FPGA ACCELERATION OF MONTE-CARLO BASED CREDIT DERIVATIVE PRICING

Alexander Kaganov, Paul Chow

Asif Lakhany

Department of Electrical and Computer Engineering University of Toronto
Toronto, ON, Canada M5S 3G4
email: { kaganova, pc }@eecg.toronto.edu

Quantitative Research Algorithmics Incorporated Toronto, ON, Canada M5T 2C6 email: Asif@algorithmics.com

ABSTRACT

In recent years the financial world has seen an increasing demand for faster risk simulations, driven by growth in client portfolios. Traditionally many financial models employ Monte-Carlo simulation, which can take excessively long to compute in software. This paper describes a hardware implementation for Collateralized Debt Obligations (CDOs) pricing, using the One-Factor Gaussian Copula (OFGC) model. We explore the precision requirements and the resulting resource utilization for each number representation. Our results show that our hardware implementation mapped onto a Xilinx XC5VSX50T is over 63 times faster than a software implementation running on a 3.4 GHz Intel Xeon processor.

1. INTRODUCTION

In the past few years there has been a growing demand for computationally intensive financial calculations. This demand can generally be attributed to the increasing number of financial instruments within a client's portfolio and the ever present need to make real-time decisions. Recently, one of the fastest growing instruments has been Collateralized Debt Obligations (CDOs). Their total global issuance has more than tripled from US\$157 Billion in 2004 to US\$552 Billion in 2006, and despite the recent sub-prime US mortgage crisis, the 2007 issuance still surpassed US\$485 Billion [1].

The mechanism behind a CDO allows financial institutions to mitigate the dangers of owning a portfolio with high risk debt assets (such as sub-prime mortgage loans) by selling the risk to investors. In a typical CDO, multiple assets are combined into a Collateral Pool, which is repackaged into different risk/profit CDO tranches, with each tranche covering a certain percentage of the monetary amount within the pool, and the tranches are 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 investors that own the riskiest tranche start losing their invested principal. When the losses exceed the amount cov-

ered by the current tranche, the next riskiest tranche starts being affected.

This process of reselling debt has proven to be an efficient way for a bank to transfer credit risk to the investors, generate money through tranche sales, and shrink its own balance sheet. A critical component in the process is being able to accurately price the cost of a given tranche in realtime, i.e., predicting how many assets will default by a given time interval. Over the recent years, multiple models have been proposed for CDO pricing. These models commonly try to leverage between accuracy and speed. The models can be divided into two categories: more generic slower (in software) Monte-Carlo models [2] and faster more restrictive analytical models [3]. Despite the speedups provided by analytical models, the Monte-Carlo pricing remains widely in use due to its flexibility and applicability to a general CDO portfolio without having to make any assumptions regarding the data. One of the most widely used Monte-Carlo models, due to its simplicity and flexibility, is the One-Factor Gaussian Copula (OFGC) first introduced by Li [2].

Previous works in hardware acceleration of financial simulation have focused on single option pricing [4][5], interest rates [6], and Value-at-Risk simulations [7]. All these works have focused on pricing individual instruments. To our knowledge we are the first to attempt credit derivative pricing, which requires a different model to calculate overall losses within a portfolio.

In this paper, we propose a hardware implementation of Li's model, which provides a significant speedup over the software implementation by exploiting fine-grain parallelism within the model. Our main contributions are:

- A simulation architecture that allows simultaneous offchip data transfers and computations,
- A pipelined hardware implementation of the OFGC model,
- A detailed examination of the precision requirements for the data and the resulting resource utilization,

 A comparison between the software implementation running on a 3.4 GHz Pentium Xeon processor and five fixed-point cores running on a Virtex 5 XC5SX50T chip, resulting, on average, in over 63-fold speedup.

The paper structure is as follows. Section 2 provides a detailed description of a CDO mechanism. Section 3 presents the OFGC model. Section 4 describes the hardware implementation. Section 5 reports the results of our implementation. Section 6 summarizes our results.

2. COLLATERALIZED DEBT OBLIGATION

A typical financial company can own a variety of risky debt obligations as part of its asset portfolio, such as: bonds, loans, credit default swaps (CDS), and even CDOs. To mitigate the risk associated with debt obligations the financial company, termed the *sponsor*, creates a separate entity called a Special Purpose Vehicle (SPV), to isolate CDO investors from its own credit risk. The sponsor then either sells the actual debt obligations to the SPV, or just the risk associated with them, while the actual assets stay with the sponsor.

