Reprinted from the *Proceedings of the Third International Conference on Artificial Intelligence Applications on Wall Street*, June 1995.

Fast Cost-Effective Computations of Derivatives

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

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

Abstract

The essential idea of this paper is that one should not separate the method of computing the expected present value of a derivative from its ultimate computing topology. In the following sections, we discuss the cost-benefit issues involved with implementing several methods for computing derivative statistics on alternate computing topologies. We show how the choice of topology impacts the computing time for a particular example of a time consuming derivative valuation. We conclude by showing how all these factors can be represented as a case-based expert system, which can be used to help an organization assess its computing alternatives.

1. Background: Algorithm Tradeoffs in Computing Derivatives

We are concerned with the computational problem of deriving the expected value and other statistics of a derivative security f at time T_0. When the underlying security S and derivative security f are modeled as stochastic processes, the problem can be solved by reformulating it as a boundary-value problem: if it is known that the derivative pays out f_T at time T, we just compute its value backwards from the risk adjusted random price movements of the underlying from t=T to t= T_0. The present value of f is just its expected discounted value in a risk-neutral world

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

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 a differentiable function of S and t, f=f(S,t), then by Ito's Lemma, f also follows an Ito process:

$$dS = \mu(t,S)\, dt + \sigma(t,S)\, dz \qquad (2S)$$

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

and f satisfies the Fokker-Plank forward diffusion equation

$$\partial f/\partial t = 1/2(\partial^2/\partial S^2)[\sigma^2(t,S)\, f] - (\partial/\partial S)[\mu(t,S)\, f] \qquad (2FP)$$

given initial condition $S(T_0) = S_0$.

Here S(t) is the probability distribution of the price of the underlying at time t, $\mu(t,S)$ and $\sigma(t,S)$ are the instantaneous drift and standard deviation rates, and dz is a Wiener Process that corresponds to Brownian motion. Note that if we know the probability distributions for S(t), and if we are given boundary conditions for f (which define the derivative), then we can solve (2FP) and derive the probability distribution for f, so that the expected present value of f can be computed from Equation (1).

The above equations are valid for all derivative securities with S as the underlying stochastic variable [4]. A vector form of Equation (2S) and (2f) is valid if S depends on other Ito processes (for example, if μ or σ are Ito processes). Here, the correlations of the underlying processes are additional factors in the dt term in Equation (2f).

Simplifications can be made: if the interest rate r is known to be constant, then it can be shown that the Ito process for $[S\partial f/\partial S)-f]$ does not depend on dz — this "continuous" hedge is "riskless." Hence, in this case, f satisfies the Black-Scholes partial differential equation

$$\partial f/\partial t = rf - (1/2)S^2\sigma^2(t,S)(\partial^2 f/\partial S^2) - rS(\partial f/\partial S) \qquad (2BS)$$

Equation (2BS) can be solved if S(t) is known and the boundary conditions that define the derivative f are provided. For example, a boundary condition for a

European call option is

$$\text{At } t = T,\ f(T) = f_T = \max(S_T - X, 0) \qquad (2CO)$$

In practice, in all but the simplest cases, the price movements of S and f follow stochastic processes that involve substantial amounts of computation. There are three general methods that have different computational consequences for computing European-style derivatives (the holder has no decisions to make during its life) and American-style derivatives (the holder has decisions to make during its life):

Method 1. Analytic Approximation for Constant Parameters. If the derivative is a European-style derivative, and the Ito process in Equations (2S), (2f), and (2BS) has constant $\mu(t,S) = \mu$, constant $\sigma(t,S) = \sigma$, and constant interest rate, then computationally nice expressions exist for the derivative security — the famous formulas derived by Black and Scholes. Analytic expressions also exist for approximating the values of American-style derivatives. In Method 1, the time required to compute the expected value of f is proportional to a constant factor G — the time required to evaluate the formula. 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), (2f), and (2BS) 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. (The lattice is a discrete form of Equation (2S-2f), and is also related to a discrete form of (2FP) and (2BS)). In this method, the time required to compute the value of a derivative depends on the number of time units N, where $N = (T-T_0)/\Delta t$, and Δt is the smallest unit of time considered in the computation. In this method, a sequence of up movements followed by down movements are valued the same as the down movements followed by the up movements. At any given point in time $T_0 + i\Delta t$, the price of the underlying may increase or decrease by an amount u and d with probability p and (1-p) respectively. Hence, at time $T_0 + i\Delta t$, the price of the underlying may be any of a set of i+1 values:

