Optimizations in Financial Engineering: The Least-Squares Monte Carlo method of Longstaff and Schwartz

Anamitra R. Choudhury* Alan King † Sunil Kumar ‡
Yogish Sabharwal*

Submitted to IEEE International Parallel and Distributed Processing Symposium - IPDPS 2008. Paper in process. Please do not distribute.
8 October 2007

Abstract

In this paper we identify important opportunities for parallelization in the Least-Squares Monte Carlo (LSM) algorithm, due to Longstaff and Schwartz [17], for the pricing of American options. The LSM method can be divided into three phases: *Path-simulation*, *Calibration* and *Valuation*. We describe how each of these phases can be parallelized, with more focus on the *Calibration* phase, which is inherently more difficult to parallelize.

We implemented these parallelization techniques on Blue Gene using the Quantlib open source financial engineering package. We achieved up to factor of 9 speed-up for the Calibration phase and 18 for the complete LSM method on a 32 processor BG/P system using monomial basis functions.

1 Introduction

Options are financial derivatives that give the holder the right, but not the obligation, to buy or sell an underlying asset at a contractually-specified *strike price* on a range of future dates. Call Options give the right to buy the underlying asset, whereas Put Options give the right to sell. There are many varieties of options. European options are exercisable only on maturity and Bermudan options can be exercised at specific dates on or before maturity. American options can be exercised on any trading date before maturity.

Valuation and optimal exercise of derivatives with American-style exercise features is one of the most important practical problems in option pricing. These types of derivatives are found in all major financial markets including the equity, commodity, foreign exchange, insurance, energy, sovereign, agency, municipal, mortgage, credit, real estate, convertible, swap, and emerging markets.

*IBM India Research Lab, Plot 4, Block-C, Institutional Area, Vasant Kunj, New Delhi, India
†IBM T. J. Watson Research Center, 10598 Yorktown Heights, NY, USA
‡Department of Electrical Engineering, University of Southern California, Los Angeles, USA

This paper investigates parallelization opportunities for an important new algorithm called Least Squares Monte Carlo (LSM) for American options pricing, due to Longstaff and Schwartz [17], that is gaining widespread adoption in the financial industry. The LSM method belongs to the class of basis function methods for American options pricing; here is a brief overview of the recent literature on the LSM method. Broadie and Detemple [9] provide a survey of recent trends in valuation methods for options pricing for basis function methods. Lai and Wong [15] review these approaches with a view to understanding guidelines for the choices of basis functions. Liberati and Platen [16] discuss the efficiency of this class of methods as a simplification of Taylor schemes for solving stochastic differential equations. Chan et al [10] discuss opportunities to reduce memory consumption in the implementation of pricing American-style options. Bolia and Junjeja [8] and Moreni [18] discuss variance reduction techniques for the basis function methods. Finally, Glasserman and Bin [13] and Stentoft [20] provide convergence analyses for the basis function methods, including LSM, under various assumptions on the stochastic processes.

Our paper presents an innovative approach to scaling the computations required to implement the LSM method on a parallel architecture, with particular focus on the *calibration* step, and discuss the computational results obtained by adapting the QuantLib implementation of LSM on IBM's Blue Gene computer. Section 2 gives a brief overview of the valuation of American options and the challenge of implementing Monte Carlo based algorithms for American option pricing, and Section 3 reviews the LSM method. Our contributions to the parallelization and optimization of the LSM method are presented in Section 4 and our computational results are presented in Section 5.

1.1 Acknowledgements

The authors gratefully acknowledge valuable conversations with Paul Glasserman on Monte Carlo methods for American options pricing in the financial industry.

2 Valuation of American Options

According to the general theory of derivatives pricing, one may price any derivative security by integrating its payouts with respect to a *risk-neutral measure* on the underlying asset price process. The existence of the risk-neutral measure is equivalent to the availability of arbitrage-free trading strategies that replicate option payouts. For background on financial options pricing and algorithms, see the standard references Hull [14] and Duffie [11].

An American option can be exercised at any time during its life up to the expiration of the option, so its payouts depend greatly on the *exercise policy* of the holder. On the expiration date, the optimal exercise strategy is simply to exercise the option if it is in the money, and let it expire otherwise. Prior to expiry, the optimal strategy is to examine the underlying asset price, compare the option's immediate exercise value with the risk-neutral expected value of holding the option, and determine if immediate exercise is more valuable. The key to optimally exercising an American option is identifying the conditional expected value of continuation under the risk neutral measure.

Algorithms for American option pricing typically fall into two classes. Lattice, or finite difference, algorithms generate a discrete lattice in time and space and iterate backwards in time from expiration, calculating the exercise value and continuation value at each point. The set of lattice points where exercise is more valuable than continuation is called the *exercise frontier*. Lattice methods are well-suited for valuation of American options when the number of space dimensions is low, typically three or four dimensions. Increasing the number of space dimensions comes at the cost of exponential growth in the size of the lattice.

Monte Carlo algorithms for options pricing are based on the well-known technique of Monte Carlo integration, which is easily implemented and trivial to parallelize. Monte Carlo techniques can be effective for high-dimensional pricing problems, since their convergence rate does not depend on the number of dimensions. These advantages are so great that Monte Carlo algorithms have become universally adopted in the finance industry as the valuation method of choice. For a comprehensive treatment of Monte Carlo methods in finance, see Glasserman's monograph [12].

There are two basic challenges to the application of Monte Carlo methods to American options pricing. The first is to construct the algorithm so that it can effectively estimate the exercise frontier. The second challenge is to construct algorithms whose performance can be improved by scaling over multiple processors. The basic issue is that simulations typically generate trajectories of state variables forward in time, while the determination of optimal exercise policies require backward-style dynamic programming techniques.

3 The LSM method of Longstaff and Schwartz

Longstaff and Schwartz [17] recently published an algorithm for American options pricing called Least Squares Monte Carlo (LSM). LSM proceeds by simulating forward paths using Monte Carlo, and then performs backwards-style iterations where at each step it performs a least-squares approximation of the continuation function over a collection of basis functions. This algorithm is simple to implement within existing Monte Carlo frameworks, and has the additional advantages that the continuation functions are constructed explicitly and the algorithm is easy to calibrate to existing market prices.

Let ω denote a sample path of underlying asset prices generated by Monte-Carlo simulation over a discrete set of τ exercise times $0 < t_1 \leq t_2 \leq \cdots \leq t_\tau = T$. The continuous exercise property of the American option is approximated by taking sufficiently large τ. Let $C(\omega, s; t, T)$ denote the path of cash flows generated assuming the option is not exercised at or before time t and the option holder follows the optimal exercise policy for all subsequent s, $t < s \leq T$.

At maturity, the investor exercises the option if it is in the money. At time t_k prior to expiration, the option-holder must decide whether to exercise at that point or to continue and revisit the decision at the next time point. Here, although the option-holder knows the immediate exercise payoff, he has no idea of the expected cash flows from continuation. According to the no-arbitrage valuation theory, $F(\omega; t_k)$, the continuation value at time t_k

Algorithm 1: LSM Algorithm for pricing

input : Strike price K, riskless interest rate, number of timesteps τ, number of paths n
output: Price of the American Option

Generate n paths each of τ timesteps, using Monte-Carlo simulations from Q
for $j \leftarrow \tau - 1$ **to** 1 **do**
  for *every path* ω **do**
    Find estimate $\hat{F}(\omega, t_j)$ of the continuation value at t_j for ω
    Exercise value at t_j for path ω is $E(\omega, t_j) = \max(K - S(t_j, \omega), 0)$ for put option, where $S(t_j, \omega)$ is simulated stock price value at time t_j along path ω
    if $\hat{F}(\omega, t_j) \leq E(\omega, t_j)$ **then**
      Optimum strategy at t_j is to exercise then
    else
      Optimum strategy at t_j is that of t_{j+1}
    end
  end
end
Generate new paths, each of τ timesteps, using Monte-Carlo simulations
Find optimum stopping points for these paths using the estimator functions of the continuation value obtained above
Discount the resulting cash flow from the stopping time to time zero.
return average *starting price of option* over all paths generated.

for path ω, is given by

$$F(\omega; t_k) = E_Q \left[\sum_{j=k+1}^{\tau} \exp^{-r(\omega, t_j; t_k)} C(\omega, t_j; t_k, T) \mid \mathcal{F}_{t_k} \right]. \tag{1}$$

where $r(\omega, t_j; t_k)$ is the interest rate in period t_k for riskless securities that mature in period $t_j > t_k$, Q is the risk neutral pricing measure and the expectation of the cash flows is taken conditional on information set $\mathcal{F}_{t_k}$ at t_k.

What we require is a good estimate of the above continuation value. Let $\hat{F}(\omega; t_k)$ denote this estimate – its derivation is discussed in Section 4.1 below. Once the continuation value is estimated, we can decide whether it is optimal to exercise at t_k or continue by comparing the immediate exercise value with $\hat{F}(\omega; t_k)$. The procedure is repeated until exercise decisions have been determined for each exercise point on every path. Thus by estimating the conditional expectation function for each exercise date, a complete specification of the optimal exercise strategy can be obtained along each path.

Once the exercise strategy has been estimated, the valuation of the American option is simple: paths of underlying asset prices are generated by Monte-Carlo simulations from the risk-neutral measure Q. For each path the optimal stopping point is determined using the estimator functions (according to option pricing theory, an optimal exercise policy will generate exactly one cash flow for each path). The American option is then priced by discounting the resulting cash flows back to time zero, and averaging the discounted cash flows over all paths.

The high level description of the pricing algorithm is given in Algorithm 1.

4 Parallelizations and Optimizations

In this section we discuss optimization and parallelization techniques for the LSM method. The main technique used in the path generation and pricing phases of the LSM algorithm is Monte-Carlo simulation, which is known to be parallelizable and scalable. The bigger challenge is in parallelizing the calibration phase of the LSM method.

We used QuantLib, an open-source library for quantitative finance, [4], as a reference implementation for demonstrating our parallelization techniques. QuantLib is written in C++ using an object oriented model. It offers tools that are useful for practical implementation and advanced modeling, with features such as market conventions, yield curve models, solvers, PDEs, Monte Carlo, exotic options, VAR, and so on. For more details about QuantLib, the reader is referred to [7]. Some of our optimizations are specific to QuantLib (c.f. Section 4.3).

4.1 Calibration in LSM

The key to the LSM approach is the use of least squares to estimate the conditional expected payoff from continuation using the cross-sectional information in the simulations. In this *calibration* step, the continuation function $F(\omega, t_k)$ from equation (1) is approximated as a linear combination of a countable set of F_{t_k} -measurable basis functions. In particular, if the first m basis functions are denoted $\{L_k(\cdot), k = 1, \ldots, m\}$, then the approximation is written

$$F(\omega, t_j) = \sum_{k=1}^{m} a_k L_k(S(t_j, \omega)). \tag{2}$$

where $S(t_j, \omega)$ is the underlying asset price at the j^{th} timestep for the path ω, and a_k is the coefficient corresponding to the k^{th} basis function L_k. These approximated continuation functions can be interpreted as the approximated option price at time t_j on path ω. The tuples of option prices and underlying asset prices form the data for the least square regression that determines the coefficients a_k.

Let A_j be the $n \times m$ design matrix for the j^{th} timestep with $A_j(i,k) = L_k(S(t_j, \omega_i))$, where ω_i is the i^{th} path, $i = 1, \ldots, n$. Let $X_j = [a_k]_j$ be the unknown vector of the basis function coefficients for the j^{th} timestep to be determined by the regression. The goal is to find the basis function coefficient vector X_j that minimizes $\| A_j X_j - Y_j \|_2$. The column vector values $[Y_j]_i$ are determined by discounting the option price on path i from time t_{j+1} to time t_j. Once the regression coefficients a_k are determined then the path i option prices are updated as follows: compare the *estimated continuation value* $\hat{F}(\omega_i, t_j)$ with the *exercise value* at $S(t_j, \omega_i)$. If the exercise value is larger then use this as the path i price update, otherwise use the path price at time t_{j+1} discounted by $r(\omega_i, t_j; t_{j+1})$.

Let $U\Sigma V^T$ be the *singular value decomposition* of A_j, where U is an $n \times n$ orthogonal matrix, V is an $m \times m$ orthogonal matrix and Σ is an $n \times m$ diagonal matrix, then

Algorithm 2: Estimation of continuation value at time t_j for each path ω

input : Values of all paths at time t_j, set of m basis functions L_k, $k = 1, 2, \cdots, m$, vector of option prices Y_j

output: $\hat{F}(\omega, t_j)$

Store path values at time t_j for different paths in array S_j, i.e., $S_j(i) = S(t_j, \omega_i)$
Using Least Square Regression, solve the system of equations $Y_j = \sum_{k=1}^{m} a_k L_k(S_j)$ to determine a_k, $k = 1, 2, \cdots, m$ (only non-zero Y_j entries considered here).
- Determine SVD of design matrix A_j $[A_j(i,k) = L_k(S(t_j, \omega_i))]$ s.t. $A_j = U\ \Sigma\ V^T$.
- Determine a_k from U, V and Σ. $k = 1, 2, \cdots, m$. $[a_k]_j = (V\Sigma^{-1}U^T)Y_j$.

return $\sum_{k=1}^{m} a_k L_k(S(t_j, \omega))$

$\| A_j X_j - Y_j \|_2$ is minimized for $X_j = \left(A_j^T A_j\right)^{-1} A_j^T Y_j = \left(V\Sigma^{-1}U^T\right) Y_j$.

The regression provides a direct estimate of the coefficients a_k for the conditional expectation function $\hat{F}(\omega, t_k)$ expressed as a linear combination of basis functions in the form (2). The algorithmic statement of the regression step is displayed in Algorithm 2.

An important optimization applied by Longstaff and Schwartz [17] is to include only those paths in the regression step for which the option price is positive, or "in-the-money". This significantly increases the efficiency of the algorithm and decreases the computational time. Finally, Longstaff and Schwarz show that the estimate $\hat{F}(\omega, t_k)$ converges in mean square and in probability to $F(\omega, t_k)$ as the number of in-the-money paths goes to infinity, and is the best estimator based on mean square metric.

The calibration phase can be divided into two subphases – the SVD of the design matrix and the calculation of option price updates. Computation time analysis of the serial algorithm for monomial basis functions shows that SVD is computationally more intensive, taking a factor of 5 more than the update part. Evaluation of the option price vector at time t_j is done using the coefficients calculated at time t_{j+1}, hence this part is intrinsically serial. The key observation that leads to effective parallelization is that the SVD computations by themselves are independent of each other, due to the fact that the design matrix at each timestep only depends on the asset price values. Therefore, the singular value decompositions of the design matrices (A_j) can be done in parallel followed by the serial update of the option price vectors (Y_j), as illustrated in Figure 1.

Thus, if τ denotes the number of timesteps and p the number of processors, the τ SVD

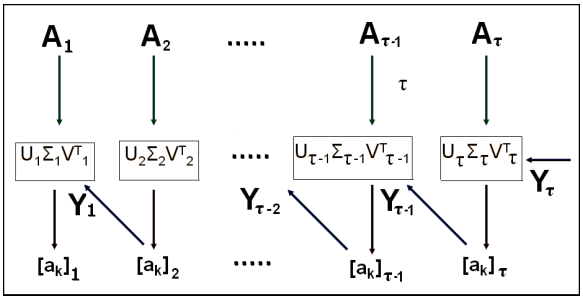


Figure 1: Calibration phase in LSM pricing method

Algorithm 3: Subroutine Par_Cal - Parallel Algorithm for Calibration Phase

input : Design matrices assigned to this processor, Values of the generated paths for corresponding assigned timesteps, riskless interest rate
output: Basis function coefficients for the assigned timesteps

Perform singular value decomposition (SVD) of the assigned design matrices
if *rank = p-1* **then**
  Generate basis functions' coefficients and option price vectors for the assigned timesteps in decreasing order
  Send the option price vector of the smallest assigned timestep to the next lower ranked processor
else
  Receive the option price vector sent from the next higher ranked processor
  Using received data, generate basis functions' coefficient and price vectors for the assigned timesteps in decreasing order
  if *rank* $\neq$ *0* **then**
    Send the option price vector of the smallest assigned timestep to the next lower ranked processor
  end
end
return *Basis function coefficients for the assigned timesteps*

computations are parallelized over the processors. If p does not divide τ, more computations are assigned to the smaller ranked processors. The goal is to overlap some of the SVD computations with option price vector updates. Since the continuation value update proceeds in order from higher to lower ranked processors, and is intrinsically serial, the lower ranked processors can be kept busy with extra SVD computations during this time.

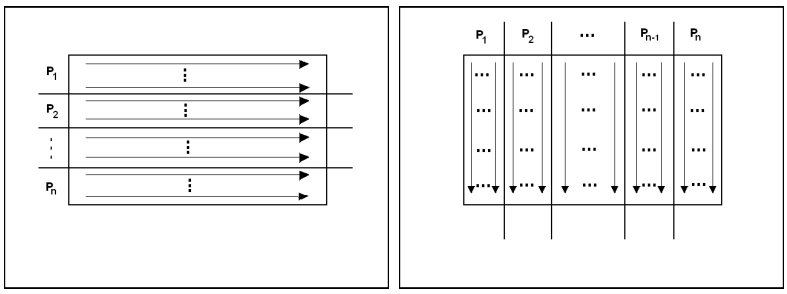
In the calibration phase, after the SVD is done, the highest rank processor calculates the option price updates for timesteps corresponding to which it has the data i.e., timesteps $\tau-2, \tau-3, \cdots, \tau-(\tau/p)$. It sends the option price vector for $\tau-(\tau/p)$ to the next ranked processor. Using this price vector, the next processor computes those of $\tau-(\tau/p)-1, \cdots \tau-2(\tau/p)$. In general, i^{th} processor from end computes the price vector for $\tau-(i-1)(\tau/p)-1$, $\cdots$, $\tau-i(\tau/p)$ $[i=2,3,\cdots,p]$, when τ is a multiple of p. This procedure is repeated till the rank 0 processor calculates the option price for t=0.

The pseudo-code of the algorithm for parallel calibration is given in Algorithm 3.

4.2 Parallelization of other phases

In this section we discuss the parallelization of the path generation and the option pricing phases. Even though these phases are Monte-Carlo simulation based and therefore parallelizable and scalable, we discuss some efficient techniques for inter-processor communication based on MPI (Message Passing Interface) [2] for the LSM method.

Consider the matrix M formed by the paths with each path stored as a row in the matrix, i.e., $M[i,j]$ = path value at time step i for the path j. In the path generation phase if n paths are to be generated in all, each processor generates $\lfloor n/p \rfloor$ or $\lceil n/p \rceil$ paths,



(a) (b)

Figure 2: Path Storing Matrix

(rows) depending on its rank, where each path consists of the values at τ different timesteps, as in Figure 2(a). In the calibration phase, however, each processor operates on $\lfloor \tau/p \rfloor$ or $\lceil \tau/p \rceil$ time values (columns) for all the n paths, as in Figure 2(b). This can be viewed as a global matrix transpose operation which can be accomplished by performing an MPI Alltoallv to exchange data amongst all the processors. Additional pre-processing and post-processing (local transpose operation) is required on the nodes in order to complete the global transpose. Such operations are typical for parallel matrix transpose and are highly optimized by MPI implementations.

Next, in the calibration phase, each processor works on path values at some timesteps and generates coefficients of the basis functions for those timesteps. In the option pricing stage, all processors are required to have the coefficient values of the basis functions for all timesteps. This is enabled by an MPI Allgatherv call at the end of the calibration phase to gather all the coefficients.

The pricing stage is performed in parallel with one master node (node 0) driving this phase. The master node determines the number of paths to be generated and commands the other nodes to generate the paths and collect the statistics. It combines the result, calculates the overall mean and error estimate over all the paths, and checks the error estimate. The procedure is repeated till the error is within required threshold bounds.

The pseudo-code of the algorithm for parallel pricing phase is given in Algorithm 4 and that for the complete parallel LSM method is given in Algorithm 5.

4.3 Other optimizations

More optimizations specific to Quantlib implementations have been done in this work. QuantLib code is highly object oriented in nature; we started by extracting out the relevant data from the object oriented paradigm and pre-computing and pre-processing of some values calculated multiple times in the code. This resulted in significant improvement in timing of the serial program. Also, the basis functions were implemented in highly object oriented manner using function pointers in Quantlib. This was done to make the code more generic that can accommodate many basis functions. These function calls cause significant

Algorithm 4: Subroutine Par_Prc - Parallel Algorithm for Pricing Phase

input : number of timesteps $\tau = t_q$, number of paths n, number of processors p, riskless interest rate, basis functions
output: Price of the American Option

repeat
  if *rank = 0* **then**
    if *error estimate* $\leq$ *threshold* **then**
      number of paths to be generated, $Numpaths = -1$
    end
    Broadcast $Numpaths$, number of paths to be generated by each processor.
  else
    Receive $Numpaths$ sent by processor 0
  end
  if $Numpaths \neq$ *-1* **then**
    Generate $(Numpaths/p)$ paths for each of τ timesteps, using Monte-Carlo simulations
    Find optimum stopping points and initial price for each path using the basis functions
    Send first and second raw moments of the option starting price over the paths to processor 0
    if *rank = 0* **then**
      Collect first and second raw moments of the option starting price from all processors to compute the mean starting price and error estimate
    end
  end
until *Numpaths = -1*
return average *starting price of option* over all the paths generated.

overhead. The code was simplified and optimized by using macros for the monomial basis function.

Algorithm 5: Parallel LSM Algorithm for pricing

input : Strike price, riskless interest rate, number of timesteps $\tau = t_q$, number of paths n, number of processors p
output: Price of the American Option

Generate (n/p) paths each of τ timesteps, using Monte-Carlo simulations
Perform MPI Alltoallv and associated pre/post processing
Determine the design matrices for each of the (τ/p) timesteps, based on the simulated path values and basis functions.
Perform calibration to get basis function coefficients (call Par_Cal [Algorithm 3])
Gather the coefficients for all columns using MPI Allgatherv
Perform the pricing (call Par_Prc [Algorithm 4])
return *value returned by Par_Prc.*

5 Results

We used QuantLib as the reference implementation to optimize the LSM method and analyzed its performance on the Blue Gene/P Supercomputer. This section describes some relevant features of the Blue Gene/P computer, the setup of our test bed, followed by analysis of the performance of the parallelized LSM algorithm. Our focus is on relative performance improvements, therefore we chose a conservative set of compiler flags for all experiments.

5.1 Blue Gene/P Architecture

The Blue Gene/P is the second-generation of the massively parallel Blue Gene supercomputer designed to attain peak performance in excess of 1 Petaflop. The nodes themselves are physically small, allowing for very high packaging density in order to realize optimum cost-performance ratio.

Each node has four embedded 850 MHz `PPC450` processor cores, allowing the system to run in different modes utilizing different number of processors for application processes. In the *virtual node mode* each node is logically separated into four nodes, each of which has a processor and quarter of the physical memory. Each processor is responsible for its own messaging. In this mode, the node runs four application processes, one on each processor.

The Blue Gene/P uses different interconnect networks for I/O, debug, and various types of interprocessor communication. The most significant of these interconnection networks is the three-dimensional torus that has the highest aggregate bandwidth and handles the bulk of all communication. Each node supports six independent 3.5 Gbps bidirectional nearest neighbor links. The torus network uses both dynamic (adaptive) and deterministic routing with virtual buffering and cut-through capability. The messaging is based on variable size packets, each $n \times 32$ bytes, where $n = 1$ to 8 "chunks". The first eight bytes of each packet contain link-level information, routing information and a byte-wide cyclic redundancy check (CRC) that detects header data corruption during transmission. In addition, a 32-bit trailer is appended to each packet that includes a 24-bit CRC. For more details about Blue Gene/P, reader is referred to [1].

5.2 Test bed and setup

The parallelization techniques were run on Blue Gene/P in virtual node mode. Performance measurements were taken with variable numbers of processors, from 4 to 32. Our objective is not to demonstrate scaling to a very large number of nodes, so we restrict ourselves to 32 nodes in our experiments. Since the computation intensity of the calibration phase depends on strike price, volatility and the nature of the basis function, we conducted experiments by varying these parameters. We consider two scenarios: in the first case, we fix the basis function and vary strike price and volatility; while in the other, we fix a particular strike price-volatility pair and vary the basis function nature. Two different strike price values (36 and 40), three different volatility values (0.2, 0.3 and 0.4) and five different basis functions (Monomial, Laguerre, Hermite, Hyperbolic and Chebyshev) are considered.

5.3 Performance

As pointed out in Section 4.3, the QuantLib code is highly object oriented. So we began by optimizing the serial QuantLib implementation of the LSM algorithm, by extracting out relevant data from the object oriented paradigm and pre-computing some of the required values. The values in Table 1 compare the times taken by the original quantlib code and our improved serial code. In the tables, K and V stand for strike price and volatility respectively, while PG, CAL and PRC represent Path generation phase, Calibration phase and pricing phase respectively. As our intention is to study the impact of our parallelization techniques, we perform all our performance comparisons in the remaining section with this improved serial code.

		Original	Improved
K=36	PG	1.71	0.2
	CAL	0.94	0.66
V=0.2	PRC	2.81	0.46
K=36	PG	1.71	0.21
	CAL	1.02	0.71
V=0.3	PRC	4.35	0.91
K=36	PG	1.71	0.2
	CAL	1.08	0.79
V=0.4	PRC	6.97	1.23
K=40	PG	1.71	0.2
	CAL	1.53	1.15
V=0.2	PRC	4.35	0.87
K=40	PG	1.71	0.2
	CAL	1.45	1.09
V=0.3	PRC	7.31	1.41
K=40	PG	1.71	0.2
	CAL	1.4	1.07
V=0.4	PRC	9.34	1.78

(a)

		Original	Improved
Monomial	PG	1.71	0.2
	CAL	0.94	0.66
	PRC	2.81	0.46
Laguerre	PG	1.7	0.2
	CAL	2.14	1.84
	PRC	3.6	1.32
Hermite	PG	1.71	0.2
	CAL	2.14	1.82
	PRC	3.69	1.35
Hyperbolic	PG	1.71	0.2
	CAL	1.87	1.6
	PRC	3.51	1.16
Chebyshev2th	PG	1.71	0.2
	CAL	3.31	2.99
	PRC	4.79	2.33

(b)

Table 1: Timing comparisons of the original and modified QuantLib code (time in secs) : (a) for varying strike price and volatility with fixed Monomial Basis function; (b) for varying basis functions and fixed volatility, V=0.2 and strike price, K=36.

5.3.1 Calibration phase

The performance for the calibration stage parallelized for 32 processors over the serial version is shown in Figures 3 and 4. Singular value decomposition is computationally more expensive than the price vector update. However, SVD computations can be split and thus scale well on larger nodes, whereas price vector updates are intrinsically serial and do not scale up. Therefore, we observe that with increasing number of processors, the calibration time tends

to converge to the time taken in price vector updates. Figure 3 shows the calibration timings for monomial basis function and different strike price/volatility ratios while Figure 4 shows that for fixed volatility and strike price and different basis functions.

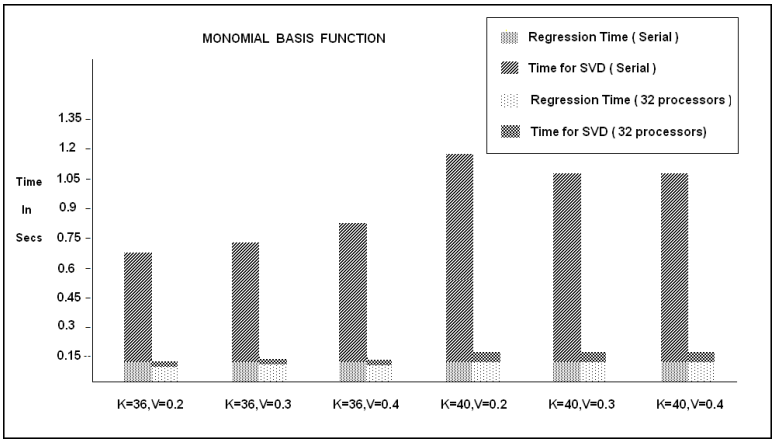


Figure 3: SVD and regression time comparisons for Monomial Basis Function

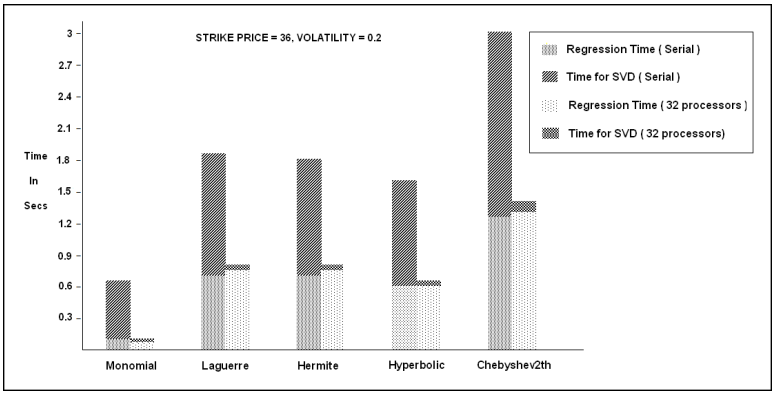


Figure 4: SVD and regression time comparisons for Strike Price =36, Volatility=0.2

As seen from Figure 4, scaling is better for the monomial basis function in comparison to the other functions. This is because the bottleneck in parallelization is the price vector update, which for monomial basis functions is very small compared to other basis functions. In addition, we replaced pointer-based function calls by macros for the monomial case thereby improving the price vector update performance approximately by a factor of 2. Similar modifications can be made for other basis functions as well at the expense of code generality.

5.3.2 Other phases

From Table 1, we observe that the time for path generation (PG) is independent of the strike price, volatility and basis function. This is expected since this phase only generates paths along τ timesteps using Monte Carlo simulations. The path generation time using 32 processors is $0.01sec$ over all combinations of strike price, volatility and basis functions. The

	Monomial	Laguerre	Hermite	Hyperbolic	Chebyshev2th
K=36, V=0.2	0.01	0.05	0.04	0.04	0.08
K=36, V=0.3	0.03	0.1	0.11	0.09	0.19
K=36, V=0.4	0.04	0.15	0.15	0.14	0.27
K=40, V=0.2	0.03	0.1	0.11	0.09	0.17
K=40, V=0.3	0.04	0.15	0.14	0.12	0.24
K=40, V=0.4	0.06	0.19	0.18	0.16	0.31

Table 2: Timings of the pricing phase parallelized for 32 processors (time in secs)

performance for pricing is listed in the Table 2. The individual stages give good speedup, as expected, since Monte Carlo simulations are scalable. However, the overall speedup is limited by the calibration stage timings.

5.3.3 Total time and calibration time scaleup

The calibration timing variations and total timing variations with the number of processors for Monomial basis function is shown in Figures 5 and 6 respectively. We obtain a maximum speedup of 9 for the calibration phase and 18 for the total time for the LSM method using Monomial basis function on 32 processors, when compared to the improved serial code. In comparison with the original QuantLib code, the maximum speedups are 12 and 62 respectively.

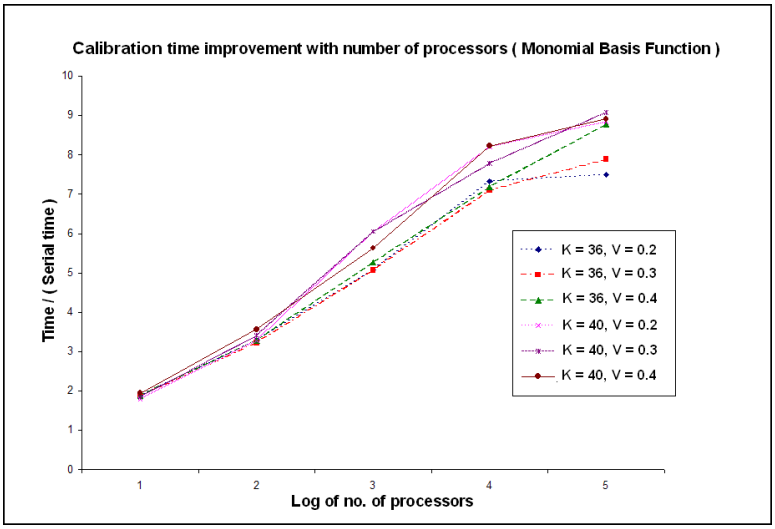


Figure 5: Calibration time variations with number of processors

It is observed that the speedup in calibration phase is limited due to the serial nature of the price vector updates. The total LSM method speedup is comparatively good primarily due to good scaling of Monte-Carlo based path generation and pricing phases.

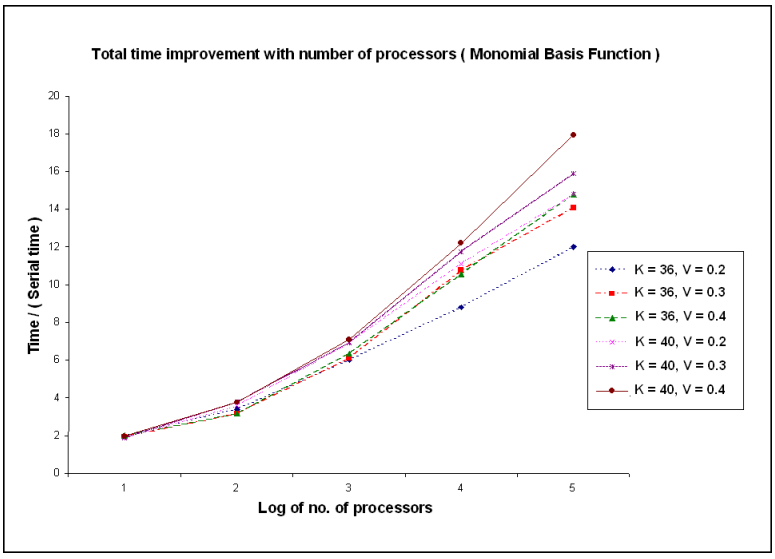


Figure 6: Total time variations with number of processors

6 Conclusion and future work

We show that the performance of the calibration phase in the LSM method can be scaled to quite an extent by dividing this phase into two subphases: SVD computations and price vector updates, and then performing the independent SVD computations in parallel. After doing this, the price vector update becomes the bottleneck. We also demonstrate that the overall performance of the full LSM method scales well, when in addition the Monte Carlo path generation and pricing phases are parallelized.

There are a number of interesting opportunities for further investigation.

1. When the number of processors is large compared to the number of SVD computations, then the individual SVD computations can be parallelized using parallel linear algebra implementations like ScaLAPACK [5] and PLAPACK [3].

2. When there are multiple maturity dates, there are further opportunities for optimization. Suppose we have to price the American option for different maturiy, say t_1, t_2, $\cdots$, t_N, where $t_i \leq t_{i+1}$, $i = 1, 2, \cdots N-1$. The same paths can be used for pricing for different maturity. This allows for reuse of some of the SVD computations.

3. When there are multiple strike prices, since LSM method includes only those paths in the regression for which the option is in the money, the design matrix A at any time step for one strike price is actually a subset or superset of the design matrix at that time step for another strike price. Hence, incremental SVD computation techniques ([19] and [6]) can be used to avoid multiple SVD computations for different strike prices.

References

[1] *IBM System Blue Gene/P*. http://www-03.ibm.com/servers/deepcomputing/bluegene.html.

[2] *Message Passing Interface*. http://www-unix.mcs.anl.gov/mpi/.

[3] *Parallel Linear Algebra Package*. http://www.cs.utexas.edu/~plapack/.

[4] *QuantLib: An open-source library for quantitative finance*. http://quantlib.org.

[5] *The ScaLAPACK project*. http://www.netlib.org/scalapack/.

[6] C. G. Baker, *A block incremental algorithm for computing dominant singular subspaces*, Master's Thesis, The Florida State University, 2004.

[7] L. Ballabio, *Implementing QuantLib*. http://luigi.ballabio.googlepages.com/qlbook.

[8] N. Bolia and S. Juneja, *Monte carlo methods for pricing financial options*, in SADHANA-ACADEMY PROCEEDINGS IN ENGINEERING SCIENCES, vol. 30, 2005, pp. 347–385.

[9] M. Broadie and J. B. Detemple, *Option pricing: Valuation models and applications*, MANAGEMENT SCIENCE, 50 (2004), pp. 1145–1177.

[10] R. H. Chan, C. Y. Wong, and K. M. Yeung, *Pricing multi-asset american-style options by memory reduction monte carlo methods*, APPLIED MATHEMATICS AND COMPUTATION, 179 (2006), pp. 535–544.

[11] D. Duffie, *Dynamic Asset Pricing Theory*, Princeton University Press, 1996.

[12] P. Glasserman, *Monte Carlo Methods in Financial Engineering*, Springer, New York, 2003.

[13] P. Glasserman and Y. Bin, *Number of paths versus number of basis functions in american option pricing*, ANNALS OF APPLIED PROBABILITY, 14 (2004), pp. 2090–2119.

[14] J. