# EFFICIENCY IMPROVEMENTS FOR PRICING AMERICAN OPTIONS WITH A STOCHASTIC MESH: PARALLEL IMPLEMENTATION[1]

Thanos Avramidis[2], Yuriy Zinchenko [3], Thomas F. Coleman[4], Arun Verma[5]

**Abstract**

We discuss a parallel implementation of Monte Carlo simulation algorithms for estimating the price of American-style options. We focus on the stochastic mesh method originally proposed in [3]. The method's statistical efficiency was improved by a bias-reduction technique developed in [1] and [2]. We report results on the efficiency of the parallel implementation of these two algorithms on an SGI Origin 2000 computer with up to 32 processors. Our conclusion is that the algorithm gains almost linear performance improvement with respect to the number of processors engaged in computations for moderate to large mesh sizes.

## 1. Introduction

In the financial markets, sophisticated, complex products are continuously offered and traded. With the increasing complexity of these products, Monte Carlo simulation is steadily becoming an important tool used in valuing and hedging complex products. In this paper, we focus on American options, where we assume that the option can be exercised discretely, as opposed to continuously--that is, the option holder can exercise the option at a fixed set of time points (also called exercise opportunities, or stages) up to expiration.

Until recently, the prevailing opinion was that American options could not be handled by Monte Carlo simulation; e.g., Hull (1997, p. 364). Recent developments, however, have started to pave the way for estimating American option prices via simulation [1,2,3,4]. An important method developed recently for pricing American options via simulation is the stochastic mesh method proposed in Broadie and Glasserman (1997a), henceforth BG1997a. For a general-purpose implementation of the method, Avramidis and Hyden (1999) observed severe bias in the mesh estimators and developed a bias-reduced mesh estimator that drastically improves the accuracy, measured as the inverse of the mean square error. For a more complete treatment of the bias-reduced estimator, see Avramidis (2000).

---

[1] This work was funded by the Financial Industry Solutions Center, a joint venture of Silicon Graphics International and Cornell University based in New York City.

[2] Assistant Professor, School of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY 14853.

[3] School of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY 14853

[4] Director, Cornell Theory Center, Ithaca, NY 14853 and Director, Financial Industry Solutions Center, 55 Broad St, New York, NY.

[5] Cornell Theory Center and Financial Industry Solutions Center

In this paper, we report on a parallel implementation of mesh-type estimators. The paper is organized as follows. For completeness, we describe the estimators in BG97 and A00 in Section 2. Section 3 contains the algorithm structure and implementation details. In Section 4 we report computational results on timings and the efficiency of the parallel implementation.

## 2. Background: American Option Pricing and Mesh Estimation

Let $S_t = (S_t^1,...,S_t^n)$ denote the vector of securities underlying the option, modeled as a Markov process on $R^n$ with discrete time-parameter $t = 0, 1,\ldots, T$. The argument $t = 0, 1, \ldots, T$ indexes the set of time points (in increasing order) when the option is exercisable, also called *exercise opportunities* or simply *stages*. Let $h(t, s)$ be the payoff from exercise at time $t$ in state $s$, discounted to time 0 with the (possibly stochastic) discount factor recorded in $S_t$. Since $S_t$ is Markovian, the option value at (time, state) pair $(t,s)$ is obtained by dynamic programming:

$$q(t,s) = \begin{cases} h(T,s) & \text{for } t = T \text{ and all } s \\ \max\{h(t,s), c(t,s)\} & \text{for } t < T \text{ and all } s \end{cases}$$

where $c(t, s)$ is the discounted value of the option associated with the decision to "continue", i.e., not exercise the option at $(t, s)$, thereby holding it until at least stage $t+1$:

$$c(t,s) = E[q(t+1, S_{t+1}) \mid S_t = s] \qquad (1)$$

The quantity $c(t, s)$ is called the *continuation value* at $(t, s)$. Arbitrage-pricing theory suggests that the arbitrage-free price of the option is obtained when the conditional expectation in (1) is with respect to the Equivalent Martingale Measure (EMM). Under the EMM, the value of any tradeable security, discounted to time 0, is a martingale.
The problem is to compute the option value at time 0, $q(0, s_0)$, where $s_0$ is the known state of underlying assets at time 0.

*Example*. In a simple application, $S_t$ is a vector of $d$ stock prices. A max call option has payoff $h(t, S_t) = (\max(S_t^1,..,S_t^d) - K)^+$; a geometric average call option has payoff $h(t, S_t) = \left((\prod_{k=1}^{d} S_t^k)^{1/d} - K\right)^+$, where $K$ is the *strike price* and $x^+$ stands for $\max(x, 0)$.

In reviewing the mesh method, we follow BG1997a. The method generates a *mesh* of randomly sampled states (also called *points*) $S_{t,i}$ , $i=1,\ldots,b$ for each $t=1,\ldots,T$. For convenience, we define nonrandom mesh points at stage 0 equal to the state of underlying securities at time 0, $S_{0,i} = s_0$, $i=1,\ldots,b$. For $t=1,\ldots,T$, let $g_t(.)$ denote the probability density from which the points $\{S_{t,i} : i=1,\ldots,b\}$ are sampled (to be specified later), and let $f_t(x,\cdot)$ denote the conditional EMM density of $S_{t+1}$ given $S_t = x$. It is assumed that $f_t(x,\cdot)$ exists for all $x$ and is known in closed form or can be evaluated numerically at negligible cost. The *high mesh* estimator of the option value is defined recursively:

$$\hat{q}_H(T, S_{T,i}) = h(T, S_{T,i}), i = 1,...,b\,; \tag{2a}$$

for t=T-1, T-2,…,0, the high mesh estimator is

$$\hat{q}_H(t, S_{t,i}) = \max(h(t, S_{t,i}), \hat{c}(t, S_{t,i})), i = 1,...,b\,, \tag{2b}$$

where the estimated continuation value of each point sampled at stage t depends on the previously estimated continuation values of all points sampled at stage t+1:

$$\hat{c}(t, S_{t,i}) = \frac{1}{b}\sum_{j=1}^{b} \hat{q}_H(t+1, S_{t+1,j}) w(t+1, S_{t,i}, S_{t+1,j}) \tag{3}$$

where the *weights* $w(\cdot,\cdot)$ are

$$w(t+1, S_{t,i}, S_{t+1,j}) = \frac{f_t(S_{t,i}, S_{t+1,j})}{g_{t+1}(S_{t+1,j})}. \tag{4}$$

The weighing of the combination of points ($S_{t,i}$ , $S_{t,i+1}$) above is necessary in light of the fact that the points at stage $t + 1$ were sampled from the density $g_{t+1}$ $(\cdot)$ instead of the density $f_t$ $(S_{t,i}$ , $\cdot)$ appropriate for sampling a path to estimate the continuation value of point $S_{t,i}$ .

The choice of densities $g_t$ $(\cdot)$ is crucial. For the rest of the paper, we assume the mesh is generated by sampling $b$ independent and identically distributed paths of $S_t$:

$$\{S_{t,i}\,,\ t = 0,\ldots,T\},\ i=1,\ldots,\ b \text{ are i.i.d. paths of } S_t\,. \tag{5}$$

We call the pair $(S_{t,i}, S_{t+1,i})$ a *parent* and *child*, respectively, to indicate the stochastic dependence. BG1997a provide evidence that a good choice is to sample $b$ paths as in (5) and view the points at stage $t$+1 as a sample of identically distributed points from the average conditional EMM density associated with their parents:

$$g_{t+1}(\mathbf{S}_t, u) = \frac{1}{b}\sum_{i=1}^{b} f_t(S_{t,i}, u)\,, \tag{6}$$

where $\mathbf{S}_t = (S_{t,i}, i = 1,...,b)$. Note that $g_{t+1}(\mathbf{S}_t, \cdot)$ depends on all parents of points sampled at stage $t$, and corresponds to "forgetting" the parent-child relationship.

