ACCELERATION OF DISTANCE-TO-DEFAULT WITH HARDWARE-SOFTWARE CO-DESIGN

Izaan Allugundu, Pranay Puranik, Yat Piu Lo and Akash Kumar

Department of Electrical & Computer Engineering
National University of Singapore
Singapore – 117583
Corresponding author email: akash@nus.edu.sg

ABSTRACT

The role of Credit Rating Agencies has come under intense scrutiny in the recent past due to their failure to accurately rate the issuers of debt obligations and instruments. With growing uncertainty in the markets and the need for accurate results, credit rating algorithms are getting more and more complex by the day.

Distance-to-default (DTD), or the leverage indicator, is one of the key indicators in credit research that determines the probabilities-of-default of firms. The greater the amount of historic data available for a given firm, the higher is the accuracy of the DTD results. However, this directly translates to a higher processing time and increases the costs of computation. The DTD computation features a linear workflow that is suited for implementation on a Field Programmable Gate Array (FPGA) which could lead to a more efficient and low-cost solution. Application of embedded platforms in implementing such algorithms have the potential to reduce the power consumption through parallelism and utilise an optimised solution offered by reconfigurable logic and customized hardware. In addition to the hardware solution, the right balance of software implementation can give the performance of such complex and intensive processes, an added boost. In this paper, we explore that very prospect of a hardware-software co-design and suitably implement a prototype of the DTD algorithm.

The software in our design is partly run on a 2.9GHz Intel processor and the FPGA soft-core processor (Microblaze) which is implemented on a Xilinx Virtex-6 ML605 FPGA, accelerated by hardware coprocessors. This resulted in a 16.6× and 317.17× speedup in the computation of the implied asset value and the log-likelihood function respectively as compared to a pure software implementation on a 2.9GHz Intel processor.

The authors would like to thank Dr. Oliver Chen and Mr. Chen Huanjia from the Risk Management Institute for their assistance in helping them understand the requirements and functionalities of the algorithm.

1. INTRODUCTION

Owing to the great amount of uncertainty in the financial industry today, emphasis is being laid on detailed and extensive analysis of financial data. In such a volatile macro-environment, the role of credit rating agencies (CRAs) becomes critical as they become key indicators that drive investor confidence. With significant changes taking place in the regulatory framework in the credit rating space, credit rating mechanisms and algorithms are constantly being updated. Further, in order to make them reliable and foolproof, an increasing amount of data is required.

Inherently, credit rating algorithms are computationally intensive. With the addition of the voluminous data used in this process, the resources required on CPUs and GPUs increase significantly along with the amount of time taken for these computations. A direct consequence of this is increased cost and energy consumption. In such a scenario, field programmable gate arrays (FPGAs), which are reconfigurable in nature, prove to be a handy alternative. They not only reduce the processing time, but costs are scaled down as well.

In the past, FPGA designs have typically utilised a fixed point arithmetic design and have been used to accelerate financial algorithms. These algorithms have revolved around single option pricing [1], credit derivative pricing [2], interest rates [3], Value-at-Risk [4] and other similar Monte-Carlo simulations. This paper takes a different approach to computing a separate class of financial algorithms – Distance-to-Default, using floating point arithmetic hardware, and compares the performance of such a design to a high performance vectorised MATLAB code.

Distance-to-default (DTD), also known as the leverage indicator, is one of the twelve risk indicators of a model which generates probabilities of default for over 50,000 firms on a daily basis. It determines the time frame in which a company is likely to default on its debt. In the existing set-up, the computation of DTD for all firms takes almost two

days to complete.

In this paper, we propose a hardware-software implementation of the DTD model, which provides a significant speedup over the software implementation by exploiting fine-grain parallelism within the model. To our knowledge, we are the first ones to implement such a system for the calculation of DTD. Our main contributions are:

- A methodology to design and implement the DTD model with hardware-software codesign.
- A pipelined hardware implementation of various parts of the DTD model, which includes the computation of the implied asset value and log-likelihood computation.
- A system enabling off-chip data transfers and computations.
- A comparison between the software implementation running on a 2.9GHz Intel processor and the software-hardware implementation running on the Xilinx Virtex-6 ML605 FPGA and the 2.9GHz Intel, resulting in a $16.6\times$ and $317.17\times$ speedup for two of the most compute intensive functions of the algorithm.

