Acceleration of Binomial Options Pricing via Parallelizing along time-axis on a GPU

Narayan Ganesan
Dept. of Comp. Sci. and Engg
Washington University in St. Louis
Email: nganesan@wustl.edu

Roger D. Chamberlain
Dept. of Comp. Sci. and Engg
Washington University in St. Louis
Email: roger@wustl.edu

Jeremy Buhler
Dept. of Comp. Sci. and Engg
Washington University in St. Louis
Email:jbuhler@wustl.edu

I. Introduction

Since the introduction of organized trading of options for commodities and equities, computing fair prices for options has been an important problem in financial engineering. A variety of numerical methods, including Monte Carlo methods, binomial trees, and numerical solution of stochastic differential equations, are used to compute fair prices. Traders and brokerage firms constantly strive to achieve faster calculation of option prices because timely information can mean the difference between a deal struck or missed, which translates to substantial profit or loss. Hence, the latency to compute a fair option price plays an important role in short-term trading and arbitrage.

Financial firms constantly seek faster, more accurate option pricing methods in order to keep up with or ahead of their competitors. Approaches to improve latency include both more efficient trading strategies or pricing algorithms and use of specialized, high-performance computer architectures, such as FPGAs, many-core CPUs, and GPUs. An exemplary low-latency implementation on FPGA of European Options pricing via Monte-Carlo simulation was described in [1], where $15\times$ speedup over an existing server, as well as outperforming a GPU and Cell implementation was reported. Monte-Carlo methods are usually exhaustive, time and compute intensive and take into consideration various sources of uncertainties corresponding to real market conditions. On the other hand, lattice(esp. binomial tree) methods are faster and consider relatively few possibilities, uncertainties and is a reasonable approximation to a variety of standard market conditions.

In general, lattice methods in finance, generate a discrete lattice in price and time(possibilities) and iterate backwards from the expiration time to the current time. In this work, we describe a new strategy to accelerate one of the most widely used option pricing algorithms, the recombinant binomial tree model [2], on fine-grained parallel architectures. Implementations on software programming languages such as Fortran, C/C++, MATLAB, S-Plus, VBA Spreadsheets etc., are widely used in the financial industry. In addition there are also other proprietary implementations of the algorithm optimized for performance and latency on the native computer or architecture.

A parallel implementation of binomial tree method on FPGA was described in [6], wherein the multiple nodes corresponding to the same time index was computed concurrently. Another parallel implementation for GPUs on the CUDA programming language is released by NVIDIA [3] [4]. Multiple threads are programmed to work on multiple nodes of the tree corresponding to a single time-step followed by sequential processing of successive time-steps. In addition, the coarse grained parallelism is effectively utilized compute multiple options concurrently.

In this work we propose a strategy to accelerate the computation of a single option pricing over the existing parallel implementations as opposed to bulk options pricing. We achieve this by processing multiple time instants of a single binomial tree concurrently in order to minimize the latency. This strategy could be extremely useful in updating the option price of a single "high-stake" or a small group of very important assets subject to variations in underlying assumptions or changes in market conditions, so as to execute trading decisions with minimum latency. More often than not, fluctuations in specific market conditions or changes in underlying assumptions affect only a subset of the assets within the entire portfolio. Hence, a minimum latency updating of the relevant prices is extremely important in the light of competition. In the binomial tree model, since time instants are inherently dependent on the previous time-instants we propose a strategy to break the relation by introducing symbolic dependencies between nodes far apart in time. Since the symbolic dependencies could be computed in parallel we thus achieve a speedup compared to other implementations. The implementation was carried out NVIDIA GPUs, though the algorithm could be implemented on other parallel architectures as well. We theorize an optimal speedup of $15\times$ over a comparable parallel implementation for a problem size of 1000 time steps. In general the expected speed-up is proportional to the square root of the problem size.

II. Background

An option is an agreement between an *option seller* and an *option buyer* for the right to buy or sell an asset for a fixed price X at some future time T. The agreed-on time T of the asset transaction is its *expiration time*, while the agreed-on price X is called the *strike price*. The asset itself has a time-varying price $S(t)$; at the time the option is created, this is price called the *spot price* S. The option buyer may *exercise* the option at time T to buy or sell the asset at price X, regardless of its actual value $S(T)$ at that time; alternatively, the option may be allowed to expire without being exercised. The price of an option depends not only on S, T, and X but also on factors such as the inherent volatility σ in the asset's price and the short-term *risk-free interest rate* r, which provides a guaranteed rate of return. A *call option* grants the holder the right to buy the underlying asset at the strike price, while a *put option* grants the holder the right to sell the underlying asset at the strike price.

