Exploring Financial Applications on Many-Core-on-a-Chip Architecture: A First Experiment

Weirong Zhu[1], Parimala Thulasiraman[2,⋆],
Ruppa K. Thulasiram[2], and Guang R. Gao[1]

[1] Department of Computer Science, University of Manitoba Winnipeg,
MB R3T 2N2 Canada
[2] Department of Electrical and Computer Engineering, University of Delaware,
Newark, DE, USA
{thulasir, tulsi}@cs.umanitoba.ca,
{weirong, ggao}@weirong@capsl.udel.edu

Abstract. Computational requirements for solving models of financial derivatives, for example, the option pricing problems, are huge and demand efficient algorithms and high performance computing capabilities. This demand has been rekindled by the recent developments in the mobile technology making wireless trading a possibility. In this paper, we focus on the development of a Monte-Carlo algorithm on a modern multi-core chip architecture, Cyclops-64 (C64) under development at IBM as the experimental platform for our study in pricing options. The timing results on C64 show that various sets of simulations could be done in a real-time fashion while yielding high performance/price improvement over traditional microprocessors for finance applications.

1 Introduction

Research in financial derivatives is one of the important areas of computational finance. Finance models used for evaluation and forecasting purposes to help the investor with the selection process typically lead to large dynamic, nonlinear problems that have to be solved in a short time span to beat the competitors in the market place. The computational requirements for solving such financial models are huge and demand efficient algorithms and high performance computing capabilities [1].

In our study, we focus on development of a Monte-Carlo algorithm with historic volatility and GARCH (Generalized Auto Regression Conditional Heteroskedasticity) fitted volatility to price options accurately on Cyclops 64 (C64), a modern many-core-on-a-chip architecture. The purpose is to facilitate pricing of options in a real time fashion. Moreover, our earlier studies using Monte-Carlo technique for the option pricing problem on distributed architectures yielded results [2,3] that are not amenable for real-time implementation, an important requirement for current day trading scenario.

⋆ Corresponding author.

This is a first and preliminary study in option pricing on a many-core-on-a-chip architecture. This study opens up many more studies and has many implications: (i) this could be a fore-runner for futuristic embedded architectures such as mobile devices where the current chip technology could be replaced by a CMP (Chip Multi Processor); (ii) wireless trading is becoming a reality in the recent past [4,5,6] and the current work could become a fore-runner for real-time wireless trading; (iii) experience from the current study encourages us to look into other computing techniques popularly used in finance, for example, binomial lattice and finite-differencing technique for implementation on many-core-on-a-chip architecture to enable real time trading.

1.1 Background and Related Work in Option Pricing

A *Call Option* [7] is a contract that gives the right to its holder (i.e. buyer) without creating an obligation, to *buy* a pre-specified underlying asset at a predetermined price (*strike price*). Usually this right is created for a specific time period (*maturity date*), e.g. six months. A *Put Option* gives to its holder the right to *sell*. If the option can be exercised only at its expiration/maturity date (i.e. the underlying asset can be sold only at the end of the life of the option), the option is referred to as an European style Call/Put Option (or *European Call/Put*). If it can be exercised on any date before its maturity, then the option is referred to as an American style Call/Put Option (or *American Call/Put*).

Black and Scholes [8] proposed a model to price option, which has become a classical and celebrated model for pricing options. This model is basically a stochastic partial differential equation with option price as the unknown and underlying asset price and the time being dependent variables together with various parameters such as volatility of the asset price, expiration date, strike price and interest rate. In the current study future asset prices are generated with random number generated in the Monte-Carlo (MC) simulation and volatility is generated by two methods: historic volatility (based on the past changes in the asset price) and GARCH fitted volatility. Clark [9] and Thulasiram et al. [10] developed parallel algorithm for the binomial lattice approach [11] to price options. Use of fast Fourier transform (FFT) technique for option pricing was introduced by Carr and Madan [12]. Extending this model, Barua et al. [13] have developed an efficient parallel algorithm to enable quicker and accurate pricing of options by introducing data swapping technique in FFT. Mayo [14] evaluated American options using the implicit finite-difference method giving a fourth order accuracy in the log of the asset price and second order accuracy in time. Thulasiram et al. [15] designed a second order L_0 stable algorithm for the pricing problem which achieves the same error bound as that of the traditional Crank-Nicholson scheme, while at the same time assures that the error will not propagate. Srinivasan [16] used the quasi MC simulation technique for option pricing while Rahmail et al. [2] used the traditional MC simulation to study the effect of incorrect volatility for underlying assets on option pricing errors.