The following section of the paper, Section 2, discusses other notable financial algorithms which have been hardware accelerated along with their benefits, effectiveness and costs. Thereafter, Section 3 details how the DTD algorithm works and focuses on the key components of this process. Next, we move to Section 4, which gives an overview of the entire system developed, details of the hardware-software codesign and how the different components of the DTD algorithm are implemented. Section 5 is an evaluation of our system in which we look at the acceleration obtained in the components of our design. The speedup achieved on our software-hardware co-design is compared to the performance of this algorithm on a 2.9GHz Intel processor in terms of the time taken and the clock cycles along with a discussion of the resources used to implement the design. The final section, Section 6, sums up our results, the benefits of such a concept and possible areas for further research in this unexplored field of credit rating algorithms.

2. RELATED WORKS

The focus of research in the area of financial algorithms on FPGAs have been on the acceleration of Monte-Carlo based financial models. Monte-Carlo computations are based on repetitive random sampling which can be parallelised. A loans portfolio simulator was accelerated through a similar approach [5]. The simulator reflects changes in economic conditions and behaviour of individual loans within a portfolio. The hardware simulators were implemented using a Virtex-4 xc4vsx55 device running at 233MHz, and compared to four parallel software simulation threads running in a quad-core Pentium-4 Core2 at 2.4GHz. The speedup achieved was up to 100 times.

Another Monte-Carlo based simulation implemented using an FPGA accelerated model is the Collateralized Debt Obligations pricing [2]. The speedup achieved was over 63 times when implemented on Xilinx XC5VSX50T as compared to a software implementation on a 3.4 GHz Intel Xeon Processor.

In the FPGA acceleration of mean variance framework for optimal asset allocation, the design has combined software and hardware implementation. The generation of inputs is by software implementation (on a host PC) while the output is computed through hardware acceleration. The speedup achieved was 221 times over a software implementation running on two 2.4 Ghz Pentium-4 CPUs [6]. The methodology developed is similar to our approach of implementing the DTD system.

Previous work detailing comparisons of floating point and fixed point arithmetic operators used in financial applications reveal that fixed point implementations of algorithms are about 36.7% faster than floating point implementations, and 46.5% faster compared to PC implementations [2]. This implies that in cases where the dynamic range of floating point values is required, good performance gains can be obtained compared to PC software implementation. This underlines that the trade-off between precision, speed and computational resources used is still an open problem and also strongly depends on the target application.

Our implementation of the hardware cores utilizes the FloPoCo floating point arithmetic cores with an 8 bits exponential field and 23 bits significant field for compliance with the single-precision IEEE754 standard. The FloPoCo project [7] is a freely available arithmetic core generator for FPGAs that provides more arithmetic operations such as logarithm and exponential operations than what is provided by Xilinx's floating point core generator [8]. The FloPoCo cores also provides additional status bits optimized for FPGA design usage.

3. DISTANCE-TO-DEFAULT (DTD) BACKGROUND

The DTD variable is one of the twelve risk indicators used to compute the individual probabilities used in the overall credit research model. The DTD computation used in the CRI system is not a standard one. Standard computations exclude financial firms, but excluding the financial sector means neglecting a critical part of any economy. So the standard DTD computation must be extended to give meaningful estimates for financial firms as well. The following section first presents background information on the implied asset