The *profit* made by exercising an option is the difference between the underlying asset price $S(T)$ and the strike price X at the the time of exercise, minus the price of the option. For call options, the profit increases with $S(T) - X$, while for put options, it increases with $X - S(T)$. The fundamental assumption in determining the fair option price for an asset, called the *risk-free assumption*, is that the total profit made from the entire transaction, given the volatility of the asset, must be at least equal to the return from an investment at the risk-free interest rate.

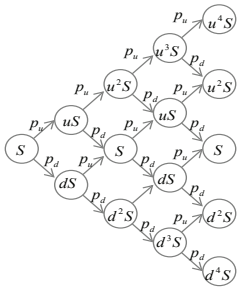

Fig. 1. A binomial tree describing the evolution of an asset's price.

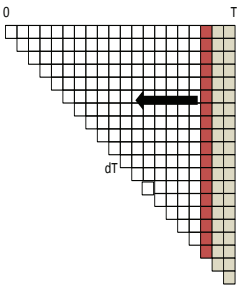

Fig. 2. Dynamic programming matrix for computing option prices. We may concurrently compute all prices at time $t - \Delta t$ given the prices at time t.

A. Binomial Tree Options Pricing

The binomial tree pricing model is a probabilistic model that describes the change in price of an asset over time. The model discretizes time into increments Δt and assumes that in each time increment, the price can only go up or down by a fixed fraction. If $S(t)$ is the asset price at time t, then the price $S(t+\Delta t)$ at the next time step either increases to $u \cdot S$ with probability p_u or decreases to $d \cdot S$ with probability $p_d = 1 - p_u$. The factors u and d depend on the asset's volatility σ; in particular, $u = e^{\sigma\sqrt{\Delta t}}$ and $d = e^{-\sigma\sqrt{\Delta t}}$. Note that $u \cdot d = 1$.

Under the risk-free assumption, the price of the asset S at time $t+\Delta t$ is given by $S(t+\Delta t) = e^{r\Delta t}S(t)$. Assuming the investment is also risk-free, the expected price of the asset at time $t + \Delta t$,

$$S(t + \Delta t) = p_u uS(t) + p_d dS(t) \tag{1}$$

should be equal to $S(t)e^{r\Delta t}$. The probabilities of the movements can now be determined as $p_u = (e^{r\Delta t} - d)/(u - d)$. Now, with the purchase of the option at the price f, the value of the entire portfolio is $S(t) - f$. Under an upward or downward movement of prices, the expected value of the portfolio is $p_u(uS - f_u) + p_d(dS - f_d)$ where f_u and f_d are new values of the options under a up-tick or down-tick of the asset price at time $t + \Delta t$. Under the risk-free assumption, the price of the options f_u and f_d can now be related to its current price f by

$$f = e^{-r\Delta t}(p_u f_u + (1 - p_u)f_d) \tag{2}$$

from Equation (1) and the fact that $p_d = 1 - p_u$.

III. Implementation

We want to compute the current option price f for an underlying asset with spot price S and strike price X at expiration time T. We are also given the asset's volatility σ and the assumed risk-free interest rate r.

To compute f, we use a backwards dynamic programming recurrence based on Equation (2) as follows. The recurrence is initialized at the leaves of the binomial tree in Figure 1, corresponding to the expiration time T, and proceeds backwards to the root node, which corresponds to the current time (arbitrarily designated as 0). In general, let $f_t(c)$ be the option price at time t assuming that the corresponding asset price at this time is c. At time T, the fair price for a call option is simply the difference between the asset price at time T and the strike price, i.e.

$$f_T(S(T)) = \max(S(T) - X, 0).$$

The max with 0 indicates that the option is of no value if the strike price is greater than the asset's actual price at time T. To work backwards to earlier time steps, we use Equation (2) as follows:

$$f_{t-\Delta t}(c) = e^{-r\Delta t}(p_u f_t(u \cdot c) + (1 - p_u)f_t(d \cdot c)). \tag{3}$$