We continue by defining the mesh estimators in Avramidis (2000). A first step is to construct the mesh low estimator $\hat{q}_L(t, S_{t,i})$, which is a biased-low estimator for the option value at ($t$, $S_{t,i}$) obtained from within the mesh. The idea behind the construction is to use disjoint sets of points for estimation of the optimal exercise policy and the estimation of continuation values (in case the estimated optimal policy is to continue). Unlike the BG1997a mesh high estimator, we "remember" the parent of each point. For simplicity, assume $b$ is even, and define

$$B_1 = \{1,2,...,\frac{b}{2}\},\ \ \mathrm{B}_2 = \{\frac{b}{2}+1,\frac{b}{2}+2,...b\}\text{, and } \tau(j) = \begin{cases} 2, & j \in B_1 \\ 1, & j \in B_2 \end{cases}. \quad (7)$$

To calculate the low mesh estimator at ($t$, $S_{t,i}$), assume the low mesh estimator of the option value at each of the sampled points at stage $t$+1 has been calculated. Define the $j$-th estimate of the optimal-exercise action using only the points in $B_j$ from stage $t$+1:

$$\hat{e}_j(t,S_{t,i}) = 1\{h(t,S_{t,i}) > \hat{c}_L(t,S_{t,i},B_j)\},\ \ j=1,2, \quad (8)$$

where $1\{\cdot\}$ is the indicator function of the corresponding event, and

$$\hat{c}_L(t,S_{t,i},B_j) = \frac{1}{b/2}\sum_{k\in B_j}\hat{q}_L(t+1,S_{t+1,k})w_L(t,S_{t,i},S_{t+1,k}), \quad (9)$$

where the weights above are

$$w_L(t,S_{t,i},S_{t+1,k}) = \frac{f_t(S_{t,i},S_{t+1,k})}{g_{t+1,B_j}(\mathbf{S}_t,S_{t+1,k})}. \quad (10)$$

The weights in (10) correspond to viewing the points $\{S_{t+1,k}\}_{k\in B_j}$ as being identically distributed from a density equal to the average conditional EMM density associated with their parents:

$$g_{t+1,B_j}(\mathbf{S}_t,u) = \frac{1}{b/2}\sum_{l\in B_j} f_t(S_{t,l},u). \quad (11)$$

The *low mesh* estimator of the option value is defined recursively:

$$\hat{q}_L(T,S_{T,i}) = h(T,S_{T,i}), i=1,...,b; \quad (12a)$$

for $t$=$T$-1, $T$-2,…,0, the low mesh estimator is

$$\hat{q}_L(t,S_{t,i}) = \sum_{j=1}^{b}\left\{\hat{e}_{\tau(j)}(t,S_{t,i})h(t,S_{t,i}) + [1-\hat{e}_{\tau(j)}(t,S_{t,i})]\hat{q}_L(t+1,S_{t+1,j})\frac{f_t(t,S_{t,i},S_{t+1,j})}{g_{t+1}(\mathbf{S}_t,S_{t+1,j})}\right\}, i=1,...,b \quad (12b)$$

By construction, each forward point $S_{t+1,j}$ is used in conjunction with an estimate of the optimal exercise action $\hat{e}_{\tau(j)}(\cdot,\cdot)$ based on points *other than* $S_{t+1,j}$ (recall (7)).

We are ready to define the *recursively averaged* estimator of Avramidis (2000):

$$\hat{q}_A(T,S_{T,i}) = h(T,S_{T,i}), i=1,...,b; \quad (13a)$$

for $t$=$T$-1, $T$-2,…,0, the recursively averaged estimator is

$$\hat{q}_A(t,S_{t,i}) = \frac{1}{2}\left(\hat{q}_{H,A}(t,S_{t,i}) + \hat{q}_{L,A}(t,S_{t,i})\right), i = 1,...,b, \qquad (13b)$$

where $\hat{q}_{H,A}(t,S_{t,i})$ and $\hat{q}_{L,A}(t,S_{t,i})$ differ from $\hat{q}_H(t,S_{t,i})$ and $\hat{q}_L(t,S_{t,i})$, respectively, in that in the former estimators, we use the values of the recursively averaged estimator at stage $t$+1, $\hat{q}_A(t+1,\cdot)$, instead of the values $\hat{q}_H(t+1,\cdot)$ and $\hat{q}_L(t+1,\cdot)$ for the calculation at stage $t$, respectively. For motivation and results on the statistical efficiency of these estimators, see Avramidis (2000).