$$S\, u^{j}\, d^{\,i-j} \quad \text{where } i=0,..N;\ j = 0,..,i.$$

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), (2f), and (2FP) 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. Consequently, in evaluating the possible price of S, after N time increments there are 2^{N+1} possible prices (none are recombined as in Method 2; if recombining is allowed, there are only (N+1)(N+2)/2 prices). In Method 3, where recombining is not possible, all 2^{N+1} possible prices must be generated to get the "complete" distribution for the expected value in Equation (1). Pragmatically, this is impossible, since for n=500, this is approximately 10^{150} prices. The alternative here is to create a representative random "Monte Carlo sample" of f so that the expectation in Equation (1) 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. Accuracy in the evaluation of f is a statistical problem relating to the standard error of the estimate of the sample mean. Since it is known that the standard error in computing an expectation is proportional to $M^{1/2}$, reduction of the error by a factor of 2 necessitates increasing M by a factor of 4. Consequently, different "variance reduction" techniques could be employed [2]. Note that in using Method 3, a model for S can depend on other Ito processes: for k processes, a complete set of N time samples would require $2^{k(N+1)}$ computations. Method 2 may require several orders of magnitude of computation more than Method 2.

Methods (1), (2), (3) can also be combined. For example, one can value an American-style derivative with stochastic average interest rate and stochastic average volatility by generating Monte Carlo samples for r and σ as input to a recombining binomial lattice for f. Computational infrastructure is stretched when these three methods are used to value a portfolio of P

derivatives. Consequently, the total amount of computation required for a portfolio is proportional to:

P*G, for Method 1
N*P, for Method 2
N*M*P, for Method 3

and, in general, the computation time for each method corresponds to

Method1 << Method 2 << Method 3.

2. Incorporating More Computing Power

There are tradeoffs in model accuracy and computing time in the three above Methods. These model tradeoffs are further compounded by the computational tradeoffs in alternative computing infrastructure. 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:

> "We have alot of programmers who write C applications. We have alot of Unix workstations, but most are efficiently used all day and all night. Our derivative evaluation application is based on Monte Carlo methods, and we need to improve the accuracy without sacrificing time."
>
> "We need to evaluate our very large portfolio in almost real time. We already have a supercomputer but we could use 2 more. Should we buy another million-dollar parallel processor? We have alot of idle workstations."
>
> "We run a lattice-type American-style valuation application each day on my entire inventory. We can do one evaluation each day. We keep getting more clients. Should I go back to a Black-Scholes formula? My application runs on a PC and I do not understand parallel computation. We have no programmers on staff."

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

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

Their general characteristics are summarized in Figure 1.

Figure 1. Alternate Computing Topologies

	Number of Processors	Speed in MFLOPS	Memory in MBytes	Cost $K
Workstation	1	1-25	32	<10
Faster Workstation	1-4	>25	64	>10
Cluster	>1	>2000	>128	40 -4000
Supercomputer	>1	2000	>128	1000-20000

The problem that we address in this paper is concerned with the cost effective computation of the expected value in Equation (1), with respect to the tradeoffs between Methods 1-3 and the above computing topologies. Note that these alternatives are not mutually exclusive, their boundaries are "fuzzy" and they may be combined.

The essential idea of this paper is that one should not separate the method of computing the expected value in Equation (1) from its ultimate computing topology. Different topologies may be more cost-effective than other topologies. This is a point also made in [1], even though their evaluation was basically concerned with showing the computing potential of the cluster topology, not its cost-benefit tradeoffs with respect to an organization's requirements.

