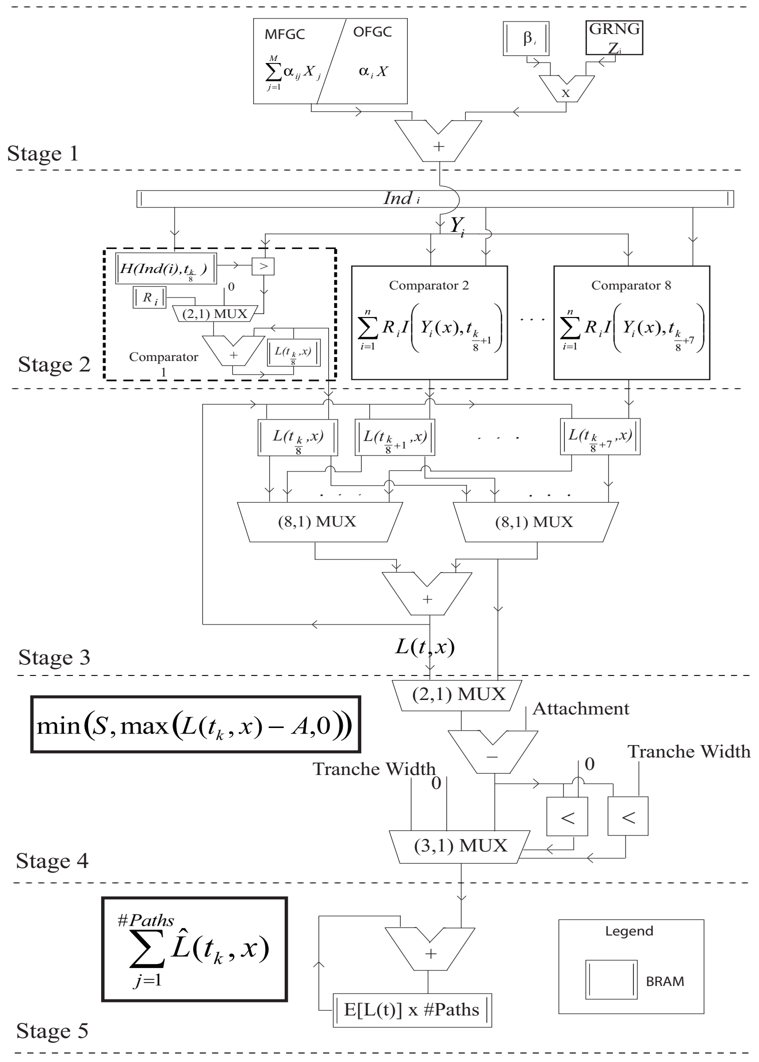

MFGC
$\sum_{j=1}^{M} \alpha_{ij} X_j$
OFGC
$\alpha_i X$
β_i
GRNG Z_i
Stage 1
Ind_i
Y_i
$H(Ind(i), t_{\frac{k}{8}})$
R_i
(2,1) MUX
Comparator 1
Comparator 2
$\sum_{i=1}^{n} R_i I\left(Y_i(x), t_{\frac{k}{8}+1}\right)$
Comparator 8
$\sum_{i=1}^{n} R_i I\left(Y_i(x), t_{\frac{k}{8}+7}\right)$
Stage 2
(8,1) MUX
(8,1) MUX
$L(t,x)$
Stage 3
$\min(S, \max(L(t_k,x) - A, 0))$
Attachment
Tranche Width
Tranche Width
(3,1) MUX
Stage 4
$\sum_{j=1}^{\#Paths} \hat{L}(t_k, x)$
E[L(t)] x #Paths
Legend
BRAM
Stage 5

Fig. 4. CDO Pricing Core Block Diagram

data transfer.

3.2 Multi-Factor Gaussian Copula Hardware

The asset correlation in the MFGC model is captured in Eqn. 4. To create a MFGC model, Eqn. 4 had to be incorporated into the CDO pricing engine presented in [Kaganov et al. 2008]. The single systemic factor computation at *Stage 1* has been replaced by a Factor Accumulation Module (FAM).

