SCCS-273b

The Fifth Australian Supercomputing Conference
Royal Melbourne Institute of Technology and The University of Melbourne
Melbourne, Australia, December 7-9, 1992

Software Issues and Performance of a Parallel Model for Stock Option Pricing [1]

Kim Mills
Gang Cheng
Michael Vinson
Sanjay Ranka
Geoffrey Fox

Northeast Parallel Architectures Center

Syracuse University
111 College Place, Syracuse, New York 13244-4100 U.S.
kim@npac.syr.edu

Abstract

The finance industry is beginning to adopt parallel computing for numerical computation, and will soon be in a position to use parallel supercomputers. This paper examines software issues and performance of a stock option pricing model running on the Connection Machine-2 and DECmpp-12000. Pricing models incorporating stochastic volatility with American call (early exercise) are computationally intensive and require substantial communication. Three parallel versions of a stock option pricing model were developed which varied in data distribution, load balancing, and communication. The performance of this set of increasingly refined models ranged over no improvement, 10 times, and 100 times faster than a sequential model. A straightforward approach to this problem involves use of two-dimensional dynamic arrays. When asymmetric arrays are mapped on the DECmpp-12000, distribution of data to physical processors is inefficient and performance suffers. The regular communication patterns in the model can also be expressed in one-dimensional arrays, improving data distribution. Performance of this version is similar on both parallel machines. Combining one-dimensional parallel and sequential arrays achieves efficient data distribution, reduces interprocessor communication, and further improves performance (100 times faster than a sequential workstation model). The performance improvements possible on parallel supercomputers presents new opportunities for

[1] We gratefully acknowledge support for this study from the Office of the Vice President for Research and Computing at Syracuse University, and Corporate Partnership funding from Digital Equipment Corporation.

pricing entire portfolios, performing large scale model and market comparisons, and using optimization techniques to improve model price estimates.

Introduction

Option pricing models

Stock options are contracts that give the holder of the contract the right to buy or sell the underlying stock at some time in the future. Option contracts are traded just as stocks are traded, and models that quickly and accurately price option contracts are valuable to traders and financial managers. Speculators participate in the option market to capture potential high profits with relatively small investment capital. Financial managers buy and sell options to hedge risk in their investment portfolios.

Since the opening of the first organized options exchange in April, 1973, by the Chicago Board of Options Exchange, and the introduction of a constant volatility, European pricing model [1], finance researchers have sought improved methods to price options with stochastic volatility on American contracts. Key model parameters, which cannot be directly observed but must be estimated from market information, include volatility of the underlying asset σ, variance of the volatility ξ, and correlation between asset price and volatility ρ.

Monte Carlo models are the conventional standard of comparison for option pricing models, but are computationally so intensive that they tend to be used only for research purposes. Binomial models are used as approximations of Monte Carlo models for pricing options with stochastic volatility and American call (early exercise). Binomial models use binary trees to represent possible up/down movements in asset price over the life of an option contract, and are more efficient than Monte Carlo methods. Finucane [2] demonstrated that binomial methods provide price estimates within a few cents of Monte Carlo models for market observations with known model parameters σ, ξ, and ρ.

Performance issues

The purpose of this study is to examine performance issues encountered in implementing an option pricing model, written in Fortran90, on the CM-2 and the DECmpp-12000. We focus on a single model, a binomial model incorporating stochastic volatility and American call, and apply this model to a set of options market data. Load balancing and communication are the important computational issues in this application. We use three versions of the pricing model to examine performance. In a related study we compared four option pricing models with historical market data [4]. Models incorporating stochastic volatility with American call produce more accurate price estimates than simpler models based on constant volatility and European call. Option pricing models are highly sensitive to model input parameters. Preliminary studies show great promise for using optimization techniques for model parameter estimation [4].

Summary of results

In our initial approach, we used two-dimensional arrays to run the pricing model in Fortran90 on the Connection Machine-2 and DECmpp-12000. This approach follows the natural structure of the problem, but requires dynamic distribution of data on the parallel machines. We observed important differences in performance between CM-2 and DECmpp-12000 for this version of the model. This pricing model requires a substantial amount of communication, but happens to be nearest neighbor communication along only one axis. This feature allowed us to define a second version of the model using a static data distribution, and similar performance was observed for both parallel machines. In a third version of the model, we took advantage of a data layout strategy making use of serial arrays, and reduced interprocessor communication. This version of the model runs approximately 100 times faster on the CM-2 and DECmpp12000 than a sequential version of the model running on a SUN4(25MHZ) workstation.