MC simulation is a forward-based procedure. Option pricing via MC can be divided into three basic steps: (1) simulate the stochastic process underlying stock returns, where each realization is a sample path; (2) evaluate the value of the option in a backward manner in order to find the early exercise point and obtain a sample point estimate; and, (3) average over multiple sample estimates to form an interval estimate that includes some measure of precision (e.g., standard error). Obviously, the existence of the precision measure is an advantage of MC over other numerical methods.

In this study, we explore the possibility of expediting the MC simulation using a many-core-on-a-chip architecture in the hope that future hand-held devices will be embedded with CMP architectures and hence achieving parallelism on such devices.

2 Cyclops-64 Architecture

The Cyclops64 (C64), based on a cellular architecture, is a Many Core System on Chip (SoC) petaflops supercomputer (see Figure 1) project under development at IBM T.J. Watson Laboratory. C64 system consists of 13,824 C64 chip, connected by a 3D mesh network. The C64 chip architecture (Figure 2) consists of 160 hardware thread units, half as many floating point units, same amount embedded SRAM memory banks, an interface to off-chip DDR SDRAM memory, and bidirectional inter-chip connection ports on a single silicon chip. Each of the 80 processors have two thread units, a floating point unit, and two SRAM memory banks of approximately 32KB each. Five processors share a 32KB instruction cache. Instead of data cache, a portion of each thread unit's corresponding on-chip SRAM bank is configured as the scratchpad memory (SP). Therefore, a thread unit can achieve fast access to its own SP, i.e., one cycle for a store, and two cycles for a load. The remaining sections of all on-chip SRAM banks together form the global memory that is uniformly addressable from all thread units. The C64 also employs the Network-on-Chip (NoC) concept, all on-chip

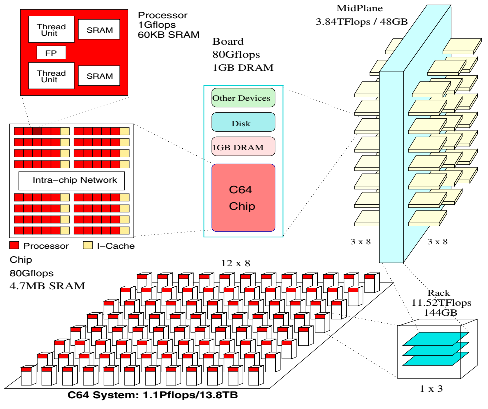

Fig. 1. Cyclops 64 Supercomputer

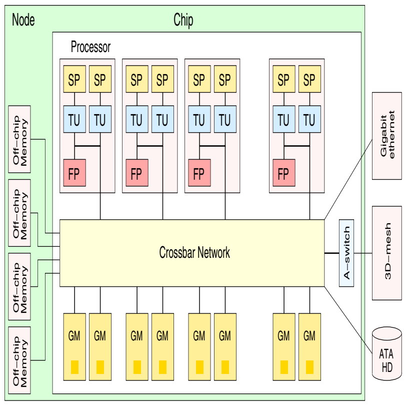

Fig. 2. Cyclops 64 Node

resources are connected to a 96 ports on-chip crossbar network, which provides a 4GB/s bandwidth per port per direction, 384 GB/s per direction in total. This huge bandwidth sustains all the intra-chip traffic communication.

3 Experimental Results

3.1 Monte Carlo Experiment Design

In the Black-Scholes option pricing model volatility is not observable. In the experiments we consider the following: (a) use of historical volatility of continuously compounded stock returns; (b) use of GARCH-fitted volatility of continuously compounded returns.

