Proceedings of the International MultiConference of Engineers and Computer Scientists 2014 Vol II,
IMECS 2014, March 12 - 14, 2014, Hong Kong

Accelerating Financial Code through Parallelisation and Source-Level Optimisation

Nan Zhang and Ka Lok Man

***Abstract*—In this paper we summarise the experiences we obtained during past years in accelerating financial code through parallelisation and source-level optimisation. We have been focusing on developing optimised parallel programs to speedup financial computations where either binomial tree method or Monte Carlo simulation was applicable. The parallelisation was through explicit POSIX multi-threading on x86 shared-memory multi-processor systems. The source-level optimisations we found most useful were data structure optimisation and elimination of common sub-expressions.**

***Index Terms*—Parallel computing, Monte Carlo simulation, Binomial tree method, Source code optimisation, POSIX multi-threading**

I. Introduction

SOFTWARE routines that solve computational finance problems are often time and resource consuming. The purpose of this paper is to briefly discuss two practical approaches to accelerate such financial routines, namely, parallelisation and source code optimisation. We will discuss their applications on binomial tree method and Monte Carlo simulation. The purpose is to speedup their executions on x86 shared-memory multi-processor systems. Both these two methods are popular computational approaches in quantitative finance. Binomial tree method is often applied in pricing American-style options whose built-in feature of early exercise cannot be handled by analytical methods. However, for those complex options whose value depends on a basket of basic assets with high-dimensionality Monte Carlo simulation is often the only effective way to evaluate their price.

The parallelisation under our discussion is achieved through explicit POSIX multi-threading. In this approach one has to explicitly create and manage threads through invoking corresponding POSIX functions. Although it requires much more programming efforts than automatic multi-threading through OpenMP directives, the performance is much better. The code optimisations we are going to discuss include data structure optimisation and elimination of common sub-expressions.

II. Explicit multi-threading in binomial tree method

The binomial tree method is an often-used approach in financial computing to solve problems like option pricing. To parallelise the computation on a binomial tree the key things are how to decompose the tree so that different threads work on different parts independently, and how to synchronise the working threads. Fig. 1 shows such an example where the nodes in a 12-level binomial tree are processed by three threads in parallel. The parallel algorithm reported in [1] partitions a binomial tree into blocks of multiple levels. A block is further divided into sub-blocks. Each sub-block is assigned to a working thread. All working threads in parallel process the sub-blocks in a block, and then, when finishing, they move to the next block. The algorithm is an improved version from the one presented in [2].

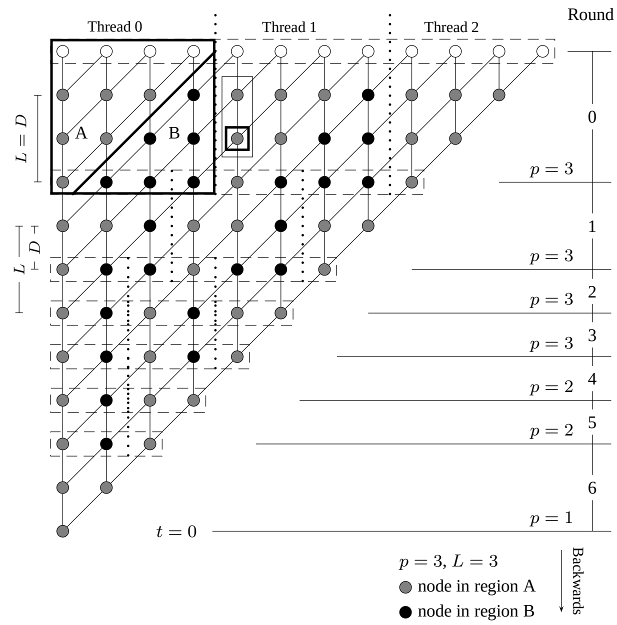

Fig. 1: Processing the nodes in a binomial tree by multiple threads.

A slightly modified algorithm is designed to work on CPU-GPU heterogeneous platforms [3]. Because on a GPU, accessing local memory is much more faster than accessing global memory, the GPU binomial tree algorithm uses double buffers in shared local memory to reduce the times the global memory has to be accessed.

III. Explicit multi-threading in Monte Carlo simulation

Monte Carlo simulation is another popular method used to solve complex computational finance problems. In a typical application of this approach a large number of scenarios are generated, and computations are performed on each of these scenarios. Usually, the computations performed on one scenario is independent of the computations performed in another scenario. For this reason, it is natural to parallelise Monte Carlo simulations using multiple processors. For instance, on a shared-memory multi-processor system

Manuscript received December 10, 2013; revised January 31, 2014.

Nan Zhang is with the Department of Computer Science and Software Engineering, Xi'an Jiaotong-Liverpool University, China. Email: nan.zhang@xjtlu.edu.cn

Ka Lok Man is with the Department of Computer Science and Software Engineering, Xi'an Jiaotong-Liverpool University, China, and Baltic Institute of Advanced Technologies, Vilnius, LithuaniaEmail: ka.man@xjtlu.edu.cn

ISBN: 978-988-19253-3-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

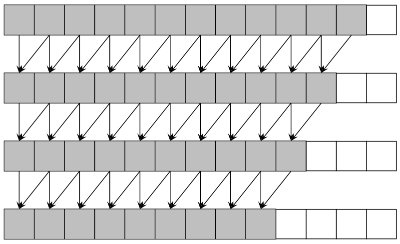

Fig. 2: Computations performed in an one-dimensional array which stores nodes at a level of a binomial tree.

if the number of processors is p, the number of scenarios to generated is n, a simple parallelisation scheme is to divide the n scenarios evenly divided among the p processors so that each processor works on n/p scenarios.