We store the option prices corresponding to each asset price at time t in an array F_t, such that $F_t[i] = f_t(u^{t/\Delta t} \cdot d^{2i} \cdot S)$, for $0 \le i \le t/\Delta t$. Using this mapping, we can rewrite the recurrence (3) as

$$F_{t-\Delta t}[i] = e^{-r\Delta t}(p_u F_t[i] + (1 - p_u)F_t[i + 1]). \tag{4}$$

This last recurrence is computed as depicted in Figure 2;by iterating $T/\Delta t$ time steps, the desired option price for time 0 is now $F_0[0]$.

Given sufficient computational resources, the option prices at time $t - \Delta t$ can all be computed concurrently based on the prices at time t. For example, in a GPU, the array F_t could be stored in shared memory, and $|F_t|$ threads could be used to simultaneously compute the option prices. Setting $N = T/\Delta t$, an architecture with sufficient parallelism could therefore compute the desired price f_0 in time $\Theta(N)$, whereas a serial implementation would require time $\Theta(N^2)$. If parallelism is limited to at most n concurrent processing elements, computation time could be reduced to $\Theta(N^2/n)$.

A. Parallelization

Following equation (4), the price of the options at time $t - 2\Delta t$ can be related to the price at time t as follows,

$$F_{t-2\Delta t}(i) = e^{-r\Delta t}(p_u F_{t-\Delta t}(i) + (1 - p_u)F_{t-\Delta t}(i + 1))$$

and hence,

$$F_{t-2\Delta t}(i) = e^{-2r\Delta t}(p_u^2 F_t(i) + 2p_u p_d F_t(i + 1) + p_d^2 F_t(i + 2))$$

For the next step the option prices at time $t - 3\Delta t$ could be related to the prices at t via the relation,

$$F_{t-3\Delta t}(i) = e^{-3r\Delta t}(c_0 F_t(i)+c_1 F_t(i+1)+c_2 F_t(i+2)+c_3 F_t(i+3))$$

for some coefficients $c_0, \cdots, c_3$. In fact, the process could be extended to any number of finite steps to arrive at a relation, where,

$$F_{t-N\Delta t}(i) = e^{-Nr\Delta t} \sum_{j=0}^{N} c_j F_t(i + j) \tag{5}$$

Hence with the knowledge of the coefficients c_j and the option prices at time f_t, it is possible to determine the corresponding prices at time $t - N\Delta t$. The coefficients c_j could either be calculated as a closed form expression or could be updated iteratively as,

$$c'(i) = p_u c(i) + p_d c(i + 1) \tag{6}$$

and setting $c'(i) = c(i)$ at the end of the iteration, with memory and compute requirements exactly same as the option calculation itself. Hence the threads working on the rightmost partition compute the actual option prices while the threads working on other partitions update the coefficients $c(i)$ based on their values in the previous iteration. Hence a dependency relation is established between the

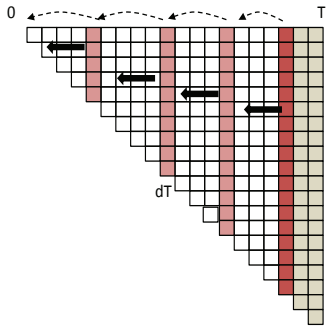

Fig. 3. The calculations can be parallelized where the rightmost partition updates the option prices while the other partitions update the dependencies.

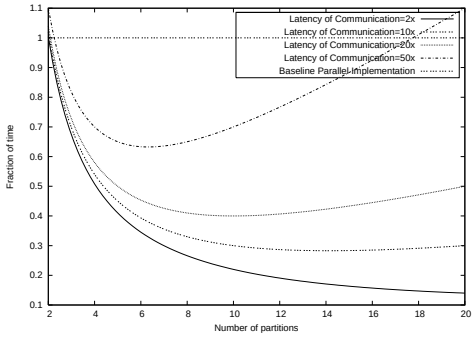

Fig. 4. Relative time with respect to a comparable parallel implementation. The expected time for execution for different latencies for a problem of 1000 time steps.

start and the end of each partition of size T/p time steps, where p is the number of partitions. Now, the option prices could be propagated from one boundary to the next, starting from the last with the dependency relation just established, with a stride of T/p time steps until we reach the first partition, which bears the option price at the current moment, thus achieving a speed-up of p, as shown in figure (3). Now, with the knowledge of the option prices at each boundary, the values in the interior nodes could be filled in parallel for all the partitions, if needed(as in American options).