A Parallel Binomial Pricing Model

Mathematical formulation

The binomial model represents the continuous time processes of stock price and volatility as discrete up/down movements. Following the discussion in [2], the mathematical formulation of the binomial pricing model can be outlined in the following way. Volatility, σ, and stock price, S, follow continuous time stochastic processes represented as

$$\frac{d\sigma^2}{\sigma^2} = \mu_\sigma dt + \xi d\widetilde{W} \tag{1}$$

$$\frac{dS}{S} = \mu_s dt + \sigma d\widetilde{Z} \tag{2}$$

where $\widetilde{W}$ and $\widetilde{Z}$ are standard Wiener processes with correlation ρ. Change in stock price, or more accurately, the ratio of stock price change to initial stock price $\frac{dS}{S}$ is expressed as the sum of an expected component $\mu_s dt$ (drift of stock price over time) and a random component $\sigma d\widetilde{Z}$ (volatility).

The magnitude of the increase (u) or decrease (d) in volatility for any given time period is expressed as

$$u = e^{(\mu_\sigma - \xi^2/2)\Delta t + \xi\sqrt{\Delta t}} \tag{3}$$

$$d = e^{(\mu_\sigma - \xi^2/2)\Delta t - \xi\sqrt{\Delta t}} \tag{4}$$

where the probability of an increase/decrease being equally likely, μ_σ is the drift of the volatility process (a constant) and ξ is the variance of the volatility (not directly observed, but estimated from market data). With the introduction of correlation, ρ, the variance of stock price (which is volatility squared) after i periods with j upward movements and $i-j$ downward movements is defined as

$$\sigma^2 = \left(\sigma^2_{0,0}\right) u(\rho)^i d(\rho)^{i-j} \tag{5}$$

where $\sigma^2_{0,0}$ is initial volatility (estimated from market data). In the limit, as Δt approaches zero, the binomial process approaches the continuous time process

$$d\sigma^2 = \mu_\sigma \sigma^2 dt + \xi \sigma^2 d\widetilde{W} \tag{6}$$

The magnitude of increases (U) and decreases (D) in stock price at the i, jth position within the binomial lattice is defined as

$$U_{i,j} = e^{(r_f - \sigma_{i,j}^2/2)\Delta t + \sigma_{i,j}\Delta t} \tag{7}$$

$$D_{i,j} = e^{(r_f - \sigma_{i,j}^2/2)\Delta t - \sigma_{i,j}\Delta t} \tag{8}$$

Problem structure

A binomial lattice is illustrated in Figure 1 showing asset price or volatility over time. Important elements of the binomial lattice include initial price (S_0) and volatility (σ_0 or V_0), time of dividend payout (t_{div}), the $2^{t_{\text{div}}}$ nodes at time of the dividend where t_{div} ranges over values 1 to $T-1$, and the 2^T nodes at terminal time T. A single option price C_0, is estimated by integrating over the 2^T prices at time T and discounting to the present time T_0. The life of an option contract is typically represented in $T = 17, 18, 19, 20$ periods. A model of size of $T = 17$ was used by [2] in a previous, related study, and we found little improvement in model error reduction with model sizes greater than 17 periods in this study. We applied the binomial model to Chicago Board of Options Exchange (CBOE) market data for January through June, 1988. In the discussion below, we compare parallel versions of the pricing model with a sequential Fortran77 version of the model running on a SUN4 (25MHZ) workstation. The sequential model runs in approximately 4.0 seconds.

Early exercise and shape of arrays

American pricing models incorporate early exercise, which can occur at any time in the life of the option contract. In practice, early exercise occurs just prior to dividend payout. In the first version of our model, we express the two-dimensional lattice structure (stock price over time, volatility over time) in two-dimensional Fortran arrays. We designate the time steps in our model from 1 to t_{div} as stage 1 of the model, and timesteps from t_{div} to maturity T as stage 2 of the model. This breakdown of the American pricing model allows us to easily track price movements after dividend payout and determine percentages of early exercise.

Figure 2 illustrates how we express the binomial lattice in two-dimensional Fortran arrays. Stage 1 of the model runs from period $T = 1$ through the time of dividend payout, t_{div}. After dividend payout, further up/down moves of the $2^{t_{\text{div}}}$ nodes in the lattice at time t_{div} are tracked in stage 2 of the model. As Figure 2 shows, when $t_{\text{div}} = 2$, the $2^{t_{\text{div}}}$ or 4 nodes in the lattice evolve into 2^{T-2} nodes at terminal time T. In version one of our model, the value of t_{div} defines the shape of the two dimensional Fortran array $(1 : 2^{t_{\text{div}}}, 1 : 2^{T-t_{\text{div}}})$. Each market observation has its own value of t_{div} which is not accessible to the model until runtime, requiring dynamic arrays.