Fig. 4 shows a block diagram of the design. *Stage 1* demonstrates the two different equations modeled, the FAM and the Single Systemic factor in the MFGC and OFGC designs, respectively. In *Stages 2* and *3*, there are eight replicas of a

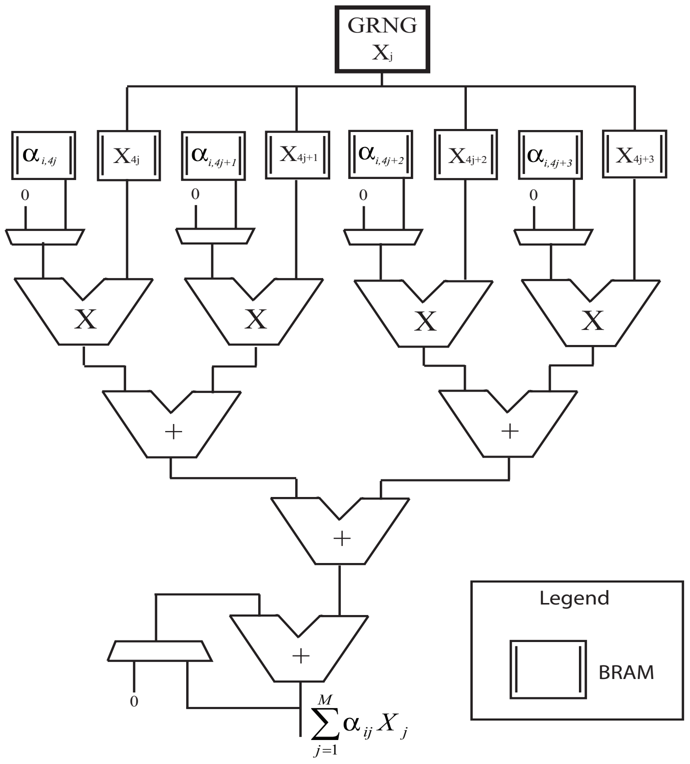


Fig. 5. Factor Accumulation Module

comparator, $Y_i < H_i(t_k)$ and Eqn. 7, respectively. This allows eight time steps to be processed simultaneously. It was found that eight replicas on average provided the best performance to chip resource trade-off for our benchmarks, refer to Subsection 4.1. *Stage 4* converts the pool loses to tranche losses, and *Stage 5* accumulates the losses over the number of MC paths.

The FAM, shown in Fig. 5, consists of eight separate memory blocks. Four hold the correlation factors, $\alpha_{i,j}$, and the other four hold the systemic factors, X_j, for the current MC path. The factors are generated using a variant of the Wallace Gaussian Random Number Generator (GRNG) [Lee et al. 2005]. To reduce latency, the idiosyncratic factor, Z_i, is generated by a second Wallace GRNG.

To minimize stalls, the GRNG generates systemic factors for the next path while the old ones are still being used in the calculations. The new systemic factors are written into double buffered BRAMs. In a given MC path the same system factors are reused for all instrument, and new ones are generated for a new MC path. The architecture allows for a maximum of four factors to be processed in parallel; a decision which was made based on a typical MFGC simulation that usually has three to four factors [Glasserman and Suchintabandid 2007]. The Wallace GRNG produces one random sample per cycle, while the rest of the module accumulates up to four factors per cycle. It takes n cycles to produce n Gaussian variants, and $\left\lceil \frac{\text{n x \# Instrument}}{4} \right\rceil$ to accumulate all the factors. Hence, to make sure the GRNG keeps up with systemic factor use, the simulation has to have at least as many instruments as factors, or a minimum of four instruments for $n > 4$.

Unlike, the OFGC model, which operates on single-precision floating-point Gaussian samples, the FAM uses 5-integer and 27-fractional bits fixed-point notation. The use of fixed-point notation allows for single-cycle addition, which in turn allows for a single-cycle final systemic factor accumulation, thereby reducing module latency.

One major difference between the OFGC and the MFGC designs is the interface between *Stages 1* and *Stages 2* . In the OFGC design a new creditworthiness index, Y_i, was generated as input to *Stage 2* of the design every-cycle. This is no longer possible in the MFGC design where it might take a few cycles to produce a Y_i. This can potentially cause *Stages 2* and beyond to stall. However, since *Stage 2* can only process up to eight time steps a cycle, a new Y_i is only needed every $\left\lceil \frac{\text{\# of Time Steps}}{8} \right\rceil$ cycles. While FAM can produce a new Y_i every $\left\lceil \frac{\text{\# of Systemic Factors}}{4} \right\rceil$ cycles. Hence, in certain cases there is no delay due to the multi-factor accumulation, otherwise *Stage 2* stalls for $\left\lceil \frac{\text{\# of Systemic Factors}}{4} \right\rceil - \left\lceil \frac{\text{\# of Time Steps}}{8} \right\rceil$ cycles.

4. TEST METHODOLOGY

4.1 Benchmarks

To measure performance nine benchmarks from [Kaganov et al. 2008] were used. They are listed in Table I. Benchmarks 1 through 8 are based on Dow Jones CDX indices, and have the same number of assets and credit rating distribution as the March 24, 2008 CDX indices [CDX 2008]. The notional amounts are based on Moody's corporate bond defaults for 1999. There is a wide range in notional sizes from \$0.6 million to \$6.6 billion. Benchmark 9 is added to represent a very large semi-homogeneous collateral pool of 400 assets, with only four different notional sizes: 20, 50, 100, and 200 million.

For all other input data:

—β_i: uniformly distributed from [0, 1].

—Return rate: normally distributed with a mean of 0.40 and 0.15 variance, [K.Jackson et al. 2007].

—Number of time steps: normally distributed with a mean of 20 steps and a variance of 10 steps.

—Each asset in the pool is randomly assigned a default barrier.

For each benchmark in Table I, 16 sub-benchmarks were generated with the number of systemic factors ranging from 1 to 16. Due to an artificial limitation of the current design, Benchmark 9 had a maximum of four system factors (a single BRAM contains a maximum of 512 32-bit entries resulting in maximum number of systemic factors of $\frac{4 \times 512}{\text{\# of Instruments}}$). However, since on-chip memory is not the

Table I. Test Benchmarks

#	Based on Data from	# of Assets	# of Time Steps	# Systemic Factors	Software Runtime (sec)
1	CDX.NA.HY	100	15	1-16	1.415 - 2.804
2	CDX.NA.IG	125	35	1-16	2.891 - 4.594
3	CDX.NA.IG.HVOL	30	19	1-16	0.494 - 0.940
4	CDX.NA.XO	35	22	1-16	0.635 - 1.151
5	CDX.EM	14	6	1-16	0.120 - 0.350
6	CDX.DIVIRSIFIED	40	23	1-16	0.729 - 1.306
7	CDX.NA.HY.BB	37	13	1-16	0.497 - 1.038
8	CDX.NA.HY.B	46	26	1-16	0.941 - 1.612
9	[K.Jackson et al. 2007] Semi-Homogeneous	400	24	1-4	7.236 - 8.510

limiting resource, in future designs a larger number of BRAMs could be utilized with minor design alterations, allowing for a larger number of systemic factors.

For each new factor sub-benchmark the new α_{ij} is generated from β_i in a recursive manner:

$$\alpha_{i,j} = U_j(1 - U_{j-1})...(1 - U_1)(1 - \beta_i^2), \tag{11}$$

where U_i is a uniform random number.

The software runtime column in Table I presents the empirically measured average run time for the smallest and the largest number of systemic factors, for a software setup described in Section 4.2.

4.2 Validation Methodology

All hardware designs were compared to an optimized double precision C++ software program (compiled using the gcc version 4.2.4 -O2 optimization flag), running on a single-core Pentium Xeon 3.4GHZ processor with 3GB RAM. A single core processor was chosen to allow a direct comparison with a single core hardware implementation, to demonstrate how FPGA exploits fine-grain parallelism. Both software and hardware can exploit the course-grain parallelism. Since MC paths are completely independent (and the amount of data for setup and reduction is relatively small), adding additional cores should create a linear speedup both in hardware, which we demonstrate Section 5.2.4, and software.

All hardware designs are written in Verilog and synthesized using Xilinx ISE 9.2. The hardware results were validated on a Xilinx ML506 Evaluation Platform, with a Virtex 5 SX50T -1 speed grade chip. The performance results in this paper are extrapolated to a faster -3 chip. To obtain the -3 speed grade performance values, the designs were placed and routed on a -3 speed grad chip and the execution times obtained on a -1 speed grade were multiplied by $\left(\frac{\text{Frequency -1 Speed Grade}}{\text{Frequency -3 Speed Grade}}\right)$.

To obtain performance and accuracy measurements each benchmark was run 100 times, each time with a different GRNG seed, for 100,000 MC paths on all designs. 100,000 MC paths provided a Standard Deviation below 0.1% across different GRNG seeds. The software and hardware results obtained with the same GRNG seed were compared. For accuracy measurements the final results were rounded up to a nearest cent, considering cents as the smallest meaningful entity. This allowed the hardware double-precision floating point to match the processor double precision, despite the Pentium Xeon processor using 80-bit Floating-Point registers.

For each benchmark, the computational error due to finite word-width effects is computed as:

$$\frac{1}{100}\sum_{1}^{100}\frac{|\text{Hardware Results for Run i} - \text{Software Results for Run i}|}{\text{Software Results for Run i}}, \qquad (12)$$

The final error reported is the benchmark computational errors averaged over the number of benchmarks and the maximal error is the largest observed benchmark error.

5. RESULTS

5.1 Resource Utilization

In the hardware designs three different recovery adjusted notional representations were explored: floating-point double-precision, floating-point single-precision, and integer. For the integer design: all the recovery adjusted notional values were multiplied by 100 to convert to cents (resulting in integer values). It was assumed that no financial transaction could contain values smaller than one cent and since Li's [Li 2000] algorithm just performs additions and subtractions on the adjusted notionals amounts, no smaller value could be generated. Table II summarizes the resource utilization for each design. The first percentages, next to the resource consumed count, indicates the portion used out of the total available. The second percentage, in bold, indicates the change compared to the corresponding OFGC design with the same notional representation [Kaganov et al. 2008].

As can be seen from Table II, the floating-point single-precision design uses almost half of the resources of the double-precision counterpart. However, compared to double-precision it has a 0.38% (maximal 1.10%) computational error. To achieve the resource utilization of the single-precision and the accuracy of double-precision designs, a hybrid design was created with a double-precision accumulator added, at *Stage 5*, to the single-precision design. This design significantly reduces the average computational error 10,000-fold, to 3.19E-5% (maximal 4.99E-5%), while keeping the resource utilization similar to single-precision. Furthermore, for our benchmarks, it was found that a 42-bit integer design with a 54-bit accumulator (at *Stage 5*) produced results identical to the double-precision floating point design, while using significantly less resources.

As seen from Table II, the MFGC design, compared to OFGC, uses between 9.1% and 19.3% more flip-flops, and between 3.4% and 10.5% more LUTs. The most significant increase is seen in terms of DSP48Es [Przybus 2008] usage, with an increase between 35%-48.3% for the floating point designs and a tripled DSP usage for the integer design. The majority of the new DSPs originate from the multipliers used in the factor accumulation module. The single-core frequency changes were very small and can probably be accredited to routing differences. However, when the number of cores was replicated to fill the entire chip the increase in the overall design size was evident as the overall frequency went down by approximately 7.2%.

