From the Proceedings of the 1996 IEEE/IAFE Conference on
Computational Intelligence for Financial Engineering

New Computational Architectures for Pricing Derivatives

Roy S. Freedman
Inductive Solutions, Inc.
380 Rector Place
New York, NY 10280
Roy@Inductive.com

Rinaldo Di Giorgio
Sun Microsystems, Inc.
1 New York Plaza — 35th Floor
New York, NY 10004
Rinaldo.DiGiorgio@East.Sun.com

Topics: Financial Computing Environments, Structured Securities, Stochastic Processes

1. The Problem and Previous Work

The problem that concerns us is the cost-effective computation of the expected value of a derivative security. In a previous paper [3], we showed that one should not separate the method of computing the expected present value of a structured security from its ultimate computing topology. In particular, we concluded that the network infrastructure is as least as important a factor in cost-effective computing as the algorithm design and its processor implementation.

In the following sections, we extend this work by further investigating the network issues involved with deploying sophisticated derivative analytics on a modern computer network. We show that same technology that can be used to exploit parallelism can also be used to deploy sophisticated analytics to authorized users in a cost-effective way that is secure, easily updatable, and relatively machine independent.

We put these ideas to practice by extending our derivative computation system used in [3]. That system was used to compare the derivative valuations on various computing network architectures. The Benchmark Problem computes an American put option under various interest rate scenarios using a combination of Binomial Lattice and Monte Carlo methods. We rebuilt the system as an executable Derivative Calculator "applet." It is currently viewable on any Java-enabled Web Browser on the World Wide Web (at www.zeitgeist.com), independent of the computer processor or operating system. It also exploits parallelism: it will use any processor available on its local host to to automatically speed itself up.

2. Derivative Valuation Algorithms

The computational problem of deriving the expected value and other statistics of a derivative security f at time T_0 can be solved when the underlying security S and derivative security f are stochastic processes If it is known that the derivative pays out f_T at time T, its expected discounted value in a risk-neutral world is

$$\text{Expected Present Value} = E[e^{-r(T-To)} f_T] \qquad (1)$$

Here, r is the average instantaneous risk-free interest rate between $t=T_0$ and $t=T$. When the underlying S follows an Ito process, and if the derivative is some differentiable function of S and t, f=f(S,t), then by Ito's Lemma, f also follows an Ito process:

$$dS = \mu_{(t,S)}\, dt + \sigma(t,S)\, dz \qquad (2S)$$

$$df = (\partial f/\partial S)dS + [\partial f/\partial t + (1/2)\sigma^2(t,S)(\partial^2 f/\partial S^2)]\, dt \qquad (2f)$$

$$= [\mu_{(t,S)}(\partial f/\partial S) + \partial f/\partial t + (1/2)\sigma^2(t,S)(\partial^2 f/\partial S^2)]dt + \sigma(t,S)(\partial f/\partial S)\, dz$$

In practice, there are three standard methods that have different computational consequences for computing European-style (the holder has no decisions to make during its life) and American-style (the holder has decisions to make during its life) derivatives:

Method 1. Analytic Approximation for Constant Parameters. If the derivative is a European-style derivative, and the Ito process in Equations (2S) and (2f) has constant $\mu_{(t,S)} = \mu$, constant $\sigma(t,S) = \sigma$, and constant interest rate, then computationally nice expressions exist for the derivative security. In general, G depends on the efficiency of computation of special functions (like the normal distribution).

Method 2. Recombining Lattice-Type Computations. If the Ito process in Equations (2S) and (2f) has constant $\mu_{(t,S)} = \mu$, constant $\sigma(t,S) = \sigma$, and constant interest rate, then the valuation of a European- or American-style derivative is usually computed by simulating the up-down price movements in a recombining binomial lattice. Hence, at time $T_0 + i\Delta t$, the price of the underlying may be any of a set of i+1 values: $Su^j d^{i-j}$; i=0,..N; j = 0,..,i. A recombining binomial lattice must compute and store a total of (N+1)(N+2)/2 prices for the underlying and derivative. For N=500, this requires approximately 10^5 computations, and represents much greater computational overhead than Method 1. This method may require several orders of magnitude of computation than Method 1.

Method 3. Non-Recombining Simulation. If f is a European-style derivative, and the Ito process in Equations (2S) and (2f) has non-constant $\mu_{(t,S)}$, non-constant $\sigma(t,S)$, and possibly non-constant interest rate, then Method 2 may not work because the up values and down values of a price movement may not combine: a sequence of up movements followed by down movements are not valued the same as the down movements followed by the up movements. In Method 3, where recombining is not possible, a representative random "Monte Carlo sample" can be computed directly from the random sample of prices, and not from the complete set of prices. In Method 3, the time required to compute the value of a derivative depends on the number of discrete time units N and the number of Monte Carlo samples M generated for f.

These standard methods are all discussed in [5].

3. Impact of the Network Architecture on Computation

There are several ways of incorporating additional computing power to speed up the computation of derivatives, and the "obvious" answer of "getting a faster computer" may not be obvious, or may even be "obviously wrong." For example:

The alternative computing topologies considered here are (listed in order of increasing cost):

1. Workstations
2. Faster Workstations
3. Supercomputers
4. Networked ("Clustered") Systems, that could contain both workstations, supercomputers, or both.

The derivative evaluation problem is typically more compute-intensive than memory intensive. For example, that during the time that one computer is sending another computer 1 Megabyte of data, the other computer could have done over 100 million floating point divides [3]. This latency only gets worse for memory-intensive computation. On the other hand, some implementations of Method 2 may send large lattices around a network : for N=500, this would amount to about 1 Megabyte.

The model builder should be concerned with building a parallel version of the algorithm. The idea here is to implement the algorithm in such a way so that n-processors can solve the problem in (1/nth) the time as one processor. (Note that this is a type of algorithm optimization that is orthogonal to that produced by compiler optimizations.) Some of these issues have been investigated and compared for derivative evaluation in [1] and [2].

Method 3 is a problem that can be solved with "weakly-coupled parallelism": for example, Monte Carlo samples can be generated on two different processors, f can be evaluated, and the discounted expected value computed on a third processor. The first two processors are totally independent of each other (assuming they both do not generate the same set of "random" samples). Consequently, one can optimally expect a 2:1 speed-up (minus the communication overhead discussed above). Weakly-coupled applications require relatively little effort in creating a parallel speed-up.

Method 2 is a problem that can be solved with "fine-grained parallelism." It can be shown that each computation along the diagonal of the lattice can be done in parallel. Consequently, if N processors are available, instead of performing (N+1)(N+2)/2 sequential computations, a fine-grained parallel implementation requires only (N+1) sequential computations. Fine-grained parallelization usually requires more effort in modifying the algorithm than weakly-coupled parallelization.

4. Computing on the World Wide Web

The World Wide Web is a hypertext-based universal computing environment that cleverly exploits the TCP/IP computer addressing schemes in an easy to use interface: a Web Browser. A Web page typically contains links to several computers at once: it is the Browser's job to process these links and display them as text, audio, video, or images.

Sophisticated computing can be brought to the Web by special Web-based programming languages, of which Java [4] is the most popular. Java is an object-oriented language that supports parallel processing (multithreaded execution) and can easily identify other computers (identified by their Uniform Resource Locator) on the Internet. It is possible to write a program in Java that can execute, without modification, on a single processor or a multiprocessor. These "applets" are downloaded from a server, in much the same way that an image or text document is downloaded and parsed by the Browser. Thus, for computation, the primary advantage that Web computing offers over conventional networks are that parallel algorithms may be specified in a relatively machine independent way.

From a deployment perspective, Web-based analytics provide a cost-effective way that is easily updatable, since the latest versions of the analytics (maintained on the server) are what is downloaded and executed on the client. This downloading of executable code mechanism is secure (ie, virus-free), if the executing language and Browser enforces security restrictions. Among the mandatory security restrictions are (i) not allowing the use of pointers; (ii) not allowing the reading of files on the client; (iii) not allowing the writing of files on the client; (iv) not allowing the program to access the client operating system.

5. The Derivative Calculator

This Benchmark Problem that evaluates an American put option problem used in [3] was rehosted to a Web-based infrastructure. It is currently hosted at http://www.zeitgeist.com and is also accessible through a number of other sites, such as http://www.gamelan.com. The initial page is shown in Figure 1.