In the following sections, we discuss the cost-benefit issues involved with implementing the above methods for computing Equation (1) on alternate computing topologies. We show how the choice of topology impacts the computing time for a particular example of a time consuming derivative valuation. We conclude by showing how all these factors can be represented as a case-based expert system, which can be used to help an organization assess its computing alternatives.

3. Risk Tradeoffs of Alternative Computing Topologies

The problem is: Given the algorithmic alternatives and parameters G, N, M, P as defined in Section 1, find a computing topology that minimizes the time and cost required for a valid computation. It is convenient to group the costs into the categories of Opportunity Costs, Infrastructure Costs, and Algorithmic Costs. The first two costs are general and may be applied to any kind of alternative topology problem; Algorithmic Costs are specific to derivative computations. From another perspective, these costs can be used to describe potentially new benefits of changing to an alternative topology: if the business benefit does not outweigh the

other costs, then there may be no cost-effective reason to change.

Opportunity Costs. These costs reflect the risks associated with the nature of the routine function of the business. Assessed here are the costs of a late answer, cost of a wrong answer, cost of no answer, and cost of infrastructure breakdown. For example, a fixed income group may require real time evaluation of their entire derivative position 30 minutes before the monthly speech of the Federal Reserve Chairman. If this cannot be done, then there is an opportunity cost.

Infrastructure Cost. These costs reflect the risks associated with maintaining the existing computing infrastructure as well as the additional risks of modifying the infrastructure to a new topology. Assessed here are Client-Server Costs (costs of additional workstations and servers, together with software); Network Costs (costs of network hardware and software); Infrastructure Modification Costs, Runtime Costs; and System Administration Cost.

The cost and benefit tradeoffs can indicate whether "getting a faster computer" presents a good alternative: the network performance impact is almost as great as the computing processing. For example, purchasing a supercomputer may result in slower performance if the network the supercomputer is on is slow or is saturated with traffic. Figure 2 further illustrates the impact of network performance on computation. This table shows, 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. This latency only gets worse for memory-intensive computation. The derivative evaluation problem is more compute-intensive than memory intensive. 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.

Ethernet (Network)	To send 1MB requires (sec)...	No. of float divides on 200MHz chip...
Regular Ethernet	0.56	112,000,000
Fast Ethernet	0.056	11,200,000
OC-3	0.051612903	10,322,581
OC-24	0.006451613	1,290,323
SP2	0.033333333	6,666,667
Tightly Coupled (Parallel Processor Backplane)		
100MB/sec bp	0.01	2,000,000
320 MB/sec	0.003125	625,000
640 MB/sec	0.0015625	312,500
1200 MB/sec	0.00078125	156,250

Figure 2. Network Speed vs. Computation

Algorithmic Costs. These costs reflect the risks associated with maintaining the existing computing algorithm as well as the additional risks of modifying and porting the algorithms so they work on the new topology. Assessed here are the costs of optimizing an algorithm. While many compilers offer one such level of optimization, two other levels of analysis should also be performed. On a macro level, there is a cost-benefit analysis involved in determining the best combination of Methods 1-3. This is essentially the job of the model builder. From a micro perspective, there is a degree of algorithm optimization that is orthogonal to that produced by compiler optimizations. One such optimization is 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. Moreover, new processors actually require "supercomputer style" optimizations (such as loop unrolling, blocking, and memory access patterns) to keep data paths efficient.

At this point the tradeoffs between a weakly-coupled parallelism versus a fine-grained parallelism should be addressed.

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, an algorithm can be configured (or "vectorized" by a skilled programmer) that, at time k, computes the values of S and f in the k+1 nodes on k+1 processors (see Figure 3).