## 3. Algorithm structure and implementation

The following three estimators were implemented:

- The high mesh estimator $\hat{q}_H$ of BG1997a
- The low mesh estimator $\hat{q}_L$ of Avramidis (2000)
- The recursively-averaged estimator $\hat{q}_A$ of Avramidis (2000)

An algorithm for the computation (including parallel processing) of $\hat{q}_L$ follows (the code for the other two estimators is similar and is thus omitted).

Step 1. Generate random mesh points as in (5)
Step 2 (Backwards recursion):
$t = T$; Compute option values as in (12a).
For $t = T$-1, $T$-2, …, 0:
    For $j$ = 1, 2, …, $b$  % IN PARALLEL
        Compute the density function as in (6) with $u = S_{t+1,j}$  (O($b$) work)
    End For
    For $j$=1,2
        For $k \in B_j$  % IN PARALLEL
            Compute the density function as in (11) with $u = S_{t+1,k}$  (O($b$) work)
        End For
    End For
    For $i$ = 1, 2, …, $b$  % IN PARALLEL
        For $j$=1,2
            Compute the weight as in (10)
            Compute the continuation value as in (9)  (O($b$) work)
            Compute the estimate of the optimal exercise action as in (8)
        End For
        Compute low mesh estimator as in (12b)  (O($b$) work)
    End For
End For

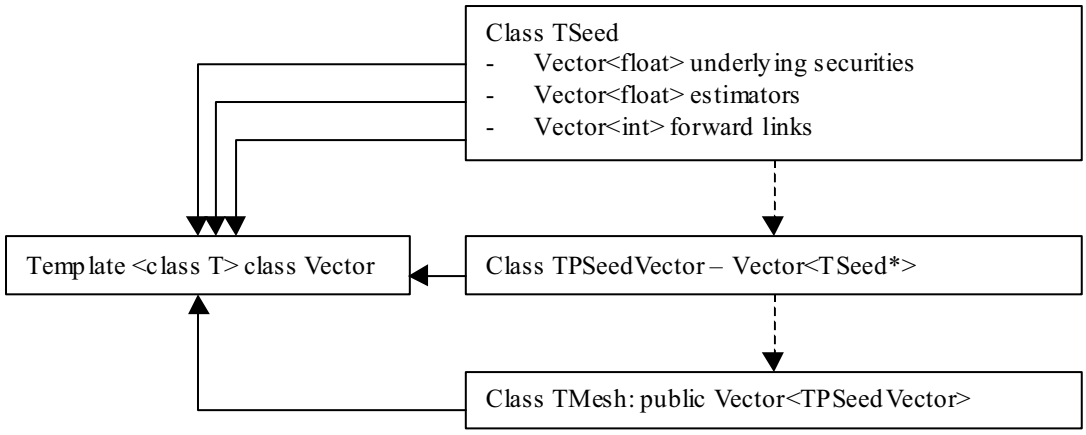
For statements that involve work that grows with $b$, we indicated the work requirement in parentheses in O(·) notation. The algorithm complexity as a function of $T$ and $b$ is as

follows. The generation of mesh points in Step 1 requires O($Tb$) work. The backward computation of option values for all mesh points in Step 2 has total work requirement of order O($Tb^2$). Since the work in Step 1 is an asymptotically negligible fraction of total work as $b$ increases, we implemented parallel processing only for Step 2 (clearly, the generation of paths in Step 1 could also be parallelized). The For loops labeled "% IN PARALLEL" are executed in parallel, as the computations for each loop iteration are independent of each other. This analysis shows that the O($Tb^2$) work of mesh-type algorithms can be parallelized up to $b$ processors with almost perfect efficiency, in which case the execution time will be approximately O($Tb$), an impressive speedup.