Table II. Resource Utilization

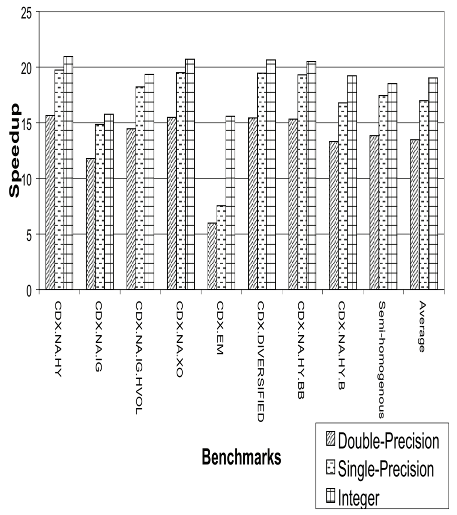
	Single-Precision Floating-Point	Double-Precision Floating-Point	Single-Precision Notionals & Double-Precision Accumulator	Integer
Flip-Flops	7205 (22.1%) **(+10.3%)**	10811 (33.1%) **(+9.1%)**	7564 (23.1%) **(+12.5%)**	5852 (17.9%) **(19.3%)**
LUTs	7347 (22.5%) **(+4.2%)**	13779 (42.2%) **(+3.4%)**	8037 (24.6%) **(+5.8%)**	5775 (16.7%) **(+10.5%)**
BRAMs	20 (15.2%) **(+25.0%)**	37 (28.0%) **(+19.4%)**	20 (15.2%) **(+25.0%)**	20 (15.2%) **(+25.0%)**
DSP48Es	43 (14.9%) **(+48.3%)**	54 (18.8%) **(+35.0%)**	44 (15.3%) **(+46.7%)**	21 (7.3%) **(+200.0%)**
Freq (MHz)	246.9 **(-0.8%)**	195.9 **(+2.6%)**	244.5 **(-0.1%)**	262.1 **(-2.2%)**
Average Error (%) [Max Error]	0.38 [1.10]	0	3.19E-5 [4.99E-5]	0
Maximum # of Cores	4	2	4	5
Replicated Freq (MHz)	195.1 **(-6.4%)**	138.6 **(-1.6%)**	185.8 **(-11.5%)**	198.1 **(-9.3%)**

5.2 Performance

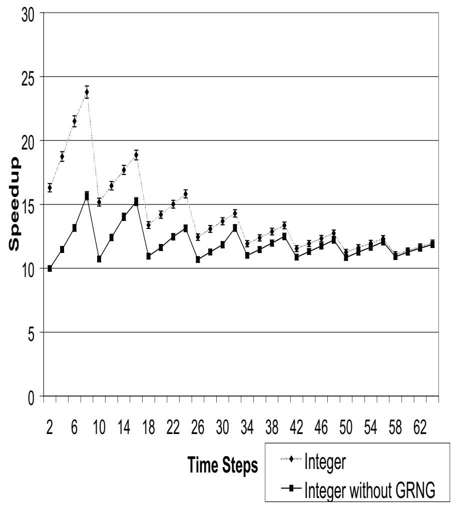
5.2.1 *Single Core.* Fig. 6(a) depicts the single core performance results for the nine benchmarks averaged over 16 Systemic factors. The floating-point hybrid design was grouped together with the single-precision design, since both have the same cycle-by-cycle behavior and operate at almost identical maximum frequency. As can be seen from the figure the average speedup across the benchmarks varies between 6 and 21, with the average speedup for double-precision, single-precision and integer designs being 13.5, 17.0, and 19, respectively. The difference between the single-precision and double-precision acceleration is mainly due to the frequency difference between the two designs. However, in addition to a frequency advantage the integer design has a significantly shorter pipeline, which is especially evident for Benchmark 5 (CDX.EM). The CDX.EM is the smallest benchmark with only 14 instruments. This causes all designs to stall between *Stages 2* and *3* as the partial sums are combined. However, due to a shorter pipeline the integer design creates the least number of partial sums, and due to a single-cycle integer adder it takes the least time to add up the sums. Therefore, the integer design stalls for the least number of cycles, creating a significantly larger acceleration compared to the floating-point designs, 15.5-fold compared to 6- to 7-fold.

To examine the differences in acceleration between the benchmarks, the effects of the number of time steps, instruments, and systemic factors have on acceleration are explored next.