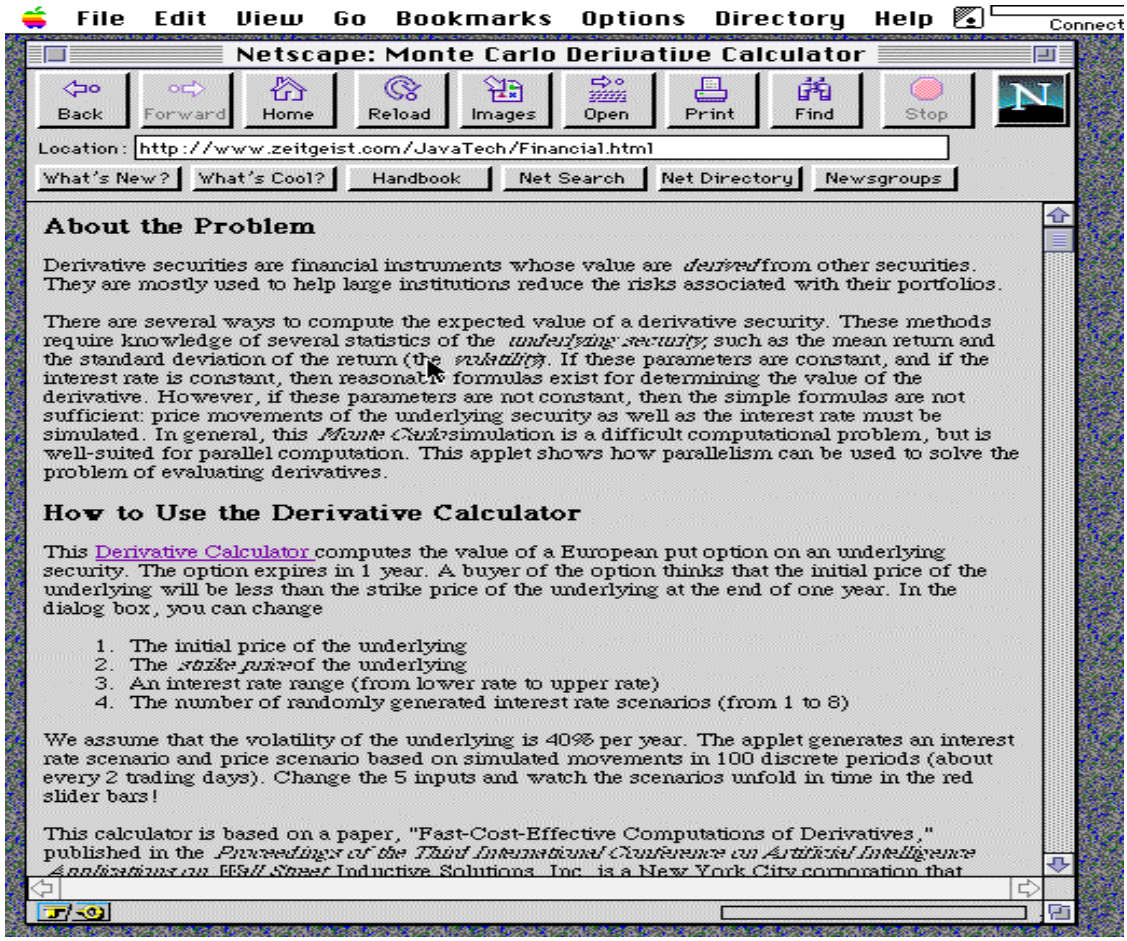

Figure 1. Derivative Calculator (Page 1)

The Benchmark Problem for the American put option used in [3], uses the most compute-intensive aspects of Method 2 and Method 3. S follows an Ito process with constant μ and σ. The user supplies these parameters, as well as the current price, strike price, time to expiration in months, and number of time increments N. The program builds a standard recombining lattice [5] of (N+1)(N+2)/2 nodes to find the expected value of f. Next, we vary the average instantaneous interest rate r by taking a set of M Monte Carlo samples. Thus the value of f is the sample average of M lattice evaluations. The algorithm was implemented to support the weakly-coupled parallelism of Method 3.

The first page of the Derivative Calculator explains the problem; the actual derivative calculator (hyperlinked to the first page) is shown in Figure 2.

