An Interactive Visualization Environment for Financial Modeling on Heterogeneous Computing Systems*

Gang Cheng[†‡] Kim Mills[‡] Geoffrey Fox[†‡]

Abstract

Financial modeling represents a promising industry application of high performance computing. In previous work, parallel stock option pricing models were developed for the Connection Machine-2 and DECmpp-12000. These parallel model run approximately two orders of magnitude faster than sequential models on high-speed workstations. To further develop this application, a portable, workstation based, interactive visualization environment was developed for a heterogeneous computing environment. Application Visualization System (AVS) was used to integrate massively parallel processing, workstation based visualization, an interactive system control, and distributed I/O modules. A preliminary performance analysis of this distributed model is discussed.

1 Introduction

Advances in parallel computing systems and network technology provide new opportunities for implementing computationally intensive applications. Applications that integrate modeling and simulation with large scale information processing often require an interactive graphical user interface (GUI) in a real-time computing environment. Most GUIs are event-driven, and serial in nature, making them unsuitable for parallel systems. On the other hand, visualization tools for parallel systems often require special hardware support, or are developed for a specific hardware/software system and are not portable. Parallel computing environments of the future will likely be based on networked computing resources varying in size and architecture. Large applications will be decomposed into subproblems, and distributed to appropriate nodes within the network. To use these systems, we need portable, interactive, visualization tools in a parallel computing environment.

In this study, we purposefully use a diversity of network resources to demonstrate the integration power of Application Visualization System (AVS) software. We couple multiple computational modules with network based visualization, interactive system control, and distributed I/O. Our heterogeneous computing system combines network connected single instruction multiple data (SIMD) and multiple instruction multiple data (MIMD) parallel architectures with high performance workstations. For a stock option pricing application, we describe system integration issues and outline a performance model for combining visual and scientific computing in a heterogeneous computing environment.

*This study was supported in part by the Office of the Vice President for Research and Computing at Syracuse University, and Corporate Partnership funding from Digital Equipment Corporation.

[†]School of Computer and Information Science, Syracuse University, Syracuse, NY 13244

[‡]Northeast Parallel Architectures Center, Syracuse University, Syracuse, NY 13244

2 Stock Option Pricing Models

Stock option pricing models are used to calculate a price for an option contract based on a set of market variables, (e.g. exercise price, risk-free rate, time to maturity) and a set of model parameters. Model price estimates are highly sensitive to parameter values for volatility of stock price, variance of the volatility, and correlation between volatility and stock price. These model parameters are not directly observable, and must be estimated from market data.

We use a set of four option pricing models in this study. Simple models treat stock price volatility as a constant, and price only European (option exercised only at maturity of contract) options. More sophisticated models incorporate stochastic volatility processes, and price American contracts (option exercised at any time in life of contract)[1][2]. These models are computationally intensive and have significant communication requirements. The four pricing models are: BS – the Black-Scholes constant volatility, European model; AMC – the American binomial, constant volatility model; EUS – the European binomial, stochastic volatility model; and AMS – the American binomial, stochastic volatility model.

In previous studies, we developed serial and data parallel versions of these models, compared model prices with historical market prices, and evaluated model performance and parallel software issues [3][4]. Stochastic volatility models tend to price options more accurately than simpler models. Parallel models on the Connection Machine 2 (CM-2), Connection Machine 5 (CM-5), and DECmpp-12000 ran 100 times faster than sequential models on high speed workstations. Using optimization techniques for model parameter estimation holds great promise for improving model accuracy.

Analytic models are useful tools in the financial market, but require expert interpretation. To further evaluate and optimize pricing models to run in a parallel computing environment, we combine high performance computing modules for real-time pricing with real-time visualization of model results and market conditions, and a graphical user interface allowing expert interaction with pricing models. We envision a market expert using such a system to start and stop a set of models, adjust model parameters, and call optimization routines according to dynamically changing market conditions.

3 System Configuration

Our heterogeneous computing system for stock option pricing consists of four compute nodes, a home machine, and two file server machines. All workstations, including the front-ends of the DECmpp-12000 and CM-5, are connected by a 10MBit/second Ethernet based LAN.

The home machine is a IBM RS/6000 workstation with a 24-bit color GTO Graphics Adapter. The AVS kernel and system modules run on this machine which displays the graphical user interface, renders graphical output, and monitors user run time interaction. The user logs into this machine, sets up and interacts with the system through keyboard, mouse, and other I/O devices.

The four option pricing models run on remote compute nodes: BS model on a DEC5000, AMC model on a SUN4, EUS model on a CM-5 and AMS on a DECmpp-12000(SX). Each remote compute node has its own I/O capability. Our DECmpp-12000 is a massively parallel SIMD system with 8192 processors. Each RISC-like processor has a control processor, forty 32-bit registers, and 16 KBytes of RAM. All the processor elements are arranged in a rectangular two-dimensional grid and are tightly coupled with a DEC5000 front-end workstation. The theoretical peak performance is 650 Mflops DP. Our CM-5 is

a parallel MIMD machine with 32 processing nodes. Each processing node consists of a SPARC processor for control, four proprietary vector units for numerical computation, and 32 MBytes of RAM. The control node of the CM-5 is a SUN4 workstation. The theoretical peak performance is 4 Gflops. Sequential compute nodes include a DEC5000 and a SUN4. The DEC5000 performs at 6.8 Mflops, and has 16 Mbytes memory. The SUN4 runs at 4.3 Mflops and has 32 Mbytes memory.

The user interface runs on a remote SUN4. This machine combines user runtime input (model parameters, network configuration) with historical market databases stored on disk, and broadcasts this data to remote compute nodes. System synchronization occurs with each broadcast.

A second IBM RS/6000 is used as a file server for non-graphical output of model data. In this application, model prices calculated at remote compute nodes and corresponding market data are written to databases for later analysis.

In summary, the heterogeneous computing system illustrated in Figure 1 provides distributed computing, memory, and input/output for the stock option pricing application.

4 System Integration

Our heterogeneous computing system integrates diverse functions–computation, visualization, and system control over a diverse set of hardware. We use a mix of programming languages on the remote compute nodes–Fortran77 on the DEC5000, C on the SUN4, CM-Fortran on the CM-5, and MPL (data parallel C) on the DECmpp-12000. AVS integrates visualization, networking functionality, and computation. At the operating system level, all remote modules are compiled and linked as stand-alone programs. Input and output ports are defined in modules by the programmer using specific library routines provided by AVS. Each module represents a process. Inputs and outputs between remote modules are implemented via socket connections.

There are two source of input data: historical market data read from disk files, and runtime input of model parameters by the user through a GUI. Output from all four models is rendered in a graphics window, displayed numerically in a shell window, and written to a database by the file server. Figure 2a illustrates control flows on the home machine. There are two types of events, GUI events (e.g. changing network configurations, window displays) and modeling events (e.g. changing model parameters, controlling model execution). Figure 2b illustrates how market data and model parameters are combined before broadcast to the compute nodes. System synchronization occurs at this step.

Figure 3 illustrates the GUI for managing user runtime input and output, and the system configuration. Runtime input includes user defined model parameters and system execution styles. Outputs include 2-dimensional displays of model and market prices calculated by the compute nodes. The system configuration includes choice of pricing models, network configurations and interface layouts.

Pricing models are extremely sensitive to model parameters for implied volatility(σ), variance of stock volatility(ξ) and correlation between stock price and its volatility(ρ). These parameters may be read from data files (historical estimates), calculated just prior to running the pricing model (by optimization), or defined at run time (expert user). Dial widgets are used to set initial values of σ for each model. Slider widgets are used to set ξ, and ρ. One-shot widgets are used to run the system in single-step execution mode and call optimization functions on remote compute nodes.

Graphical output is rendered in AVS Graph Viewers. Using the network editor running