The SPV groups all the debt obligations into a Collateral Pool and issues tranches to the investors as shown in Fig. 1. Each tranche has an attachment and a detachment point. When the cumulative losses in the Collateral Pool exceed the attachment point of a given tranche, the investors in the tranche start to lose their principal, and when the cumulative losses reach the detachment point, the investors in the tranche lose their entire investment. However, for the lifetime of the tranche, the investor receives interest payments on the remaining principle [8].

Each tranche has a different risk factor. As can be seen from Fig. 1, the Equity tranche, with the 0% attachment point, is the riskiest tranche, while the Super Senior tranche, with the 12% attachment point, is the safest. Using the attachment and detachment points alongside the expected pool losses, each tranche can be priced.

3. ONE-FACTOR GAUSSIAN CUPOLA MODEL

In 2000, Li [2] introduced a Gaussian Cupola model for estimating Collateral Pool losses. The flexibility and the simplicity of the model established it as one of the most prominent methods of pricing CDOs. However, its main drawback is that it uses Monte-Carlo paths to calculate expected losses, which could be time consuming on a typical PC. However, the model contains a high degree of parallelism that can be exploited in hardware to attain a significant speedup.

For a given tranche the problem definition is:

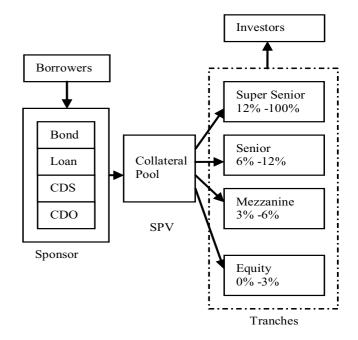


Fig. 1. Collateralized Debt Obligation Structure.

A: Attachment point

D: Detachment point

 N_i : Recovery adjusted notional, which is the monetary amount a financial institution can recover in the case the i^{th} asset defaults

 α_i : Correlation factor between the state of global market and asset i

$$\beta_i$$
: $\sqrt{1-\alpha_i^2}$

 τ_i : The time at which asset i defaults

In a pool of n assets, Li proposed modelling default probabilities using a Poisson process with a parameter λ_i . The probability of the i^{th} asset defaulting prior to time t becomes:

$$P(\tau_i < t) = 1 - \exp(-\lambda_i t), \tag{1}$$

The curve $P(\tau_i < t)$ is known as a default boundary curve for asset i. Furthermore, the model assumes that the default probabilities relate to a random variable Y_i by:

$$P(\tau_i < t) = P(Y_i < y(t)), \tag{2}$$

Where Y_i is:

$$Y_i = \alpha_i X + \beta_i Z_i, \tag{3}$$

in which both X and Z_i are zero mean unit variant Gaussian random numbers. X is a systemic factor that represents the current market condition and is constant for all assets in the pool for a given Monte-Carlo path. Z_i is the idiosyncratic factor, unique to each asset.

Since, both X and Z_i follow standard normal distributions, Y_i is also normally distributed. It follows that:

$$y(t) = \Phi^{-1}[P(\tau_i < t)],$$
 (4)

where Φ is the standard normal cumulative distribution function

Combining Eqns. (4), (3), and (2), and conditioning on the market state X = x, the default probability becomes:

$$P[\alpha_i x + \beta_i Z_i < \Phi^{-1}(P(\tau_i < t))]$$

$$= \Phi^{-1}(\frac{\Phi^{-1}(P(\tau_i < t)) - \alpha_i x}{\beta_i}),$$
(5)

Eqn. (5) is the equivalent of searching for the intersection point between Y_i and the default boundary curve in each Monte-Carlo path.

For each Monte-Carlo path, the overall pool losses for a given time instance t, are:

$$L(t) = \sum_{i=1}^{n} N_i \times I_i(Y_i, t), \tag{6}$$

where I_i is the indicator function:

$$I(Y,t) = \left\{ \begin{array}{ll} 1 & Y < P(\tau < t_k) & t_0 \le t_k \le t \\ 0 & Otherwise \end{array} \right\}, \quad (7)$$

A tranche can only start sustaining losses when the total loss exceeds A, and only cover up to D-A losses, which gives the following tranche loss equation for a single Monte-Carlo path:

$$\hat{L}(t) = \min(\max(L(t) - A, 0), D - A),$$
 (8)

The expected value for the actual tranche loss is the average of all Monte-Carlo paths:

$$E[\hat{L}(t)] = \frac{1}{\#Paths} \sum_{j=1}^{\#Paths} \hat{L}_j(t),$$
 (9)

4. HARDWARE IMPLEMENTATION

In this section we present a multi-OFGC core simulation architecture, as well as the hardware implementation of the one-single factor Gaussian Copula model.