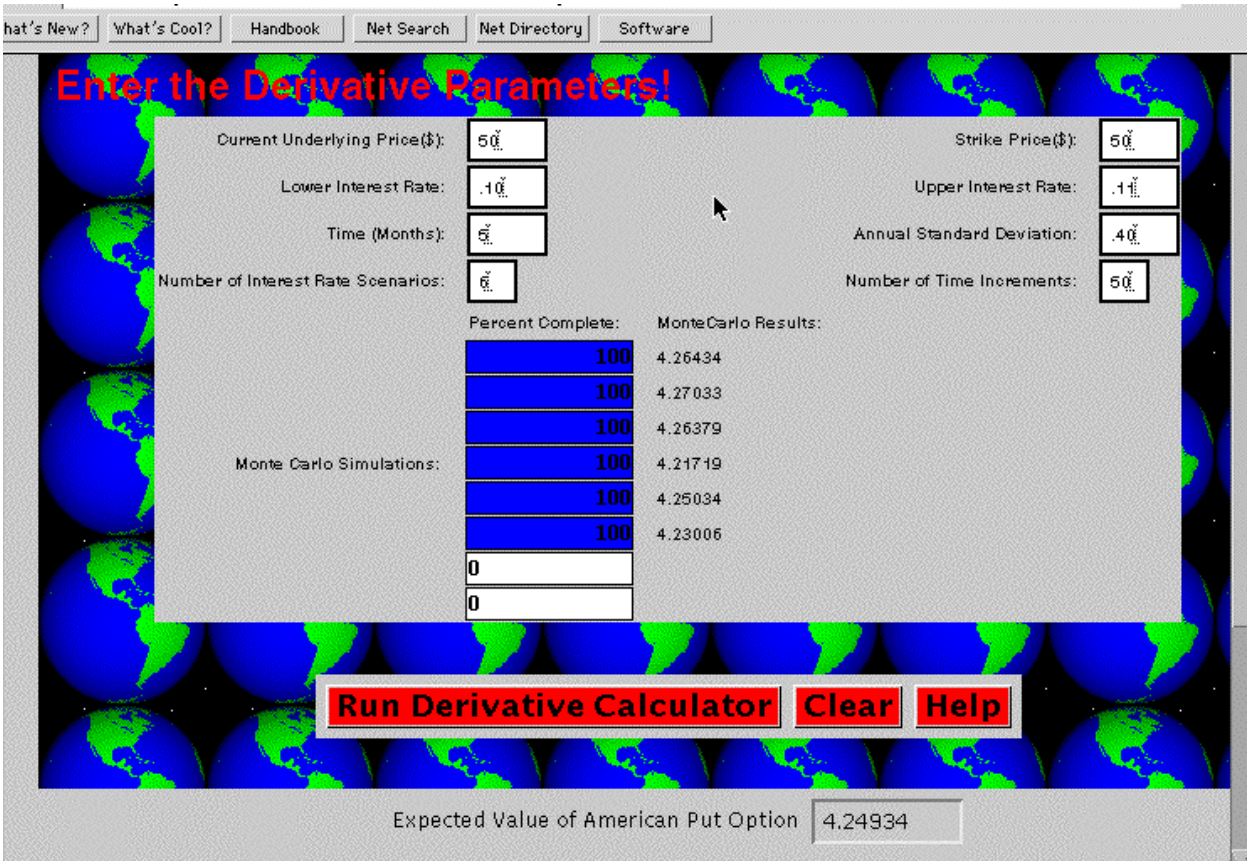

Figure 2. Derivative Calculator (Page 2)

The last two parameters — the number of time increments N, and the number of Monte Carlo Samples M — are the most significant as far as performance is concerned.

The parameter N affects the granularity and precision of the evaluation: a general rule is that that the larger the N, the more accurate the evaluation [5]. Of course, the penalty for large N is the requirement for large memory. In the original Benchmark Problem [3], N was set to 100. In the Web-based calculator, N is a parameter that we fixed to range between 5 and 100. The reason is that we cannot determine a priori the memory on the client. Client computers with more memory would permit a larger N.

The parameter M effects the variance of the Monte Carlo estimator. The goal of the original Benchmark was to ascertain the tradeoffs between this M and the degree of multiprocessing: each lattice can be evaluated on a separate processor, if enough processors were available. If not, then the lattice evaluations would be time-shared across independent parallel processes (threads). In the original Benchmark Problem [3], M was set to 1000. In the Web-based calculator, we realized that the average user did not have a powerful multiprocessor available; consequently, we deliberately fixed M to range from 1 to 8 in order to dynamically show the user the progress of each lattice evaluation (as indicated by the percentage-moving sliders). Each lattice evaluation is implemented as a separate (parallel processing) thread.

6. Conclusions

The deployment of derivative analytics independent of the computer processor or operating system is a reality. We have demonstrated this in a way that also exploits parallelism: the implementation will use any processor available on its local host to to automatically speed itself up.

There still remain computational tradeoffs involved with deploying sophisticated derivative analytics on a modern computer network . Internet-based solutions exploit parallelism, and can be used to deploy sophisticated analytics securely and cost-effectively, as long as the computational requirements of the client can be either known a priori, or parameterized in a reasonable way.

7. References

1. Cagan, L., Carriero, N., and Zenios, S., "A Computer Network Approach to Pricing Mortgage-Backed Securities," **Financial Analysts Journal**, March-April 1993.

2. Clewlow, L., and Carverhill, A., "Quicker on the Curves." **Risk**, 7(5), May 1994.

3. Freedman, R.S., DiGiorgio, R., "Fast Cost-Effective Computations of Derivatives," **Proceedings of the Third International Conference on Artificial Intelligence Applications on Wall Street**, Software Engineering Press, June 1995.

4. Freedman, R.S., DiGiorgio, R., **Programming with Java**, O'Reilly & Associates (to appear in 1996).

5. Hull, J., **Options, Futures, and Other Derivative Securities**, Prentice-Hall, 1993.