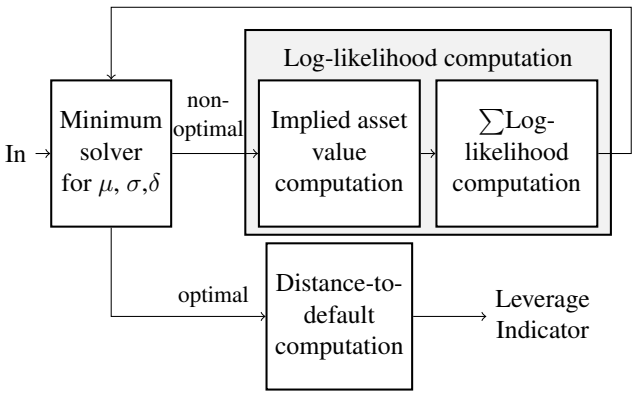

Fig. 1. Computation of the distance-to-default

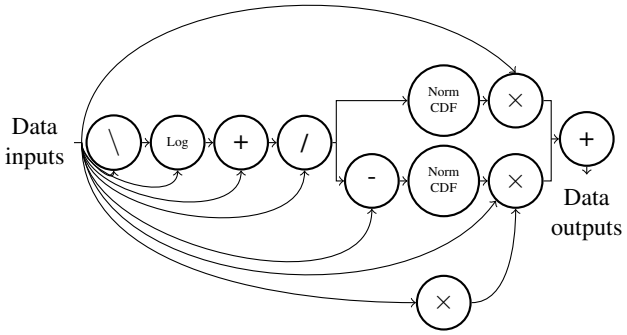

Fig. 2. Model transfer function for implied asset value

value computation, followed by the log-likelihood function and finally the minimum solver.

The computation of the distance-to-default first requires the calibration of the firm's parameters – μ, σ and δ, using a solver where the log-likelihood function is minimal. Then, the distance-to-default value is computed using the optimised set of variables along with the data gathered. As is seen in Figure 1, the log-likelihood function requires the computation of the implied asset value containing search methods and numerical analysis methods. The distance-to-default is currently computed with MATLAB on a grid of several hundred computers administered by the NUS Computer Centre. This set-up takes almost two days to carry out the DTD computation.

3.1. Implied Asset Value

The implied asset value takes in the set of equity values based on the company's market cap, debt and interest rate along with the variables to be optimised to determine the resultant asset value.

The computation of the implied asset value is done iteratively with numerical analysis methods such as bisection and Newton-Raphson after determining the bounds of the function using an exponential-step initial search until the output is stable. The model function is summarised in Figure 2. The step size for Newton-Raphson or the midpoint value for bisection for the step used in subsequent iterations is also computed.

3.2. Log-Likelihood Function

The log-likelihood function is a part of a two-stage process used in estimating the maximum likelihood of the unknown parameters – μ, σ and δ. Given a set of observed equity values of a firm and the corresponding calculated asset values, the likelihood of μ, σ and δ attaining certain values equals the probability of the observed set of equity values given the parameter estimates. This probability, called the log-likelihood function value, is then passed to the minimum solver to complete the parameter estimation process.

3.3. Minimum Solver

The computation of DTD requires multiple maximum log-likelihood estimations. The model involves a constrained minimisation of these multiple estimations to determine the parameters – μ, σ and δ, to be used for the computation of the DTD. In the current CRI system, the MATLAB function 'fmincon' from the Optimisation Toolbox is used. Since the optimization is over 13 dimensions, thousands of evaluations are required. Our new approach uses a C solver, Mixed Integer Distributed Ant Colony Optimization (MIDACO) [9].

4. SYSTEM OVERVIEW

The acceleration layer is developed as hardware coprocessors that are attached to the main processor through direct links as shown in Figure 3. The developed system utilises two types of hardware computation cores developed specifically to compute the implied asset value and to compute the sum of the log-likelihood function given the implied asset values, and allows the use of multiple hardware acceleration cores as needed by scaling up the number of cores and links.

The embedded system first receives the data-sets from the ethernet source containing the firm observation values on a daily basis. The software solver can decide the parameters μ, σ, δ by optimising the log-likelihood function and proceed to compute the distance-to-default if minimal. The processor also acts as an intermediate data storage device and as a control unit for the hardware blocks.