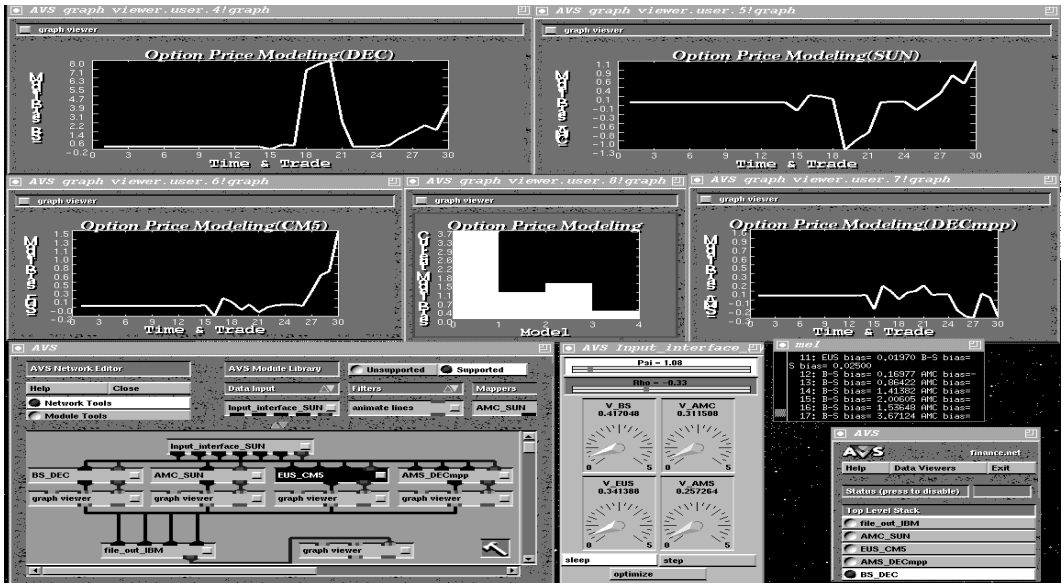

FIG. 3. *The Graphical User Interface on the Home Machine*

on the AVS kernel, we can configure a network at runtime by graphically connecting and disconnecting the data flows between remote compute nodes and the Graph Viewer.

5 Preliminary Analysis of Performance Requirements

A generalized performance model of our networked system is shown in Figure 4. In one complete modeling cycle, starting with broadcast of new data from M_I, and ending with renderings on M_h, let t_{calc} be the summed calculation time on all machines and t_{comm} be the summed communication time in the system. Let T_d, T_s be the total time to complete such a cycle in the distributed system and in a single-machine system, respectively. Other symbols are defined as follows: c_{k} – communication time of required message passing from M_I to M_k; d_{k} – communication time of required message passing from M_k to M_h; m_{k} – calculation time of performing required computation on M_k; r_{k} – calculation time of performing required rendering by P_k on M_k; t'_{calc} – the total calculation time on a single-machine system; m'_{k} – calculation time of performing required computation on a single machine system.

In a single-machine system, $M_I = M_h = M_1 = M_2 = \cdots = M_n, t_{\text{comm}} = 0$ and $T_s = t'_{\text{calc}} = \sum_{k=1}^{n} (m'_k + r_k)$. In the distributed system based on TCP/IP protocol, we observe: sequential message passing from M_I to M_k, although the sending order can be scheduled; execution of computing process on M_k upon receiving new message from M_I; and sequential rendering of process P_k on a single-processor machine M_h, where each P_k is activated only after receiving a new message from M_k, and no other $P_j (j \neq k)$ is running on M_h.

Calculation and communication among different machines in the system are pipelined. We can overlap the following processes: (1) c_k with m_j, d_j, and r_j; (2) m_k with d_j and r_j; and (3) d_k with r_j. Figure 5 illustrates this overlap of processes in system of $(n = 2)$ machines. For each machine M_k, the total time to complete a cycle is $T_d = c_k + m_k + d_k + r_k + i_k$, where i_k is the idle time of M_k in the cycle, excluding the waiting time r_k for the rendering on M_h. Thus, $nT_d = t_{\text{comm}} + t_{\text{calc}} + t_{\text{idle}}$, where $t_{\text{comm}} = \sum_{k=1}^{n}(c_k + d_k)$, $t_{\text{calc}} = \sum_{k=1}^{n}(m_k + r_k)$, and $t_{\text{idle}} = \sum_{k=1}^{n} i_k$.