Data distribution

We observed important differences in performance between the CM-2 and DECmpp-12000 for version one of our model. Figure 3 illustrates the difference in performance between the two machines. As t_{div} approaches 1 (or 16), arrays become asymmetric and deterioration in performance occurs on the DECmpp-12000.

To illustrate how the DECmpp [3] inefficiently maps asymmetric arrays to the 8K physical processor grid (PE grid) of 128×64, we first consider a worst-case example for our application. In this case, illustrated in Figure 4A, $T = 17$, $t_{\text{div}} = 1$, and $T - t_{\text{div}} = 16$. This results in an array shape of $(1 : 2^1, 1 : 2^{16})$. The DECmpp operating system uses a cut and stack method which maps two-dimensional arrays column-to-row. Column one of this Fortran array, which contains two elements, is mapped to row one of the PE grid (along PE x in Figure 4A). Column two of the Fortran array is mapped to row two of the PE grid. When the number of columns in the Fortran array exceeds the number of rows in the PE grid, which is fixed at 2^6, the remaining columns in the array are "wrapped around" into memory. In our worst-case mapping example, the first 2^6 columns of the Fortran array are mapped to dimension PE x, and the remaining $2^{16} - 2^6 = 2^{10}$ columns of the array are wrapped or layered into memory.

Another view of a worst-case example is illustrated in Figure 4B. In this case, $t_{\text{div}} = 16$ and the resulting Fortran array has shape $(1 : 2^{16}, 1 : 2^1)$. Part of column one of this Fortran array, which contains 2^{16} elements, is mapped to row one of the PE grid, which has a capacity of 2^7 elements. When the capacity of PE x is reached, the remaining elements are wrapped into memory. To map this Fortran array of shape $(1 : 2^{16}, 1 : 2^1)$ requires $2^{16} - 2^7 = 2^9$ layers.

The best-case mapping example for our application occurs when $t_{\text{div}} = 7$. A nearly symmetric two-dimensional array results from this value of t_{div}. Figure 4C illustrates how the resulting Fortran array of shape $(1 : 2^7, 1 : 2^{10})$ is mapped in 2^4 layers.

In comparison, the CM-2 arranges array elements "horizontally". Arrays of multidimensional rank are mapped as a one-dimensional array across processors, one element per processor. This approach fills up the processor grid so as to maximize the number of physical processors in use and minimize virtual processor looping [5].

In summary, performance of the binomial model on the DECmpp-12000 is sensitive to array shape. When the two-dimensional Fortran arrays used to express stock price or volatility in the model are asymmetric, data are inefficiently distributed to the fixed processor grid of the DECmpp-12000. DECmpp mapping directives can be specified to change the default mapping, but we cannot take advantage of compile-time directives for this application. The Fortran array dimensions of version one of our model are defined by t_{div} which is accessible to the model only at run-time.

Strategies for Improved Model Performance

To address the load balancing issue described above, we defined a second version of the binomial model based on static, one-dimensional Fortran arrays. This approach improves load balancing by completely filling the 8K physical processors of the DECmpp. In a third form of the model we combine a one-dimensional array model with a data layout strategy to make use of in-processor arrays. This implementation combines load balancing with reduced communication for improved performance.

Load balancing

In the second version of our model, we represent the the 2^T points in the binomial lattice in a Fortran array of size $(1 : 2^T)$. Figure 5 illustrates this one-dimensional array model for $T = 17$ and $t_{\text{div}} = 2$. In stage 2 of the model each subtree, one for each node in the lattice at time t_{div}, is mapped to a section of the array of size $2^{T-t_{\text{div}}}$. The first element of each array segment corresponds to a node at the time of dividend. From this initial state, the volatility and price lattice evolves forward in time. We use the Fortran 90 intrinsic function *eoshift* inside a loop to calculate, then to communicate values representing up/down moves in price and volatility through the array section.

The DECmpp maps one-dimensional arrays to the processor grid in raster-scan fashion. For a model of $T = 17$ periods, the 2^{17} one-dimensional array completely fills the 128×64 physical processors of the 8K DECmpp, then is layered into memory. By using a one-dimensional array to represent the binomial lattice, we limit the number of required layers to 2^4 for any value of t_{div}. Program performance remains constant for all values of t_{div}. Version two of our model, based on static one-dimensional arrays, is similar on both parallel machines, and an order of magnitude faster than the sequential version (Figure 6).

Reduced Communication

In the third version of our model, we combine one-dimensional arrays with in-processor arrays to provide load balancing and reduced communication. We use a one-dimensional static array to represent the first 8K nodes of the binomial lattice, and the *eoshift* function to completely fill the 8K physical processors of the DECmpp. Depending on the value of t_{div}, the model may be in either stage 1 ($t_{\text{div}} < 13$), or in stage 2 ($t_{\text{div}} > 13$). Once the 8K processor grid is filled, all further movements of stock price and volatility are expressed in local arrays. Each processor calculates up/down movements in stock price and volatility and assigns the result of the computation to an element of a serial array stored in local processor memory. Reduced communication improves model performance by two orders of magnitude over the sequential version (Figure 7).