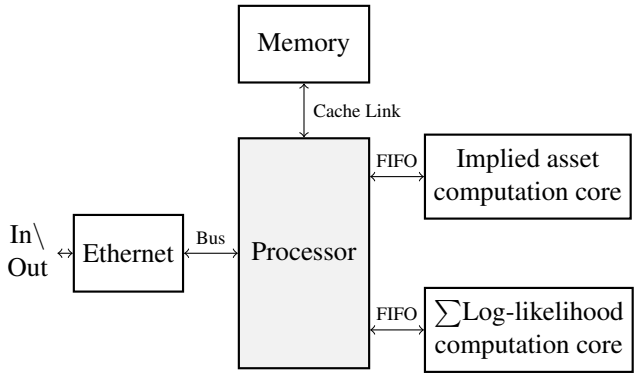

Fig. 3. System block diagram of the developed system

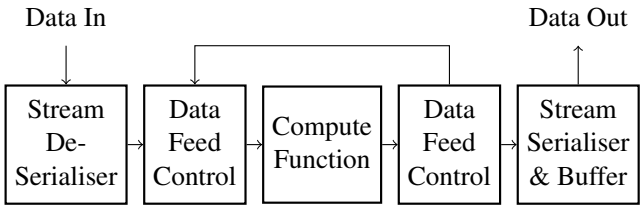

Fig. 4. Implied asset computation data-path

4.1. Implied Asset Value Computation

The implied asset value computation core is a fully pipelined design capable of producing up to 1 implied asset value per clock cycle. It is built using the FloPoCo arithmetic cores [7] with a few custom cores to provide functions such as single-cycle absolute, negation, multiply and divide by 2. Other crucial functions such as detecting the smallest difference of 2 floating-point numbers and the computation of the normal cumulative distribution function are also used in the development of the implied asset value data-path.

The process of computing the implied asset value begins with reading the data from the FIFO link. The values read are de-serialised and converted to the appropriate FloPoCo format. The persistent nature of the computation loop implies that the data feed control logic reads the new set of input values only if the pipeline slot is not filled by data from the previous iteration, else the write into the pipeline will be delayed.

The compute function of the implied asset computation data-path will compute the model transfer value, along with the computation of the step size for Newton-Raphson or the midpoint value for bisection for the step used in the subsequent iteration. In addition, the compute function will also updates the bound of the function to a narrower range if we are searching for the boundaries of the values or if bisection has been used. The data output of the compute function will then be checked by the data feed control which will output the value to the FIFO link if the data is ready, or continue through further iterations if otherwise.

4.2. Log-Likelihood Function Computation

The log-likelihood function utilises a software-hardware co-design. The C segment which runs on the MicroBlaze, orders and cleans-up data received from the PC. Hence, this part is light in terms of the complexity of the operations and the number of cycles they take. The hardware core carries out the computationally intensive operations to calculate the log-likelihood function value.This hardware core has been developed using the FloPoCo arithmetic cores and the customised normal cumulative distribution function core. There are three key components of this core which have been developed on the basis of the three important values required to compute the log-likelihood function value – sum_Nd1, sum_logVA, sum_sqterms.

The data to be fed into the core is read via the FIFO link. The 12 input variables are required at the same time for computation and are pumped into the computation core, as soon as they are all read. Thereafter, they are converted to the required FloPoCo format and pushed into the three individual components. Given that there are n number of valid observations, with each observation consisting of 12 variables, each of these components will produce n outputs every 12 cycles, which will be added to three different accumulators to sum up the results. Since each individual component has a different pipeline length, the three final outputs, which are the results from each individual accumulator, are produced at different times. However, they are pumped out of the core, after conversion to the IEEE754 format from the FloPoCo format, only when the final result from the component with the longest pipeline is ready.

It is important to note here that the final results will only be ready once all the n observations are processed. Till then, the results which are already processed are stored in registers, as shown in Figure 5. Hence, it is intuitive that the FIFO link keeps pumping in data every 12 cycles (that is, after every set of 12 new variables is received), irrespective of whether the previous variables have produced the required outputs, as long as it belongs to a single given firm. These three values which are pumped out of the core and into the C program are then used to compute the final log-likelihood function value.

4.3. Solver for Firm Parameters

The solver is utilised to determine the estimates of firm parameters – μ, σ and δ , from a specified range to be used for the computation of DTD. The parameters are established by constrained minimisation of the log-likelihood function. This process involves thousands of evaluations of the log-