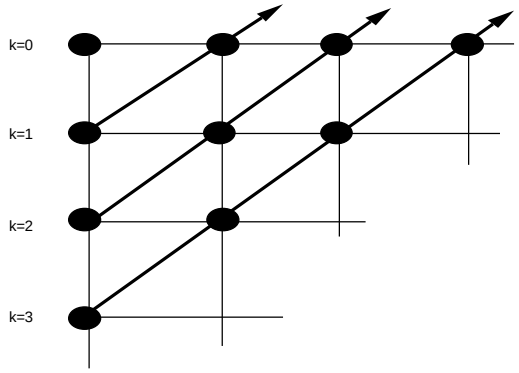

Figure 3. Fine-Grained Parallelism of Recombining Lattice Method

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.

Both weakly-coupled and fine-grained parallelization techniques require a topology to support different parallelization operators. Fine-grained topologies often rely on semaphores, condition variables, and shared memory areas. Some of the operators for the weakly-coupled topologies include:

Broadcast. One processor node sends the same message to other nodes. The simplest broadcast operation is to start running all programs on all nodes.

Scatter. One processor node sends a different message to each node. An example: in Method 3, we can use a scatter operation start running all programs with a different seed to the random number generator.

Gather. Every processor node sends a message to a single member. An example: we gather the Monte Carlo sampled values f for averaging at processor node 0.

Barrier Synchronization. All processor node must reach the same point before any can proceed. An example: in the fine-grained parallelism of Method 2, computation must synchronize for each diagonal to be completed.

4. Evaluating Tradeoffs: An Example

The following problem, using the most compute-intensive aspects of Method 2 and Method 3, was used as a benchmark in evaluating topology tradeoffs. S follows an Ito process with constant μ and σ, and f is an American-style derivative. We use a recombining lattice to find the expected value of f. Next, we vary the average instantaneous interest rate r in by taking 1000 Monte Carlo samples. Thus the value of f is the sample average of 1000 lattice evaluations. The algorithm was implemented to support the weakly-coupled parallelism of Method 3.

We compare the impact of several implementations in Figure 4.

	Time for 1 Sample (sec)	Time for 1000 Samples (sec)	No. of Processors
Workstation	5	5000	1
Faster Workstation	1	1000	1
Cluster (PVM)	3	crashed	4
Cluster (PVM/custom)	3	4-103	1000
Cluster-(PVM/SMP)	0.225	225	4
Supercomputer	0.03	30	40

Figure 4. Benchmark Performance of 6 Computing Topologies

The clusters were implemented under Parallel Virtual Machine, a package that permits the utilization of a heterogeneous network of parallel and serial computers as a single computational resource [7].

The three cluster implementations of the benchmark problem. In the first cluster inplementation, the benchmark problem crashed the system. There were too many Monte Carlo requests for the network task scheduler to handle the barrier operations. In the second cluster inplementation, the problem was reconfigured to allocate one Monte Carlo sample to each processor. The time rquired to perform 1000 samples then depended on the latency of the network: it is variable because the network is a shared resource. In the third cluster inplementation, the network was a dedicated high-speed backplane (see Figure 2). In this "Symmetric Multi-Processing" implementation, only four processors were allowed to be active at one time.

These results show that the underlying network topology, is the crucial factor in designing cluster computing solutions. Similar results on cluster computing performance are discussed in [8].

5. An Expert System for Assessing Computing Alternatives

We have collected several cases that can be used to assess the transition between alternative computing technologies for the optimal computation of Methods 1-3. There are 16 basic cases, corresponding to the pairwise transitions between each of the 4 topologies, and the null transition — the alternative of keeping the computing topology the same. Our cases were derived by examining similar transition problems for other compute-intensive applications. Our case profiles include the attributes discussed in Section 2, concerned with opportunity cost, infrastructure cost, and algorithmic cost. As in other case-based reasoning systems, our cases contain typical examples and counter-examples (exceptions). We summarize the conclusions of the typical cases:

Case 1. Workstation to Workstation.
Alternatives provide marginal gain in performance. Alternatives are too expensive. No skills to perform algorithm modification. Algorithm is difficult to parallelize.

Case 2. Workstation to Faster Workstation.
No algorithm modification required. Limited Budget.

Case 3. Workstation to Supercomputer .
Algorithm exploits utilization of vectors and vector operations. Budget for the supercomputer is available. Workstations all busy. Bad network infrastructure. Require consistent performance. Skills available to modify algorithm and optimize in FORTRAN. Low modification costs.