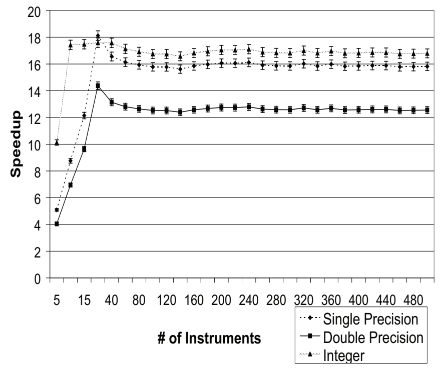
5.2.2 *Performance versus Time Steps.* Fig. 6(b) presents the effect of time steps on acceleration for Benchmark 9 as the number of instruments and number of systemic factors was kept constant at 400 and 1, respectively, while the number



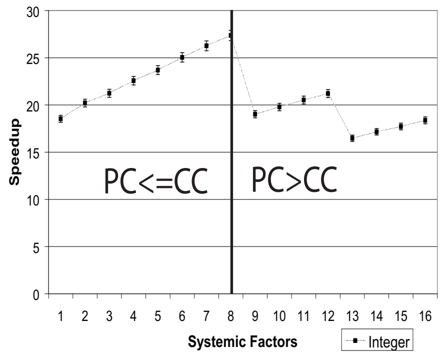
(a) Single-Core Performance



(b) Performance vs. Time Steps



(c) Performance vs. Number of Instruments



(d) Performance vs. Systemic Factors

Fig. 6. Performance Results

of time steps was varied from 2 to 64. One can see from Fig. 6(b) that as the number of time steps is closer to a multiple of eight the larger the acceleration. The number of time steps dictates the number of comparisons done at *Stage 2* of the design. While in software these comparisons are done sequentially, in hardware eight comparisons are done in parallel. Hence, in software it would take longer to price 8n+7 time steps than 8n+1 (where $n > 0$ is an integer), while in hardware it takes the same amount of time.

There is also a more general trend; speedup decreases as the number of time steps increase. We believe that the majority of this behavior is due to the hardware GRNG. As the number of time steps increases a smaller portion of the software execution time is dependent on the GRNG and hence less advantage can be taken from having single-cycle hardware GRNG. The second curve in Fig. 6(b) shows the hardware design compared to the software run-time without the time required to produce Gaussian samples. The two curves exhibit the same characteristics. However, without accounting for GRNG time there is no decrease in speedup over the number of time steps. Hence, for a small number of time steps, the hardware GRNG alone provides a significant speedup, 16.3-fold versus 10.0-fold for two time steps, but has a relatively small impact when the number of time steps is large, 12.0-fold compared to 11.9-fold for 64 time steps.

5.2.3 *Performance versus Instruments.* To test acceleration compared to the number of instruments the time steps and systemic factors were kept constant at 24 and 1, respectively, while the number of instruments for Benchmark 9 was varied from 4 to 512 (512 being the largest possible due to current design restrictions), Fig. 6(c). One can see that speedup does not change as the number of instruments is increased, indicating that in terms of instruments, the hardware design is as scalable as software. However, when the number of instruments falls below 20, the floating-point designs start stalling and the performance starts to drop. The same happens to the integer design when the number of instruments falls below 10. Hence, to obtain sizable performance improvement, the collateral pool should contain at least ten instruments.

5.2.4 *Performance versus Systemic Factors.* Fig. 6(d) is a performance plot for benchmark 1 (CDX.NA.HY), as the number of systemic factors is varied from 1 to 16 (all other benchmarks follow a similar trend). To examine the plot let us define two quantities: Consumer Cycles (CC), which is the number of cycles before *Stage 2* in Fig. 4 requires a new Y_i, and Producer Cycles (PC), which is the number of cycles it takes to produce a new Y_i. CC is given by:

$$\mathrm{CC} = \left\lceil \frac{\#\text{ of Time Steps}}{8} \right\rceil, \tag{13}$$

and PC is given by:

$$\mathrm{PC} = \left\lceil \frac{\#\text{ of System Factors}}{4} \right\rceil, \tag{14}$$

Fig. 6(d) can be divided into two regions: PC <= CC, and PC > CC. For the region in which PC <= CC the speedup increases linearly with systemic factors. In this region increasing the number of systemic factors increases the number of calculations performed in a software program, thereby increasing the software runtime. While in hardware, Y_i generation is done in parallel to Y_i consumption, hence while PC <= CC a new Y_i is generated before it's required. This results in a constant hardware runtime with respect to an increase in the number of systemic factors. In the PC > CC region, the speedup follows a step-like curve. Since FAM processes four factors at a time, the closer the number of systemic factors to the next multiple of four, the larger the acceleration.