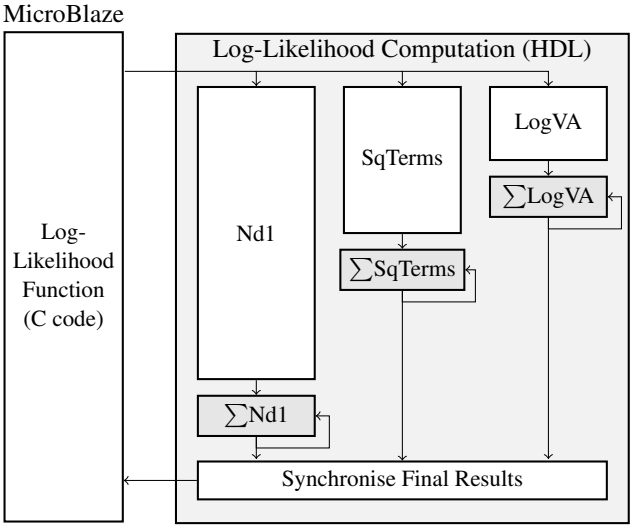

Fig. 5. Log-likelihood computation data-path

Table 1. Normal cumulative distribution performance

Platform	PC	FPGA	
Type	MATLAB	Virtex6 (C)	Virtex6 (HDL)
Values computed	511		
Values per Second	221,136	566	2,154,718
Cycles taken	6.76×10^6	9.023×10^7	2.372×10^4
Cycles per Value	13,232	176,585	46
Relative performance	1.00×	0.00256×	9.74×

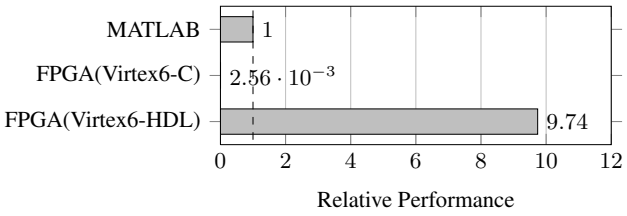

Fig. 6. Normal cumulative distribution computation performance

likelihood function. Our new approach uses a C solver, MIDACO. It is an innovative optimization solver of industrial strength for continuous and mixed integer problems. The underlying algorithm is based on a stochastic Gauss approximation technique and its combination with the novel oracle penalty method. The solver has been adapted to our approach for the DTD computation which is a combination of software and hardware implementation. In this unique set-up, the optimisation of the log-likelihood function for parameters, within a specified range, is performed through a software implementation on the CPU. However, the evaluation of each log-likelihood function is performed on the Microblaze processor.

5. EVALUATION

This section presents the test environment in order to analyze the performance of the hardware implementation, and compare the results obtained to the existing approach computed on a PC to analyse the performance improvements of the implementations.

5.1. Experimental Setup

In order to conduct a fair performance assessment of the developed hardware logic cores, we compare the results obtained running the software code with the developed hardware cores to the results obtained with a MATLAB implementation of the functions.

The hardware cores are tested and benchmarked on the Xilinx Virtex6 ML605 development board running at 100MHz with the throughput-optimised variant of the MicroBlaze processor and the developed hardware core. The MATLAB tests are run on a 2.9GHz Intel processor with 8GB of memory with the MATLAB code set to single precision, and threading limited to 1 to ensure consistent performance over runs.

5.2. Experimental Results

5.2.1. PC-FPGA Interface

The transfer of data between the CPU and Virtex-6 partition is managed through the Ethernet bus using TCP/IP packets. The connection is initiated on the CPU side with socket programming libraries and is established on the FPGA MicroBlaze Processor. The MicroBlaze Processor is instrumental in shuffling data between the computation cores and the external ethernet data source. It manages data transfer from the ethernet source using LwIP, a lightweight TCP/IP protocol suite that is suited for use with embedded platforms. The processed data is sent back to the main system through the Ethernet bus.

In our system, an input data array of size 214,880 bytes is required for the system to compute DTD. This data is transferred through the socket with a bitrate of over 375,000 bits per second and stored in main memory of the FPGA.

5.2.2. Normal cumulative distribution function

The normal cumulative distribution function is tested as a separate core to gauge the maximum throughput over the FIFO link. The test simulates looping through 511 input z-values stored in memory, processed using the pipelined hardware normal cumulative distribution function core, and the normal CDF value computed written back to memory. The results are shown in Figure 6 and Table 1.