We implemented all three estimators in the C++ programming language, which is convenient for dealing with dynamical data structures. The implementation was done in the object-oriented framework and the following classes/templates were created:

- template class Vector – used to store the information on the underlying securities and estimators for each node in the mesh, used to create a class for mesh points at a particular stage and also used as a base class for a whole mesh
- class TSeed – represents a single point of a mesh
- class TPSeedVector – container class for all the mesh points at stage t
- class TMesh – contains a mesh data structure itself and is able to resample mesh, compute the estimators etc

Here is a diagram showing how these objects are interrelated:




*3.1 Parallel Implementation*

For parallel implementation, we used the OpenMP libraries on the SGI's Origin 2000 machine. The pragma directives were used for the automatic parallelization of the C++ code. No changes to the original code were necessary. We simply identified the parallel portions and added the pragma calls before each parallel-execution loop as shown below.

```
#define _MAKE_PARALLEL_  % initialize
```

…
…

#ifdef _MAKE_PARALLEL_ % typical parallel For loop
#pragma omp parallel for
#endif
For $j$ = 1, 2, ..., $b$ % IN PARALLEL
…
End For

## 4. Computational Results

Experiments were run for two option types (max and geometric average payoff function) for different mesh sizes and varying dimensionality of the option (for the case of the geometric average option) and number of exercise opportunities (for the case of the max option). The machine used was an SGI Origin 2000 cc-NUMA multiprocessor server. Timing results were obtained with up to 32 processors. Since all three estimators have similar computational complexity, we report execution times only for $\hat{q}_A$ in Table 1. As expected, given a fixed number of processors engaged, the execution time grows approximately linearly with the number of exercise opportunities $T$ and the number of underlying securities $n$, and grows quadratically with the number of mesh points per stage $b$. For large mesh sizes ($b$ =2048) we achieved almost perfect parallel efficiency, with an approximate speedup by a factor of 28 on 32 processors. The speedup improves with the mesh size.

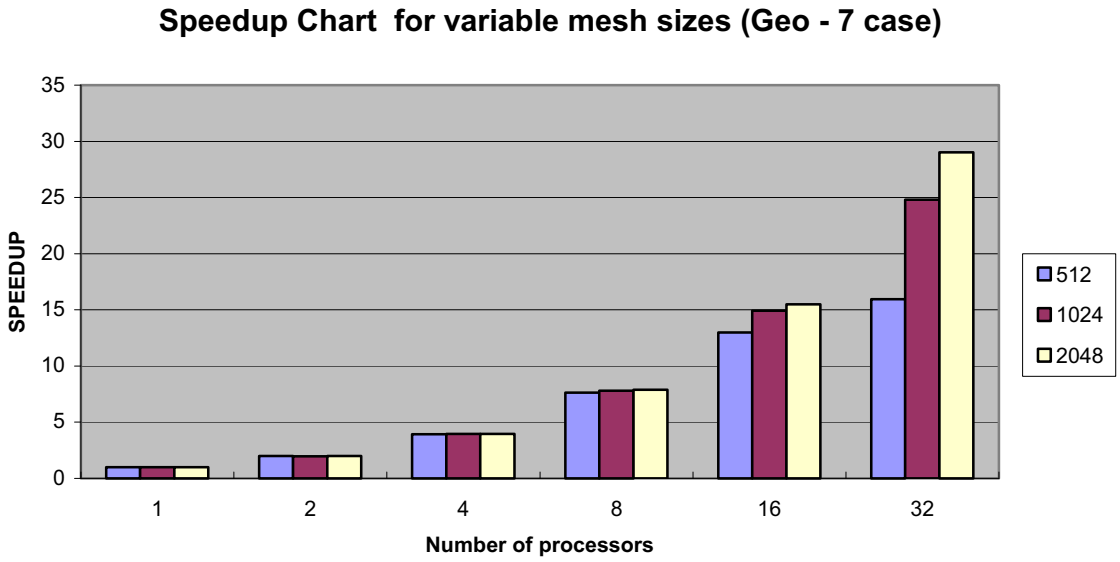
Table 1. Execution time (seconds) on SGI Origin 2000.