Case 4. Workstation to Cluster .
Have many workstations and budget is available to buy more workstations. Other departments will allow limited use of their workstations. Problem cannot be solved with supercomputers.

Case 5. Faster Workstation to Workstation.
Lose of Budget.

Case 6. Faster Workstation to Faster Workstation. Alternatives provide marginal gain in performance. Alternatives too expensive. No skills to do the rehosting. Algorithm is difficult to parallelize.

Case 7. Faster Workstation to Super- computer.
Generally same as Case 3.

Case 8. Faster Workstation to Cluster.
Generally same as Case 4.

Case 9. Supercomputer to Workstation.
Lose of Budget. Performance not good enough to continue justification of Supercomputer. Algorithm is too memory-intensive and too large for the Supercomputer. Staff unable to program in FORTRAN to get maximum Supercomputer performance.

Case 10. Super Computer to Faster Workstation. Fast Workstations can provide 50% of Supercomputer performance at 10% of the price.

Case 11. Super Computer to Cluster.
Lose of Budget. Performance not good enough to continue justification of Supercomputer. There are many workstations available. Algorithm is too memory-intensive and too large for Supercomputer. Staff unable to program in FORTRAN to get maximum Supercomputer performance.

Case 12. Super Computer to Super Computer.
Algorithm performance is satisfactory. New algorithm developed for supercomputer will not work on anything else: cost to reimplement is high. New model upgrade costs are low.

Case 13. Cluster to Workstation.
Solution is having a negative impact on business, primarily due to the saturation of the network. Performance at desktop is being hurt. Everyone is getting a workstation to exploit the computing capability.

Case 14. Cluster to Faster Workstation.
Same as Case 13. Can afford more power per desktop.

Case 15. Cluster to Cluster.
Future model of computing topology. Algorithm performance is satisfactory. New model upgrade costs are low. New faster network topologies becoming available.

Case 16. Cluster to Supercomputer.
Solution is having a negative impact on business, primarily due to the saturation of the network. Performance at desktop is being hurt. Everyone is getting a workstation to exploit the computing capability. Algorithm exploits utilization of vectors and vector operations. Budget available. Workstations all busy. Bad network infrastructure. Require consistent performance. Skills available to modify algorithm and optimize in FORTRAN. Low modification costs.

In operation, a problem profile representing attributes relating to the opportunity costs, infrastructure costs, and algorithmic costs are entered in case fields. The expert system then compares each case to the problem profile, and then ranks all cases by similarity.

It seems that as workstation costs decline, the cluster topology becomes more cost effective. However, as seen in the above cases, this alternative is not without problems. A better statement is that as workstation costs and networks improve, the cluster topology will become more cost effective. An important trend that can further improve cost-effective computation is the development of intelligent resource (process and processor) allocation and network load schedulers built into all operating systems ([3], [5],[6]).

6. 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] Huang, C., and McKinley, P., "Communication Issues in Parallel Computing Across ATM Networks," **IEEE Parallel & Distributed Technology,** Winter 1994.

[4] Ingersoll, J., **Theory of Financial Decision Making**, Rowman & Littlefield, 1987.

[5] Kaplan, J., and Nelson, M., "A Comparison of Queuing, Cluster, and Distributed Computing Systems," NASA Technical Report TM-109025 (Revision 1), June 1994.

[6] Lirov, Y., et al, "Intelligent Infrastructure for the Distributed Front Office," in **Artificial Intelligence in the Capital Markets**, ed. by R.S. Freedman, R. Klein. & J. Lederman, Probus, 1995.

[7] Beguelin, A., et al, **A Users' Guide to PVM Parallel Virtual Machine**, Oak Ridge National Laboratory, U.S. Department of Energy Contract, DE-AC-05-84OR21400.

[8] Anderson, T., et al, "A Case for NOW (Networks of Workstations)," Report for Advanced Research Projects Agency, Contract N00600-93C-2481.