Results

We implemented three forms of the binomial pricing model and compared performance between the CM-2 and DECmpp-12000. The first version of our model expresses the two-dimensional structure of a binary tree in two-dimensional Fortran arrays. We used the value of (t_{div}), known only at runtime, to define the second dimension of the array. This approach is straightforward and allows us to easily track up/down movements in the lattice and determine percentages of early exercise. The DECmpp inefficiently maps asymmetric arrays to the fixed processor grid, and performance is slower in some cases than a SUN4 IPC, 25 MHZ workstation. CM-2 performance is not sensitive to asymmetric arrays. Performance is an order of magnitude faster than the SUN4 sequential model, which runs in approximately 4.0 seconds (Figure 3).

The second version of our model is based on one-dimensional static arrays. The DECmpp efficiently maps one-dimensional arrays, improving load balancing and performance. A one-dimensional array model on the DECmpp performs similarly to one and two-dimensional array models on the CM-2. This version of the model is an order of magnitude faster than the sequential version (Figure 6).

The third version of our model combines one-dimensional arrays with in-processor arrays. One-dimensional arrays provide load balancing—all 8K physical processors of both machines are used. Once the first 8K nodes of the binomial lattice are mapped to the 8K physical processors, all future stock/volatility movements are represented in in-processor arrays. This approach reduces inter-processor communication and improves program performance. Figure 7 summarizes model performance for a sequential Fortran model running on a workstation (SUN4 25MHZ) and three versions of the model running on the DECmpp. Our third model implementation approaches two orders of magnitude improved performance over the sequential version.

Discussion and Conclusion

Our description of three increasingly refined models for stock option pricing on parallel supercomputers demonstrates a methodology for application development on parallel supercomputers. Our approach is based on integrating software, data decomposition, and algorithms to solve a real world application problem.

The binomial pricing model presents load balancing and communication issues that must be addressed to achieve high performance on the CM-2 and DECmpp-12000. Substantial load imbalances for the first several periods of the model are inherent to the binomial model, yet the most important load balancing issue in this application is due to the way model arrays are mapped by the compiler to physical processors. In versions two and three of our binomial pricing model, we observe similar performance between the CM-2 and DECmpp. Only when asymmetric two-dimensional arrays are involved does the DECmpp fail to perform on the level of the CM-2.

We speculate that the compiler design of the two machines was influenced by the network topology. The CM-2 is based on a hypercube topology and allows expression of flexible or general meshes. The CM-2 compiler efficiently maps one and two-dimensional arrays, including asymmetric shapes, to the physical processors. The physical processor grid of the DECmpp presents a less flexible topology and is perhaps the reason why the current software is not designed to efficiently map asymmetric two-dimensional arrays. We emphasize that the problem we have seen with dynamic asymmetric arrays is a feature of the current software, not the hardware, of the DECmpp-12000. Future generations of software will likely solve this type of problem.

High performance option pricing models running on parallel supercomputers provide a needed tool for investigating how different pricing models perform over a long period of time and under varying conditions. Model parameter estimation based on optimization techniques holds great promise for improving pricing model accuracy, but this approach is possible only on the most powerful computers. We expect to see the rate of adoption of parallel supercomputing technology in finance to accelerate as the relative advantage of parallel pricing models becomes better understood. Traders and financial managers require models that can price a single stock in fractions of a second, and investment portfolios in minutes, with accuracies within a few cents of the true market price. Running pricing models on parallel supercomputers make this a near-term possibility.

References

[1] Black, F. and M. Scholes. The Pricing of Options and Corporate Liabilities. *Journal of Political Economy*, May–June 1973, 637.

[2] Finucane, T. 1991. Binomial Approximations of American Call Option Prices with Stochastic Volatilities. to be published in *Advances in Futures and Options Research.*

[3] MasPar Computer Corporation, 1991. *Maspar Fortran User Guide*, Version 1.1, Revision A2, pp. 2–9.

[4] Mills, K., Vinson, M., and G. Cheng. 1992. A Large Scale Comparison of Option Pricing Models with Historical Market Data. *The Fourth Symposium on the Frontiers of Massively Parallel Computation, October 19-21, McLean, Virginia. IEEE Computer Society and NASA GSFC.* SCCS-260. 7 pps.

[5] Thinking Machines Corporation. 1989. *CM Fortran Reference Manual.* Version 5.2-0.6. pp. 368.