During the experiments we generated stock price series under the assumption that prices follow a random walk with drift. We generated increments using the normal probability distribution function. In various stages of the experiment volatility of increments was: (i) Constant; (ii) Decreasing; (iii) Increasing; (iv) Stochastic; (v) Decreasing and stochastic; and, (vi) Increasing and stochastic.

Using the generated volatilities σ_t and a pseudo-random number generator, we generate stock price series that follow the geometric Brownian motion process:

$$lnS_t = \gamma + \delta lnS_{t-1} + \nu_t \tag{1}$$

where γ is the drift in the stock price, σ is the variance rate (volatility) of the stock price, In equation 1 we have $\gamma > 0$ to make sure that prices do not fall below zero and that the increment is normally distributed:

$$\nu_t \approx N(0, \sigma_t^2) \tag{2}$$

We assumed the continuously compounded interest rate of 5% and the flat and deterministic yield curve, as it is assumed in the Black-Scholes model. Expiration date was set at 3 months from the starting point (time t), and strike price was varyied from 5 to 105 with the step size of 20. The starting prices P_0 in all cases were \$5.00. Using these parameters, we calculated call prices for the non-dividend paying stocks for each point in time t using all inputs as known. The formula used in calculations is the classical option pricing formula (see [7]). Next, we calculated option prices with all the same inputs but measured volatility.

In the first run of the experiment we estimated conditional volatilities and used them in the option pricing formula. In the second run of the experiment we estimated historical and GARCH-fitted volatilities of continuously compounded stock returns [17]. After calculating option prices (C^{TRUE}) using known data and option prices using observable and measured data ($C^{MEASURED}$), we calculated the option pricing error E in the following way:

$$E = C^{MEASURED} - C^{TRUE} \tag{3}$$

This would give us a dollar estimate of the error in case of using an improper measure of volatility in the Black-Scholes option pricing formula.

The experimental results on C64 have both the pricing and performance results. We describe our experimental design for the current study to compute the option values under two volatility criteria and the errors resulting from their use for computing the option values. On a AMD-K7-II processor, with 6 possible exercise prices, 6 patterns of volatility, and 1000 data points, one run of the experiment using the E-Views statistical package for only 20 iterations took 4 hours and 45 minutes.

Figure 3 depicts one of the many sets of pricing errors that results from the experimental study. This graph shows, which of the data generating processes creates larger errors when we use GARCH-fitted volatility of continuously compounded returns. The x-axis in all these figures corresponds to the strike prices ranging from \$5 -\$105 and the y-axis corresponds to the option pricing errors: -0.004 to 0.003 in fig 3 (a); -0.05 to 0.03 in fig 3 (b); -0.06 to 0.01 in fig 3 (c); -0.04 to 0.04 in fig 3 (d); -20.0 to 140.0 in fig 3 (e); -0.20 to 0.06 in fig 3 (f).

Figure 3 (a) corresponds to constant volatility, Figure 3 (b) corresponds to decreasing volatility, Figure 3 (c) corresponds to increasing volatility, Figure 3 (d) corresponds to stochastic volatility, Figure 3 (e) corresponds to decreasing and stochastic volatility, Figure 3 (f) corresponds to increasing and stochastic volatility. The six legends below each of these figures identify the prices starting from \$5 to \$105 in steps of \$20 for respective figures.

During the experiments the drift component of the stock price is \$0.006t, where t stands for the number of days. Therefore, we are able to plot call pricing error against various exercise prices and unconditional expectations of stock prices ($E[S_t] = 0.006 * t$). In this part of the experiment, $GARCH$ model is estimated for the sample size k with the mean equation that regresses continuously compounded returns on a constant. We generate fitted volatility and record the last value h_k. This is our input into the Black-Scholes formula for calculating the measured call price, $C_k^{MEASURED}$. Next, we add one more data point to the stock price series, estimate the GARCH model for the sample of