For each precision representation the number of pricing cores was replicated to the extent permitted by the FPGA resources. Since MC paths are completely independent, adding additional pricing cores creates a linear speedup. Using a Virtex-5 SXT50, the double-precision design was replicated twice, single-precision and hy-

Table III. Performance Summary

	# of Cores	Frequency (MHz)	PC<CC (Speedup)	PC=CC (Speedup)	PC>CC (Speedup)	Average (Speedup)
Software (Intel Xeon)	1	3400	1	1	1	1
Double-Precision	2	138.6	18.7	22.5	16.9	19.0
Single-Precision	4	195.1	52.8	63.4	47.7	53.6
Hybrid	4	185.8	50.3	60.4	45.4	51.1
Integer	5	198.1	67.0	84.5	70.1	**71.5**

brid designs four times, and the integer design five times. Table III summarizes the results across all benchmarks based on three regions PC<CC, PC=CC, and PC>CC. As one would expect the highest speedup is obtained in the PC=CC region, which is on average 84.5 (in general the highest acceleration was 103.4 seen in CDX.NA.HY for eight systemic factors). However, the average acceleration for five integer cores is 71.5.

5.3 IO Requirements

To maintain the CDO simulation engine fully utilized and obtain the speedups mentioned in the sections above, the IO device has to provide sufficient bandwidth to complete data transfer neaded for the next simulation while the previous simulation is still being priced. The worst case scenario, which is the one that requires the fastest data transfer, occurs when a simulation with the shortest computation time is followed by one with the largest input dataset. In such a case to avoid stalls the large input dataset has to be transferred within the computation time of the short simulation.

Examining computational times across all designs and benchmarks, it was found that one systemic-factor Benchmark 5 (CDX.EM), running on five integer cores had the shortest computational time of 2.93 ms. The largest dataset was the four systemic-factor Benchmark 9 (semi-homogeneous), which requires 104 Kbits. Using these worst-case empirical numbers it was found that for our benchmarks a 35.5 Mbits/s (4.4 MBytes/s) transfer rate would be sufficient to keep all pricing cores fully utilized. Going further, based on the current design memory utilization, the user could provide a maximum of 262 Kilobits. For this theoretical maximum dataset, assuming 2.93 ms computational time, the required transfer rate is 89.4 Mbits/s (11.2 MBytes/s), which is still well within the 250 MBytes/s available with a 1-lane PCI-express card.

6. CONCLUSION

In this paper we presented a hardware implementation of Li's Multi-Factor Gaussian Copula model for synthetic CDO, which allows pricing a portfolio with an arbitrary asset dependency structure. Through our benchmark exploration we demonstrated how reconfigurable hardware allows the designer to exploit fine grain parallelism.

We illustrated the advantages of replicating modules that cause data-flow bottlenecks, seen in the time step exploration. The factor accumulation module demonstrates the advantage of parallel execution, where the module execution time can completely be hidden. In addition, we demonstrated how customizing bit width based on the dataset and the general data-flow allows us to create an integer design that matches the accuracy of a double precession floating point design, for a fraction of the resource utilization.

As the need for faster pricing simulations keeps growing in the financial world, the next natural question becomes which technology is most suitable for these applications. In presenting a technology, we believe it's vital to demonstrate the performance over a large set of representative test-cases, explain which techniques have been used to obtain the acceleration, and present the trade-offs; as we have done in this paper. This allows the financial engineers to make an informed decision and ultimately teaches the generic techniques that they could eventually apply to their own designs while using this technology, which is the ultimate goal.

REFERENCES

Bank of International Settlements (BIS). 2008. Amounts outstanding of over-the-counter (OTC) derivatives. `http://www.bis.org/statistics/otcder/dt1920a.pdf`. Accessed: August 15, 2009.