Table 2. FPGA utilisation by hardware cores

Resource Type	Slices	LUTs	BRAM	DSP48E1
MicroBlaze (reference)	1,685	3,021	20	5
Normal CDF	3,240	7,976	1	27
Exponential	290	605	1	1
Implied Asset Value	10,434	25,723	18	71
Log-Likelihood Function	25,258	25,580	11	74

Table 3. Implied asset value performance

Platform	PC	FPGA	
Type	MATLAB	Virtex6 (C)	Virtex6 (HDL)
Sets computed	511		
Sets per Second	21,043	164	348,619
Cycles taken	7.11×10^7	3.13×10^8	1.47×10^5
Cycles per Set	139,050	611,524	287
Relative performance	1.00×	0.00777×	16.6×

The hardware implementation yielded 9.74× more normal cumulative distribution values computed per second as compared to the high speed general purpose processor running optimised vectorised MATLAB code.

The number of cycles spent per normal cumulative distribution value on the hardware core is 46 cycles, which is close to what is expected due to the loop control instructions, instruction word load and store instructions and computation of the offsets for the data arrays on the soft processor. The theoretical performance of the core is up to 1 normal cumulative distribution per value, which translate to 100 million values on a 100MHz co-processor design without the bottlenecks of the soft processor.

The hardware implementation of the normal cumulative core yields up to 3,804× the performance with 2× the resource requirements as compared to the C implementation on the throughput-optimised variant of the soft core processor as shown in Table 2.

5.2.3. Implied Asset Value

This test simulates the computation of the implied asset value for 511 sets of input data stored in memory, with some values precomputed using the soft core processor and the exponential hardware core before it is processed using the implied asset value pipelined hardware. The resultant implied asset value is then read from the core and written back to memory. The results are shown in Figure 7 and Table 3.

The increased number of FIFO link instructions due to the increase in the size of the data-set required for computation increased the number of cycles per set of input data to 287 cycles. The developed solution yielded a performance increase of 16.6× more implied asset values computed per second as compared to the optimised vectorised MATLAB computation. The maximum frequency of operation of this core is close to 197MHz.

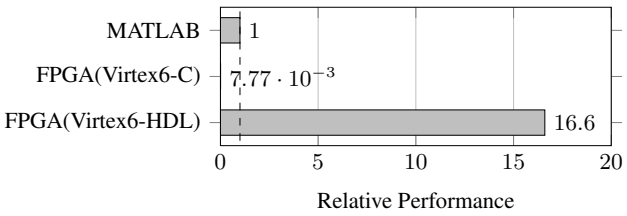

Fig. 7. Implied asset value computation performance

The performance can be improved further by scaling up the core frequency and reducing the transfer bandwidth and time required of the core by computing more starting parameters in the core as opposed to calculating it on the soft core processor, due to the need to transfer less values to the core. The theoretical performance of the co-processor is identical to what we can obtain from the normal cumulative distribution core due to the ability to process 1 set of data per cycle.

The hardware utilisation of the implied asset value computation core is about 10× the size of the soft core processor. However, the hardware implementation of the implied asset value computation yields a 2,132× performance speedup as compared to execution of the C code on the soft core processor, which represents a significant improvement in the performance per FPGA resource used.

5.2.4. Log-Likelihood function

The log-likelihood function is implemented on the MicroBlaze soft-core processor and the log-likelihood hardware co-processor. The data to be processed is stored in the memory and some basic operations are performed on the Microblaze soft core processor. The outputs of implied asset values function are processed and sent to the log-likelihood core using the second FIFO link. The data undergoes processing in the log-likelihood computation core, and the results are sent out via the FIFO link. These are used by the soft-core processor for the final computation of the log-likelihood function value. Since this final computation involves a few more complex operations, individual FloPoCo arithmetic cores are used (Log, $Squarer$, $Exponential$). The code has been placed in the block RAMs on the FPGA to improve performance.

In order to evaluate the efficiency of the log-likelihood function only, the implied asset value function used within the log-likelihood function, has been implemented in C. The performance results of this core are shown in Figure 8 and Table 4. Despite the FIFO read and write instructions adding to the number of clock cycles during the implementation on our hardware-software co-design, computations take 0.0001× fewer clock cycles as compared to MATLAB. Also, our core is 317.17× faster in terms of the processing time. The maximum frequency of operation of this core is close to 197MHz.