4.1. Simulation Architecture

Top-level parallelization is performed over the Monte-Carlo paths, since all Monte-Carlo paths are independent of each other. The paths are equally divided amongst the OFGC cores. Similarly, path independency makes it easier to distribute input data. All OFGC cores are loaded simultaneously with the same data. The difference in the outputs of the cores stems from the different Gaussian values generated at each Monte-Carlo path.

The simulation architecture is designed to perform multiple tasks in parallel. The design is broken into three separate stages as shown in Fig. 2: distributor, independent

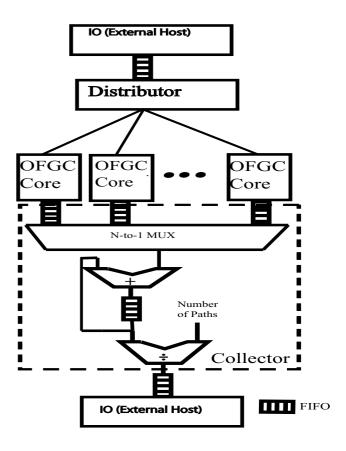


Fig. 2. Multi-Core Simulation Architecture.

OFGC cores, and a collector. The addition of separate distributor and collector cores allows the OFGC cores to be kept active at all times.

The distributor core uses double buffering, achieved using dual-ported Block RAMs, to hide the latency of loading data onto the Field Programmable Gate Array (FPGA). We have established, based on our benchmarks, that we can keep the accelerator fully active. We calculated the theoretical maximum number of transfer bits by taking the largest pool size and number of time steps within our data, 400 and 35 respectively, and assuming each asset has its own default curve (a theoretical maximum that is significantly larger than what our data would indicate). The worst case scenario number of bits is found to be 512 Kbits (actual maximum within our benchmarks was 66 Kbits), while the shortest calculation takes 2.66 ms, hence a data transfer rate over 192Mbits/s with the host will be sufficient to keep all OFGC cores busy.

The Collector core, shown in Fig. 2, is decoupled from the individual OFGC cores through FIFOs. This allows the OFGC cores to start a new simulation while the collector core finds the average tranche losses over all Monte-Carlo paths and sends the results to the host.

4.2. One-Factor Gauusian Cupola Hardware

A fully pipelined design of the One Factor Gaussian Cupola model is presented in Fig. 3.

In Stage 1, two Gaussian Random Number Generators (GRNG) are used to generate X and Z_i , and Y_i is created based on Eqn. (3).

In Stage 2, there are eight replicas of a comparator core that implement Eqns. (5), (6) and (7). Each replica performs the $Y_i < P(\tau_i < t)$ comparison for a subset of t_k s, assigned in a sequential mod eight manner. These comparisons themselves are independent and hence can be performed in parallel. The decision to select eight replicas is based on convenience and resource conservation. We define a Replication Utilization Factor, RUF:

$$RUF = t \operatorname{mod}(\#ofreplicas),$$
 (10)

where t is the total number of time steps in a simulation. More replicas potentially provide a greater speedup when RUF is approximately equal to the number of replicas; however, the overall design grows large and many of the comparator units become underutilized when RUF is about 0. Eight is chosen as a convenient power of 2, making partitioning as well as arithmetic operations in the control path more efficient, and provides a good speedup and low utilization cost for a t that is normally distributed with a mean of 20, the theoretically ideal value [3].

In Stage 2, the Block RAM (BRAM) stores multiple partial sums for each $L(t_k)$. This is done to avoid stalling the pipeline. The adders at Stage 2 have a pipeline latency that often create a situation where the value of $L(t_k)$ is needed at the input to the adder while it is still being computed. Hence, one of the partial sums is used instead. The greater the adder latency the more partial sums are in flight. The downside of this approach is that at the end of each Monte-Carlo path these partial sums have to be combined to form the total number of Collateral Pool losses at a given time step.

In Stage 3, the partial sums are combined. Once all partial sums are available in the comparator cores their values are transferred to the temporary memory storage. This allows all stages above Stage 3 to start a new Monte-Carlo path. If the number of assets in the pool or the number of time steps is sufficiently large, combining partial sums and creating new ones can be done in parallel. However, if the new partial sums are ready before the previous ones have been combined the pipeline has to stall. This is only seen once with our smallest benchmark, Benchmark 5, which contains a pool of only 14 assets and is simulated for only six time steps requiring eight partial sums for each time step, which is the maximum possible value in our design.

Stage 4 is the hardware representation of Eqn.(8). It takes the total pool losses for a given time step and calculates the losses within the currently simulated tranche.