Baxter, R., Booth, S., and et. al. 2007. Maxwell - a 64 FPGA Supercomputer. in Adaptive Hardware and Systems. In *AHS 2007. Second NASA/ESA Conference on,*. 287–294.

CDX. 2008. Markit CDX Indecies. `http://www.markit.com`. Accessed: March 24, 2008.

Dorfman, D. and D.Canning. 2007. The Actuary's High-Performance Computing Challenge. *Windows in Financial Services*.

Glasserman, P. 2004. *Monte Carlo Methods in Financial Engineering.* Springer.

Glasserman, P. and Suchintabandid, S. 2007. Correlation expansions for CDO pricing. *Journal of Banking & Finance 5*, 1375–1398.

Goodman, L. and Fabozzi, F. 2002. *Collateralized Debt Obligations.* John Wiley & Sons.

Haugh, M. and Loe, A. 2001. Computational Challenges in Portfolio Management. *Computing in Science & Engineering 3,* 8 (may/june), 54–59.

Irturk, A., Benson, B., Laptev, N., and Kastner, R. 2008. FPGA acceleration of mean variance framework for optimal asset allocation. In *Workshoop on High Performance Computational Finance, 2008. WHPCF 2008.* 1–8.

J.Hall and White, A. 2004. Valuation of a CDO and an nth to Default CDS Without Monte Carlo Simulation. *Journal of Derivatives 12,* 2 (September), 8–23.

Kaganov, A., Lakhany, A., and P.Chow. 2008. FPGA Acceleration of Monte Carlo Based Credit Derivative Pricing. In *International Conference on Field Programmable Logic and Applications.* 329–334.

K.Jackson, Kreinin, A., and Ma, X. 2007. Loss Distribution Evaluation for Synthetic CDOs. `http://www.defaultrisk.com/pp_cdo_14.htm`. Accessed: August 15, 2009.

Lee, D., Luk, W., Villasenor, J., Zhang, G., and Leong, P. 2005. A Hardware Gaussian Noise Generator Using the Wallace Method. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems 13,* 8 (August), 911–920.

Li, D. 2000. On Default Correlation: A Copula Function Approach. *The Journal of Fixed Income 9*, 43–54.

McCool, M., Wadleigh, K., Henderson, B., and Lin, H. 2006. Performance Evaluation of GPUs Using RapidMind Development Platform. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing.*

Morris, G. and Aubury, M. 2007. Design Space Exploration of the European Option Benchmark using Hyperstreams. In *Field Programmable Logic and Applications, 2007. FPL 2007.* 5 –10.

Przybus, B. 2008. The Virtex-5 SXT Option for High-Performance Digital Signal Processing. `http://www.xilinx.com/publications/xcellonline/xcell_62/xc_pdf/p34-36_62-virtex.pdf`. Accessed: August 15, 2009.

SIFMA. 2008. Global Market Issuance Data. `http:/www.sifma.org`. Accessed: August 15, 2009.

Thomas, D., Bower, J., and Luk, W. 2007. Automatic Generation and Optimization of Reconfigurable Financial Monte-Carlo Simulations. In *Application -specific Systems, Architectures and Processors, 2007*. 168–173.

Thomas, D. and Luk, W. 2007. Sampling from the Multivariate Gaussian Distribution using Reconfigurable Hardware. In *Field-Programmable Custom Computing Machines, 2007. FCCM 2007*. 3–12.

Xilinx Inc. 2009. Fast Simplex Link (FSL) Bus (v2.11b). `http://www.xilinx.com/support/documentation/ip_documentation/fsl_v20.pdf`. Accessed: August 15, 2009.

Zhang, G., Leong, P., Ho, C., Tsoi, K., Cheung, C., Lee, D., Cheung, R., and Luk, W. 2005. Reconfigurable acceleration for Monte Carlo based financial simulation. In *Field-Programmable Technology, 2005. Proceedings. 2005*. 215–222.

Received August 2009; revised February 2010; accepted March 2010