In a parallel architecture such as a GPU, the different partitions were computed independently on individual multiprocessors. While each of multiprocessors themselves work on building a dependency relation between the boundaries of the partition, they could be sized so that the total number number of nodes within each is the same. This would ensure proper load balancing and minimize the latency.

1) Optimality: While the latency of communication of boundary values between the partitions L depends on the specific architecture, for a GPU the communication between the multiprocessors is achieved through the global memory. The entire computation is structured into 3 stages, where the first stage builds up the dependency relations between the partitions in parallel, the second stage fast-propagates the boundary values up the time axis sequentially and the third fills in the values if necessary(as in an American option). With a total of p partitions, a simplistic analysis would indicate, the total execution time for N time steps with a binomial tree of $N^2/2$ nodes is,

$$T_{ex} = \frac{2N}{p} + Lp \tag{7}$$

for optimality,

$$\frac{dT_{ex}}{dp} = -\frac{2N}{p^2} + L = 0 \tag{8}$$

or $p = \sqrt{2N/L}$. Hence the total number of partition is inversely proportional to the latency of communication between them. The optimal time of execution is $2\sqrt{2NL}$. Hence, as the latency decreases the number of partitions could be increased and time of execution could be minimized. Since the latency is measured in units of the fastest available communication within the device, the minimum value is 1. It can be seen that the for smaller values of p the speedup is proportional to the number of partitions and for larger values the incremental speedup is marginal and the communication latency L overwhelms the performance, as shown in figure 4. However, a more rigorous analysis would involve shared-memory characteristics of the GPU, thread scheduling and the time to compute the intermediary sums (5).

An option pricing tree of 1024 time steps was computed on the GPU, GeForce 8600GT. The values of the nodes for a tree at most as large as 1024 time steps in single precision were stored within the 16K shared memory. Multiple blocks were processed on different multiprocessors, which worked on computing the corresponding symbolic coefficients. At the end of the current block the relating symbolic coefficients were written on to the global memory in a coalesced fashion thus utilizing the maximum bandwidth to the memory. The first(rightmost) block in charge of numerical computation fetches the coefficients from the global memory after its numerical phase, which is guaranteed to take longer than the other blocks due to its size. The symbolic coefficients are then read concurrently by an independent warp while threads in other warps processed the coefficients for the previous blocks, thus implementing a effective double buffer for concurrent communication and computation. Under the given latency in communication and intermediary processing, a speedup of 2x could be achieved in evaluating a single options tree, with a 4-way parallel GPU.

IV. Conclusion

In this work we described a framework for parallelizing dependent operations by breaking their dependency. Such a scheme could be extremely useful in accelerating other applications including, dynamic programming, numerical solution to differential equations etc. We demonstrated the applicability of this approach to options pricing problem via binomial trees and its implementation on a GPU.

Acknowledgment

This research has been supported by NIH grant R42 HG003225 and Exegy, Inc. R.D. Chamberlain is a principal in Exegy.

References

[1] G. W. Morris and M. Aubury, *Design Space Exploration of the European Option Benchmark using Hyperstreams*, Intl. Conf. on Field Prog. Logic and App., pp. 5-10, 2007.
[2] J. C. Cox, S. A. Ross and M. Rubinstein, *Option Pricing: A Simplified Approach*, J. of Financial Economics, Vol. 7, pp. 229-64, 1979.
[3] http://www.nvidia.com/cuda
[4] C. Kolb, M. Pharr, *Option pricing on the GPU* in GPU Gems 2, ISBN 0-321-33559-7, Addison-Wesley, Chapter 45, 2005.
[5] F. Black and M. Scholes, *The pricing of options and corporate liabilities*, J. of Political Economy, Vol. 81, pp. 637-59, 1973.
[6] N. Singla, S. Parsons, M. A. Franklin and D. E. Taylor, *Method and System for High Speed Options Pricing*, Patent Application US 2007/0294157 A1, 2007.