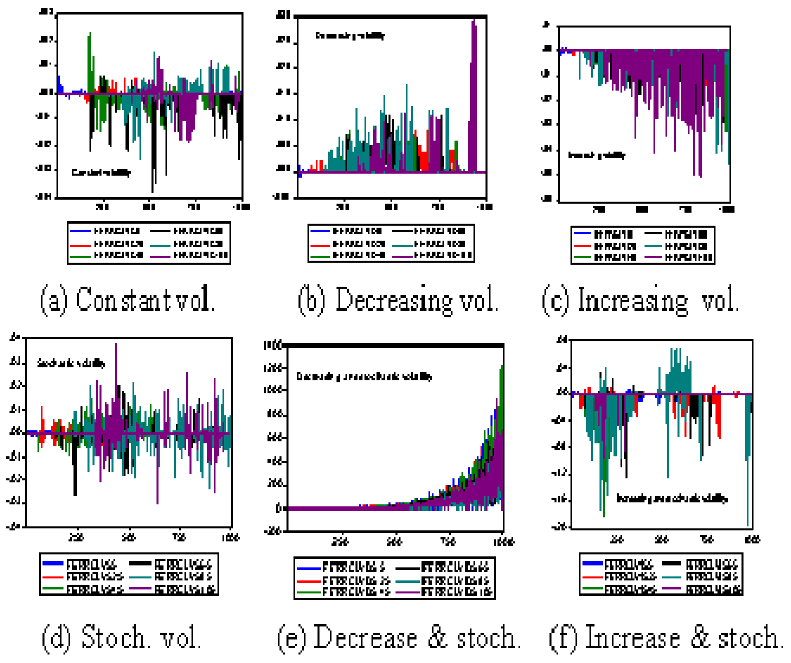

Fig. 3. Option pricing errors for stock prices generated using various patterns of volatility, average across series. GARCH-fitted volatility estimates based on continuously compounded returns are used to generate option prices.

$k+1$ observations, and use h_{k+1} to calculate $C_{k+1}^{MEASURED}$. Using this process, we generate the time series of measured call prices $C_t^{MEASURED}$.

As we can infer from Figure 3, in some cases (for example, cases b and e) option pricing errors grow with higher sample size. This can be attributed to the non-stationarity of option prices: as sample size increases, sample volatility of data approaches infinity. Therefore, we get upward-biased estimates for the volatility of stock prices. The comparative statistics of Black-Scholes model show that call price increases when stock price volatility increases. Therefore, upward-biased estimates of stock prices result in upward-biased estimates of option prices. This situation could result in a false belief that there exists a Put-Call-Parity arbitrage strategy based on erroneously calculated call prices. In case of constant and stochastic volatility of prices (Fig. 3 (a and d)) the situation is not as clear. Option pricing errors seem to be fluctuating around zero on the average.

In the following subsection, we demonstrate the performance results of running the simulation on the C64 architecture.

3.2 Monte Carlo Simulation on C64

The performance of the Monte Carlo algorithm on C64 is compared to different representative off-the-shelf processors: AMD Opteron 250, Intel Centrino, and Intel Pentium 4. The basic configurations of those processors are shown in Table 1. The computation conducted on C64 is simulated with the FAST simulator [18], which is a functionally accurate simulation tool set for the C64 cellular architecture. The parameter setting for 4 different simulations is shown in Table 2.

Table 1. Processor Configurations

Processor	Clock Rate	Cache	off-chip Memory	Compiler
Cyclops-64 Thread Unit	500MHz	No data cache 5MB on-chip SRAM memory	1GB DRAM	gcc-3.2.3 for C64
AMD Opteron	2.4GHz	1MB L2 cache	3GB	gcc-3.2.3 for x86_64
Intel Centrino	1.86GHz	2MB L2 cache	512MB	gcc-3.3.6
Intel Pentium4	3.2GHz	512KB L2 cache	1GB	gcc-3.4.3

Table 2. Parameters for Monte Carlo Simulation

Parameter	begprice	step	variety	ARSIZE
sim-1	5	20	6	1000
sim-2	5	40	8	1000
sim-3	5	40	8	2000
sim-4	5	100	10	5000

For C64, a portion of each SRAM bank can be configured as the scratchpad memory (16KB, in this case), which guarantees fast and predictable access latency for the corresponding owner thread unit. For the Monte Carlo simulation, we carefully design the code, such that the intermediate results of the computation can completely fit into a thread unit's scratchpad memory. Only the latest ith simulation results were stored on-chip for the next (i+1)th simulation. The previous 1, ... , $i - 1$ results were stored off-chip.