The speed-up S of a distributed system over a single-machine system is:

$$S = \frac{T_s}{T_d} = \frac{n}{\frac{t_{\text{calc}}}{t'_{\text{calc}}} + \frac{t_{\text{comm}}}{t'_{\text{calc}}} + \frac{t_{\text{idle}}}{t'_{\text{calc}}}},$$

Speedup depends on four components: (1) n, the number of distributed computing modules; (2) $t_{\text{calc}}/t'_{\text{calc}}$, the ratio of total calculation time on the distributed system to calculation time on a single-machine system; (3) $t_{\text{comm}}/t'_{calc}$, the ratio of total communication time in the distributed system to calculation time on a single-machine system; and (4) $t_{\text{idle}}/t'_{\text{calc}}$, the ratio of total idle time in the distributed system to calculation time on a single-machine system. As Figure 5 illustrates, for a given application on a given system (fixed c_{k}, d_{k},m_{k},r_{k}) t_{idle} depends on the order of messages sent from M_I to M_k, making this a scheduling problem. In the best case, $t_{\text{idle}} = \sum_{k=1}^{n-1} c_k(n-k) + \sum_{k=1}^{n} a_k r_k$, where $\{a_1, a_2, \ldots, a_n\}$ is a permutation of the set $\{0, 1, \ldots, n-1\}$.

Timings observed for the system configuration illustrated in Figure 1 are listed in Table 1. According to our analysis, expected $T_d = c_1 + m_1 + d_1 + r_1 + r_2 + r_3 + r_4 =$ 0.017 + 0.015 + 0.1 + 4*0.9 = 3.642 seconds (assuming the scheduled order is c_1, c_2, c_3, and c_4). This value will vary because the system runs under a resource-and-time-sharing environment.

To compare the performance of distributed and single-machine systems, we used a SUN4 single-machine system. Summarizing timings listed in Table 1, $T_s = 0.015 + 0.085 + 4.05 + 4.25 + 4^*0.9 = 12.0$, and we calculate an expected speed-up of $S = 12.0/3.642 = 3.3$.

6 Discussion and Conclusion

The availability of high speed networks, combined with advances in computational power, will support a new set of applications designed for high performance distributed computing. In the finance industry, computational power alone is not sufficient. We see a need for interactive computing environments based on high performance computing, which take advantage of the intuition and experience of market experts. We focus here on the functionality required to develop finance industry applications on heterogeneous, distributed systems. These functions include real-time modeling, visualization, and user control, all in the context of dynamically changing market conditions.

Further work is needed in the areas of software for system integration, models for performance prediction, and network technology for connecting heterogeneous computing systems. A high-end graphics workstation with hardware supported rendering capability is an essential system component. We used a 10 Mbit/second Ethernet network connection, but only 1-2 Mbit/second of its capacity is available to support the application. Improvements in the host/system interface are clearly needed. We outlined the elements of a performance model, and in the future will take advantage of on-going research [5] in performance prediction tools. These tools will help us to analyze application and system input parameters, and predict performance for the application on a specified hardware system.

In conclusion, we demonstrated the ability of commercially available AVS software to integrate a diversity of network resources. Heterogeneous computing environment supports an attractive software environment for financial modeling applications. This highly portable software environment runs on massively parallel computers, was implemented with relatively small programming effort, and allows rapid prototyping. Larger, more complex application problems will require networks of machines that integrate functions such as scientific computing, visualization, database services, real-time decision-making, and large memory distributed over a network.

Acknowledgement: We would like to thank Marek Podgorny and Tomasz Haupt for useful discussions about AVS, and Peter Crockett for systems support.

References

[1] F. Black, and M. Scholes. "The Pricing of Options and Corporate Liabilities," *Journal of Political Economy*, 81, 1973, 637-59. 1973.

[2] T. Finucane, "Binomial Approximations of American Call Option Prices with Stochastic Volatilities," to be published in *Journal of Finance*. 1992.

[3] K. Mills, M. Vinson, and G. Cheng, "A Large Scale Comparison of Option Pricing Models with Historical Market Data," in *The 4th Symposium on the Frontiers of Massively Parallel Computation*, Oct. 19-21, 1992, McLean, Virginia.

[4] K. Mills, G. Cheng, M. Vinson, S. Ranka, and G. Fox. "Software Issues and Performance of a Parallel Model for Stock Option Pricing," in *The Fifth Australian Supercomputing Conference*, Dec. 6-7, 1992, Melbourne, Australia.

[5] S. McDermott, W. Johnston, B. Ware, M. S. Lynn, and J. A. Graniero. "NYNET: A High Performance Distributed Computing Corridor," NYNEX/New York Telephone/Syracuse University/Cornell University/ Rome Labs. December, 1992. SCCS draft technical report.