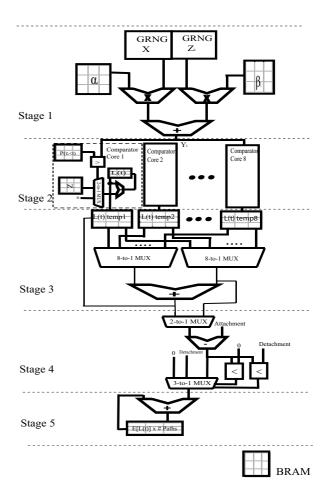


Fig. 3. One-Factor Gaussian Cupola Hardware Core.

Stage 5 is the final accumulator, which combines the tranche losses over all Monte-Carlo paths.

5. RESULTS

In this section we examine the resource utilization and the speedup obtained for different precision representations for rate adjusted notionals. Since no widely accepted benchmarks exist and all financial transactions are confidential, we developed our own benchmarks based on the Dow Jones CDX indices [9] and publicly available Moody's rating information [10].

5.1. Benchmarks

Nine different test benchmarks are constructed and shown in Table 1. The first eight are based on Dow Jones CDX indices, which are commercially traded CDO-like instruments that are based on collateral pools consisting of companies and government organizations in North America and emerging markets. Benchmarks 1 through 8 are created using the same number of assets and the credit rating as the original

Table 1. Test Benchmarks.

Benchmark	Based on	# of	# of	# of
#	Data from	Assets	Time	Default
			Steps	Curves
1	CDX.NA.HY	100	15	5
2	CDX.NA.IG	125	35	5
3	CDX.NA.IG. HVOL	30	19	4
4	CDX.NA.XO	35	22	4
5	CDX.EM	14	6	4
6	CDX.DIVER- SIFIED	40	23	5
7	CDX.NA.HY. BB	37	13	4
8	CDX.NA.HY. B	46	26	4
9	[3]	400	24	2

CDX indices. Based on the credit ratings, default boundary curves, $P(\tau_i < t)$, are obtained from Moody's[10]. However, since Moody's uses annual default rates, the values are extended using Eqn. (1) to attain quarterly time steps. The actual notionals are also obtained from [10], which are corporate bond defaults for 1999. There are a wide range of notionals from \$0.6 million to \$6.6 billion. A ninth benchmark is added to represent a very large collateral pool of 400 assets. The data for it is obtained from [3].

All other input data for every benchmark are randomly generated:

- α_i : uniformly distributed from [0, 1].
- Return rates: normally distributed with a mean of 0.40, ideal return rate [3], and 0.15 variance.
- 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 to one of the default boundary curves.

The tranche attachment points are taken to be the same as in CDX.NA.IG: 0%, 3%, 7%, 10% and 15%.

5.2. Design Evaluation

All FPGA designs are compared to a software implementation, written in C++, running on an Intel Xeon 3.4 GHz processor with 3GB RAM. All designs are written in Verilog and synthesized using Xilinx ISE 9.2. Resource utilization

and design frequency are post place and route values obtained using Xilinx Xplorer, which iteratively narrows in on an optimal design frequency. The results are validated using the Xilinx ML506 Evaluation Platform, which has a Virtex 5 XC5SX50T -1 speed grade chip, while the performance values in Table 2 assume a faster -3 speed grade.

To obtained performance and accuracy measurements each benchmark is run ten times, with different GRNG seeds, for 100,000 Monte-Carlo paths on all hardware designs and the double-precision C++ software implementation. All acceleration and accuracy values are reported with the software program as the baseline reference. For each design an average benchmark error is calculated as summation of the absolute distance between the design's result and the baseline result divided by the baseline result and averaged over the number of runs, ten. The average error reported in Table 2 is the benchmark error averaged over all benchmarks.

The most resource intensive portion of the design is in notional summation, Stages 2 through 5. We have explored different representations for the notionals: floating-point single-precision, double-precision, and fixed-point. We have also explored the benefits of using DSP units for the pipelined floating-point adders, in Stages 2 and 3. The results are summarized in Table 2, the percentages next to each utilization value indicate the portion of the total resource available on the chip that is being used by the design. For both single- and double-precision floating-point designs incorporating DSP units reduced the LUT and Flip-Flop utilization. However, the benefits are more evident in the single-precision representation where incorporating DSP units results in a larger LUT utilization savings, as well as a higher design frequency.

While the single-precision floating-point notional design occupies significantly less resources than the double-precision counterpart, the result has 1.97% error. To try to get the best of both worlds (the resource utilization of single-precision and the accuracy of double-precision) single-precision notionals are used in Stages 3 and 4, and a double-precision accumulator is incorporated at Stage 5. Experimentally, the error is found to be significantly reduced to 2.19E-5%.

Examining the data at all stages within the simulation it is established that 42 bits are sufficient to represent the notionals and 54 bits for the final accumulator to obtain identical results to the double-precision representation. This is shown as Fixed Point in Table 2. Through ISE it is found that each additional notional bit requires 62 additional Flip-Flops and 74 LUTs.

The least resource consuming design from each representation is replicated as many times as resources permit and incorporated into the overall simulation architecture as shown in Fig. 2. The Replicated Frequency is the performance of the multi-core system. The resulting acceleration is summarized at the bottom of Table 2. The smallest core,

Table 2. Performance/Area Results.

	Single-Precision Floating-Point		Double-Precision Floating-Point		Single- Precision Notionals & Double- Precision Accumulator	Fixed Point
	Without DSP	With DSP	Without DSP	With DSP		
Flip-Flops	7097 (21.7%)	6530 (20.0 %)	10454 (31.2%)	9910 (30.4%)	6721 (20.5%)	4906 (15.0%)
LUTs	8660 (26.5%)	7052 (21.6%)	13548 (41.5%)	13325 (40.8%)	7599 (23.3%)	5224 (16.0%)
BRAMs	15 (11.4%)	15 (11.4%)	31 (23.4%)	31 (23.4%)	15 (11.4%)	15 (11.4%)
DSPs	9 (3.1%)	29 (10.1%)	10 (3.4%)	40 (13.9%)	30 (10.4%)	7 (2.4%)
Freq (MHz)	235.2	248.8	187.3	190.9	244.8	268.2
Average Error (%)	1.97	1.97	0	0	2.19E-5	0
Single Core Acceleration	13.1x	13.9x	10.5x	10.7x	13.7x	15.6x
# of Cores	4		2		4	5
Replicated Freq (MHz)	208.4		140.8		210.0	218.5
Multi-Core Acceleration	46.5x		15.7x		46.9x	63.6x

Fixed Point, allows the most replications, five, which results in a 63.6 -fold acceleration.

6. CONCLUSION

This paper describes a hardware architecture for pricing Collateralized Debt Obligations using the One-Factor Gaussian Cupola Model [2]. We demonstrate how an FPGA can be used to exploit fine-grain parallelism in a Monte-Carlo financial model to achieve significant acceleration over the software implementation. We have also examined the precision requirements for the notional data and the resulting resource utilization. Similar to [5], we have established that a fixed-point representation can adequately represent the data, while utilizing the least resources. This is due to bounded notionals and a final accumulator that only needs to be large enough to a sum a known maximum number of notionals. Any other model with a similar structure can do the same. Future work will concentrate on expanding the simulation model to a more general multi-factor Gaussian Cupola.

7. REFERENCES

- [1] SIFMA, "Global Market Issuance Data," (2008). [On-line]. Available: http://www.sifma.org
- [2] D.X. Li. "On default correlation: A copula function approach," *The Journal of Fixed Income*, vol 9, pp 43-54. 2000.

- [3] K. Jackson, A. Kreinin, and X. Ma. "Loss Distribution Evaluation for Synthetic CDOs," Working paper. February 2007.

 [On-line]. Available: http://www.defaultrisk.com/pp_cdo_14.htm
- [4] G.L. Zhang et al. "Reconfigurable acceleration for Monte Carlo based financial simulation." *Proc. Int. Conf. on Field-Programmable Technology*, pp. 215-224. IEEE, 2005.
- [5] G.W. Morris and M. Aubury. "Design space exploration of the European option benchmark using hyperstreams." *Proc. Int. Conf. on Field Programmable Logic and Applications*, IEEE, 2007.
- [6] D.B. Thomas, J.A. Bower and W. Luk. "Automatic generation and optimization of reconfigurable financial Monte-Carlo simulations." *IEEE Int. Conf. on Application-Specific Systems, Architectures and Processors*, 2007.
- [7] D. B. Thomas and W. Luk. "Sampling from the multivariate Gaussian distribution using reconfigurable hardware." *In Proc. IEEE Symposium on FPGAs for Custom Computing Machines*, pages 3-12, 2007.
- [8] D. Wang, T. Svetkizarm, B. Santa, F.J. Fabozzi. "Pricing Tranches of a CDO and a CDS Index: Recent Advances and Future Research" October 2006 [On-line]. Available: http://www.defaultrisk.com/pp_cdo_44.htm.
- [9] Markit, "Markit CDX Indecies" (2008). [On-line]. Available: http://www.markit.com
- [10] Moody's Investors Services. "Historical Default Rates of-Corporate Bond Issuers, 1920-1999" January 2000. [On-line]. Available: http://www.moodyskmv.com