We performed on average ten runs on each of the machine and obtained the execution times as shown in Figure 4 for the simulation parameters chosen in Table 2, where *begprice* is the beginning price of the simulation, *step* is the step size on price, *variety* is the number of volatility patterns, and *ARSIZE* is the number of data points in each simulation.

In the Monte Carlo simulation all the thread units can work independently during the simulation. Since all the intermediate data needed for a thread unit's computation can fit into its own scratchpad memory, there is no runtime competition and conflict for shared resources, such as global on-chip memory. Through calculations for all four groups of simulations, for the Monte Carlo option pricing simulation, we can conclude that 1 C64 node delivers the performance equivalent to 18 Opteron 250 CPU, 32 Intel Centrino CPU, and 28 Intel P4 3.2GHz CPU. The approximate price for a C64 node would be quite similar to machines built with those CPUs compared. The speedup of the parallel version can increase linearly with the number of threads used (this is also demonstrated with

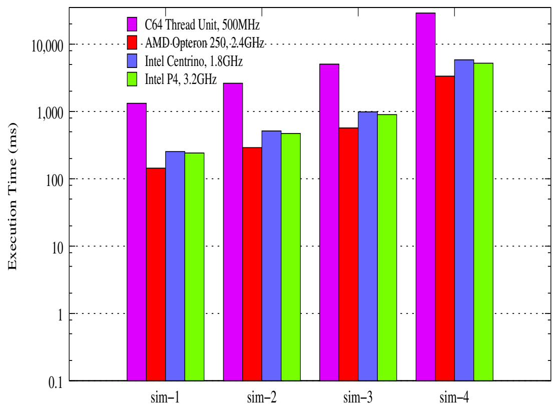

Fig. 4. Execution Time of Monte Carlo Simulation

the simulation). Therefore, the C64 delivers huge *performance/price* improvement over traditional microprocessors for computational finance applications. Moreover, for the Monte Carlo simulation, since all data fits into the scratch-pad memory, all the computation is performed locally for each thread units and the power on the very long wires going to and from the crossbar is saved. In such a situation, the power consumption of a C64 node is lower than or close to machine built with conventional microprocessor. Given a C64 node delivers tens of times performance, the C64's *performance/power consumption* ratio is much higher compared to other microprocessors. Unlike a traditional microprocessor, which dies if any parts on the chip is broken, the C64 chip can still be in working condition, even if one or more thread units/memory banks fail.

4 Conclusions

The current work is a first attempt to study (option pricing using Monte-Carlo simulation). We showed that the use of incorrect volatility of the asset prices in the Black-Scholes model would result in moderate to large errors in option prices. We conclude that 1 C64 node delivers the performance equivalent to 18 Opteron 250 CPU, 32 Intel Centrino CPU, and 28 Intel P4 3.2GHz CPU. This translates to high *performance/price* and *performance/power consumption* improvement over traditional microprocessors for computational finance applications.

Acknowledgement

Part of this work was conducted while the second and third authors were visiting University of Delaware in Summer 2005. These authors acknowledge the financial support from Natural Sciences and Engineering Research Council of Canada and the University Research Grant Program of the University of Manitoba. The first and last authors would like to acknowledge the support from

IBM, in particular, Monty Denneau, who is the architect of IBM Cyclops-64 architecture, ETI, the Department of Defense, the Department of Energy (DE-FC02-01ER25503), the National Science Foundation (CNS-0509332), and other government sponsors. We would also like to acknowledge other members of the CAPSL group at University of Delaware, who provide a stimulus environment for scientific discussions and collaborations, in particular Juan del Cuvillo, and Ziang Hu.

References

1. E. J. Kontoghiorghes, A. Nagurnec, and Berc Rustem. Parallel Computing in Economics, Finance and Decision-making. *Parallel Computing*, 26:207–509, 2000.
2. Sergiy Rahmayil, Ilona Shiller, and Ruppa K. Thulasiram. Different Estimators of the Underlying Asset's Volatility and Option Pricing Errors: Parallel Monte Carlo Simulation. In *Proc. Intl. Conf. on Computational Finance and its Applications*, pages 121–131, Bologna, Italy, April 2004.
3. Gong Chen, Ruppa K. Thulasiram, and Parimala Thulasiraman. Distributed Adaptive Quasi-Monte Carlo Algorithm for Option Pricing on HNOWs Using mpC. In *Proc. 9th Annual Simulation Sympoisum*, pages 90–97, Huntsville, AL, April 2006.
4. Kiran Kola. WAMAN:Web-mining-Assisted Mobile-computing-enAbled on-line optioN pricing- a software architecture towards autonomic computing. Master's thesis, Department of Computer Science, The University of Manitoba, Winnipeg, MB, Canada, May 2006.
5. H. Kargupta, B. Park, S. Pittie, L. Liu, D. Kushraj, and K. Sarkar. Mobimine: Monitoring the stock market from a pda. *SIGKDD EXplorer*, 3:37–46, 2002.
6. U. Varshney and R.J. Vetter. Mobile commerce:framework, applications and networking support. *MONET*, 7:3–4, 2002.
7. John C. Hull. *Options, Futures and Other Derivatives.* Prentice Hall, Upper Saddle River, NJ, 5 edition, 2002.
8. F. Black and M. Scholes. The pricing of options and corporate liabilities. *J.Political Economy*, 81:637–654, January 1973.
9. Iain J. Clark. Option Pricing Algorithms for the Cray T3D Supercomputer. *Proceedings of the first National Conference on Computational and Quantitative Finance*, September 1998.
10. R. K. Thulasiram, L. Litov, H. Nojumi, C. T. Downing, and G. R. Gao. Multithreaded Algorithms for Pricing a Class of Complex Options. In *Proc. Intl. Parallel and Distributed Processing Symp. (IPDPS01)*, San Francisco, CA, April 2001.
11. J.C. Cox, S.A. Ross, and M. Rubinstein. Option pricing: A simplified approach. *J. Financial Economics*, 7:229–263, 1979.
12. Peter Carr and Dilip B. Madan. Option Valuation using the Fast Fourier Transform. *The Journal of Computational Finance*, 2(4):61–73, 1999.
13. Sajib Barua, Ruppa K. Thulasiram, and Parimala Thulasiraman. High Performance Computing for a Financial Application using Fast Fourier Transform. In *LNCS Vol. 3648, Proc. European Parallel Computing Conference (EuroPar05)*, pages 1246–1253, Lisbon, Portugal, aug-sep 2005.
14. A. Mayo. Fourth Order Accurate Implicit Finite Difference Method for Evaluating American Options. In *Proc. Intl. Conf. on Computational Finance 2000*, London, England, June 2000.

15. Ruppa K. Thulasiram, Chen Zhen, Amit Chhabra, Parimala Thulasiraman, and Abba Gumel. A second order l_0 stable algorithm for evaluating european options. *Intl. Journal of High Performance Computing and Networking (in press)*, 2006.
16. Ashok Srinivasan. Parallel an Distributed Computing Issues in Pricing Financial Derivatives through Quasi Monte Carlo. In *Proc. Intl. Parallel and Distributed Processing Symp. (IPDPS02)*, Fort Lauderdale, FL, April 2002.
17. M. Chesney and L. Scott. Pricing European Currency Options: A Comparision of the Modified Black-Scholes and a Random Variance Model. *Journal of Financial and Quantitative Analysis*, 24:267–284, 1989.
18. Juan del Cuvillo, Weirong Zhu, Ziang Hu, and Guang R. Gao. FAST: A functionally accurate simulation toolset for the Cyclops64 cellular architecture. In *Workshop on Modeling, Benchmarking, and Simulation (MoBS2005)*, Madison, WI, June 2005.