| Option Type | | # processors -> | 1 | 2 | 4 | 8 | 16 | 32 |
|---|---|---|---|---|---|---|---|---|
| Geometric Average | # assets $d$ | Mesh size b | | | | | | |
| | 5 | b = 512 | 93.67 | 47.26 | 23.94 | 12.47 | 7.57 | 6.37 |
| | | b = 1024 | 380.71 | 191.53 | 96.72 | 48.75 | 25.37 | 16.15 |
| | | b = 2048 | 1547.7 | 772.39 | 388.8 | 195.6 | 100.1 | 54.15 |
| | 7 | b = 512 | 128.1 | 64.33 | 32.58 | 16.77 | 9.87 | 8.03 |
| | | b = 1024 | 518.02 | 261.81 | 130.8 | 66.36 | 34.72 | 20.89 |
| | | b = 2048 | 2090.8 | 1046.6 | 527.2 | 264.8 | 134.9 | 72.07 |
| Max | # stages $T$ | | | | | | | |
| | 3 | b = 512 | 20.63 | 10.43 | 5.39 | 2.90 | 1.87 | 1.78 |
| | | b = 1024 | 82.18 | 41.20 | 20.77 | 10.64 | 5.80 | 3.77 |
| | | b = 2048 | 328.88 | 164.52 | 88.96 | 42.53 | 21.77 | 12.05 |
| | 6 | b = 512 | 52.14 | 26.01 | 13.26 | 7.01 | 4.24 | 3.79 |
| | | b = 1024 | 209.4 | 104.9 | 58.82 | 26.88 | 14.19 | 9.08 |
| | | b = 2048 | 847.7 | 425.0 | 216.1 | 108.3 | 55.16 | 30.28 |
| | 9 | b = 512 | 82.82 | 41.90 | 21.11 | 11.45 | 6.73 | 5.74 |
| | | b = 1024 | 333.2 | 167.6 | 84.15 | 43.26 | 22.19 | 13.98 |
| | | b = 2048 | 1338.1 | 671.7 | 336.7 | 171.6 | 86.59 | 46.69 |

*4.1 Parallel Efficiency Results*

We define:

- Speedup = serial time/parallel time.
- Parallel efficiency = Speedup/P, where P is the number of processors.




*4.2 Empirical study of serial and parallel components via regression*

To separate the serial and parallel components of the code execution time, we used the following formulas to represent parallel and serial execution times:

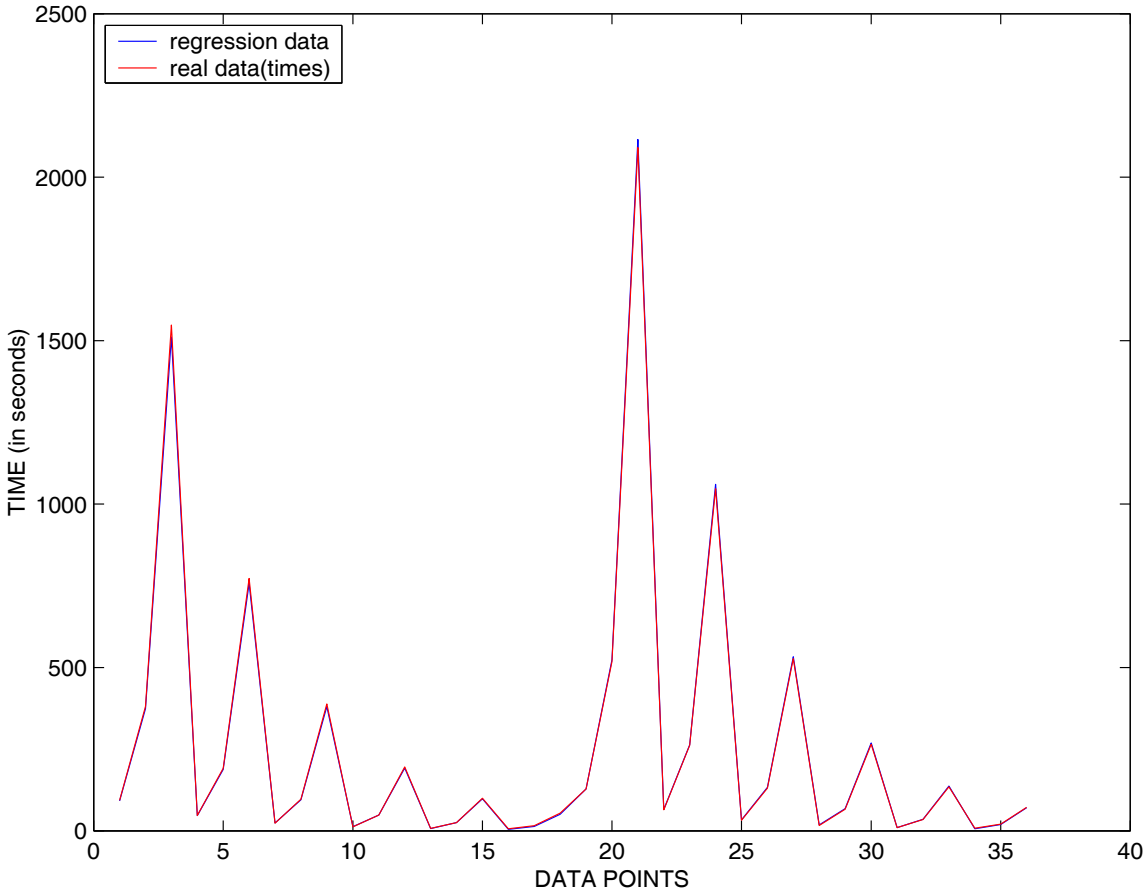
serial = c*(s1+s2*N+s3*N^2)
parallel = c*(p1+p2*N+p3*N^2).

Here c is a constant, N = b/512 measures the normalized mesh size, and s1, s2, s3 and p1, p2, p3 are the coefficients to be determined. The regression model we estimated is

Totaltime = serial + parallel/P + error,

where P is the number of processors.

The estimated coefficients were: s1= 0.2101, s2 = 0.1150, s3 = 0.0044, p1 = 0.0439, p2 = 0.1240, p3 = 19.1197. This confirms our expectation that the quadratic component of execution time is dominant. The chart below compares the actual and regression-fitted execution times; note that regression results very accurately match the data.




## 5. Conclusion

In this paper, we describe a parallel implementation of Monte Carlo simulation algorithms for the estimation of American-style option prices. We focus on the stochastic mesh method developed in [3] and [1] as a general-purpose Monte Carlo algorithm for addressing the estimation problem. Three important mesh-type estimators were implemented using C++ on SGI Origin 2000 cc-NUMA multiprocessor server.

A direct algorithm analysis and an empirical study demonstrate that almost perfect parallel efficiency can be achieved. For moderately large mesh sizes we achieved almost perfect parallel efficiency, i.e., execution time declines almost linearly with the number of processors used, and almost perfect processor utilization is achieved. The resulting benefits are: much faster estimation is feasible for computationally-intensive and/or critical applications; as American option pricing in dimension greater than 3-4 is still a hard problem computationally, fast estimation by parallel processing may facilitate further advances in research.

**References**


1. Avramidis, A. N. 2000. Efficiency improvements for pricing American Options with a stochastic mesh. Working paper, School of ORIE, Cornell University, Ithaca, NY 14853.
2. Avramidis, A. N., and P. Hyden. 1999. Efficiency improvements for pricing American options with a stochastic mesh. *Proceedings of the 1999 Winter Simulation Conference*, 344-350.
3. Broadie, M., and P. Glasserman. 1997a. A stochastic mesh method for pricing high-dimensional American options. Unpublished manuscript.
4. Broadie, M., and P. Glasserman. 1997b. Pricing American-style securities using simulation. *Journal of Economic Dynamics and Control*, 21, 1323-1352.
5. Hull, J. 1997. *Options, Futures, and other derivatives,* 3rd Edition, Prentice-Hall.