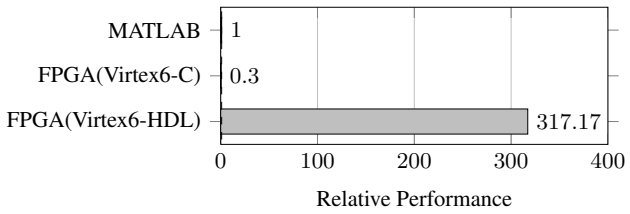

Fig. 8. Log-likelihood timing performance

Table 4. Log-likelihood performance

Platform	PC	FPGA	
Type	MATLAB	Virtex6 (C)	Virtex6 (HDL, C)
Outputs per Second	78.947	23.741	25039.65
Cycles taken	1080×10^5	126×10^5	1.12×10^5
Time Taken (s)	0.038	0.126	0.0001
Relative performance	1.00×	0.30×	317.17×

In the current set-up, given n sets of 12 variables, theoretically, the hardware core (excluding the software implementation and data transfer limitations) is capable of producing the desired outputs in 204 + ((n-1) × 12) cycles.

Having said that, the log-likelihood core can function as a fully-pipelined design. However, due to the limitations of the FIFO link (in terms of its transfer bandwidth), this attribute cannot be utilised. Presently, the core waits till it receives the entire set of 12 values before processing it. Likewise, it waits for all its three independent sections to finish computation before pushing out the values together. The input and output logic can be modified to further improve performance. Instead of waiting for all the 12 input variables to be received, they should be pumped to the respective segments as soon as the necessary input variables for each independent segment arrive. A similar approach should be adopted while outputting data. Such an implementation would perhaps require multiple instantiations of the segment with the longest pipeline. In addition, more than one FIFO link would be required.

6. CONCLUSIONS

This paper described the design and implementation of floating point hardware to accelerate the computation of the distance-to-default, along with critical software parts, that enables off-chip data transfers and computations. While many of our hardware and software designs can be further optimised, they can already run 16.6× and 317.17× faster in the computation of the implied asset value and the log-likelihood function respectively as compared to a MATLAB implementation on a 2.9GHz Intel processor.

We plan to extend our implementation by producing more extensively connected subsystems and the inclusion of more computations on the FPGA. We also intend to further optimise the software implementation to enhance the performance of the system.

7. REFERENCES

[1] G. Morris and M. Aubury, "Design space exploration of the european option benchmark using hyperstreams," in *Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on*, aug. 2007, pp. 5 –10.

[2] A. Kaganov, P. Chow, and A. Lakhany, "FPGA acceleration of Monte-Carlo based credit derivative pricing," in *Field Programmable Logic and Applications, 2008. FPL 2008. International Conference on*, 2008, pp. 329–334.

[3] D. Thomas, J. Bower, and W. Luk, "Automatic generation and optimisation of reconfigurable financial monte-carlo simulations," in *Application -specific Systems, Architectures and Processors, 2007. ASAP. IEEE International Conf. on*, july 2007, pp. 168 –173.

[4] D. Thomas and W. Luk, "Sampling from the multivariate gaussian distribution using reconfigurable hardware," in *Field-Programmable Custom Computing Machines, 2007. FCCM 2007. 15th Annual IEEE Symposium on*, april 2007, pp. 3 –12.

[5] D. B. Thomas and W. Luk, "Credit risk modelling using hardware accelerated monte-carlo simulation," in *Proceedings of the 2008 16th International Symposium on Field-Programmable Custom Computing Machines*, ser. FCCM '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 229–238. [Online]. Available: http://dx.doi.org/10.1109/FCCM.2008.41

[6] A. Irturk, B. Benson, N. Laptev, and R. Kastner, "FPGA acceleration of mean variance framework for optimal asset allocation," in *High Performance Computational Finance, 2008. WHPCF 2008. Workshop on*, nov. 2008, pp. 1 –8.

[7] FloPoCo Team, "Floating-point arithmetic core generator," 3 2011. [Online]. Available: http://flopoco.gforge.inria.fr/

[8] Xilinx Inc, "Floating Point Operator v5.0 (DS335)," 1 2011. [Online]. Available: http://www.xilinx.com/support/documentation/ip_documentation/floating_point_ds335.pdf

[9] "Midaco - solver." [Online]. Available: \url{http://www.midaco-solver.com/}