Monte Carlo approach often involves using random numbers. It is more desirable if the generation of random numbers is also parallelised. For this purpose the random number generator must support the "skipping ahead" method, such as the ones in Intel MKL [4]. With the same definitions for numbers n and p, if m is the number of random variates needed in one scenario, for the parallel generation to work the i-th processor must skip imn/p numbers in the random variate stream and generate from the imn/p-th position of the stream.

The authors' past work on this topic can be found in [5], [6], [7]. In [5], [6] the authors worked out Monte Carlo based algorithms to price American multi-asset stock options and American interest rate swaptions, respectively. In [7] the mortgage optimal refinancing problem is tackled by the parallel Monte Carlo simulation.

IV. Code optimisation in financial programming

Option pricing is at the core of many computational finance problems. Its computational routine should be written in a way that is highly efficient. For instance, to compute implied volatilities option pricing routine is repeatedly called by a root-finding procedure that compares the calculated option price with the market price. For Monte Carlo simulations, because of the large number of generated scenarios, the time needed by the computation is usually long. For these reasons, optimisations in source code level, besides parallelisation, is often necessary in order to shorten the execution times of the computational finance procedures.

One of the many optimisation techniques we find useful is to use simple data structures. For the binomial tree method, although a tree is a two-dimensional structure, we do not have to explicitly build a tree in memory. Instead, an one-dimensional array is sufficient to store the option values represented by the nodes under processing. All the computations can be done in an one-dimensional array as Fig. 2 shows. Maintaining and traversing an one-dimensional array is much faster than working with a two-dimensional tree.

Another category of code optimisation often applicable to financial code is avoiding repetitive computation on common sub-expressions. In many cases, because of the way those mathematical expressions are constructed there are common parts in them. To save computational time we can compute the common part once and save its value and re-use this value in subsequent computations. For example, at the k-th level of a binomial tree, stock prices represented by the nodes are $S_0u^k, S_0u^{k-2}, \ldots, S_0u^{-k}$, where S_0 is the initial stock price and u is the up-move factor. To compute these values we let $X_0 = S_0u^k$, $X_1 = S_0u^{k-2}$, $X_2 = S_0u^{k-4}$, etc, and $U = u^{-2}$. We can see that $X_1 = X_0U$, $X_2 = X_1U$, $X_3 = X_2U$, etc. So, in runtime what we can do is we save the value of X_i and re-use it to compute X_{i+1}. This saves execution time because multiplication takes much less time than the transcendental operation in computing u^k. This makes a big difference, especially when the option pricing routine gets called repeatedly by some higher-level procedure, as in the case of calculating implied volatilities [8]. Another example that shows such optimisation works is reported in [6], where in computing the drift term in the extended LIBOR market model this optimisation saves a big amount of execution time. To apply this optimisation one often needs to observe carefully on those mathematical constructions and find out the common sub-expressions.

V. Conclusion

We have selected some past work to discuss and the experience we have learnt. On modern x86 multi-core processors through multi-threading and certain source-level optimisations the performance of financial code can be greatly improved, as our past work demonstrate. Writing POSIX multi-threaded code by hand is a bit hard work. But the performance improvement brought about justifies the efforts spent. Mathematical expressions in those financial models are often constructed in a recursive way such that there are common sub-expressions. When implementing these models on computers one has to observe carefully to find out common sub-expressions so that repetitive computations on these sub-expressions can be avoided. In out past experience this saved a great amount of execution time.

References

[1] N. Zhang, A. Roux, and T. Zastawniak, "Parallel Binomial American Option Pricing under Proportional Transaction Costs," *Applied Mathematics*, vol. 3, pp. 1795–1810, Nov 2012, doi:10.4236/am.2012.331245.

[2] A. V. Gerbessiotis, "Architecture Independent Parallel Binomial Tree Option Price Valuations," *Parallel Computing*, vol. 30, pp. 301–316, 2004.

[3] N. Zhang, C.-U. Lei, and K. L. Man, "Binomial American Option Pricing on CPU-GPU Hetergenous System," *Engineering Letters*, vol. 20, no. 3, pp. 279–285, Aug 2012.

[4] *Intel Math Kernel Library Reference Manual - MKL 11.0 Update 5*, Intel Corporation, 2013, Document Number: 630813-061US. http://software.intel.com/sites/products/documentation/doclib/mkl_sa/11/mklman/mklman.pdf.

[5] N. Zhang and K. L. Man, "Parallel Valuation of the Lower and Upper Bound Prices for Multi-Asset Bermudan Options," in *Lecture Notes in Computer Science*, J. J. P. et al., Ed., vol. 7513. Springer, 2012, pp. 458–467.

[6] N. Zhang, K. L. Man, and E. G. Lim, "Pricing Bermudan Interest Rate Swaptions via Parallel Simulation under the Extended Multi-Factor LIBOR Market Model," in *Lecture Notes in Computer Science*, vol. 7513. Springer, 2012, pp. 477–486.

[7] N. Zhang, D. Xie, E. G. Lim, KaiyuWan, and K. LokMan, "Parallel Generation of Optimal Mortgage Refinancing Threshold Rates," in *Lecture Notes in Computer Science*, J. P. et al., Ed., vol. 7861. Springer-Verlag Berlin Heidelberg, 2013, pp. 665–675.

[8] N. Zhang and K. L. Man, "Fast Generation of Implied Volatility Surface for Exchange-Traded Stock Options," in *Proceedings of the International MultiConference of Engineers and Computer Scientists 2013*, vol. 2. IAENG, Mar 2013, pp. 741–746.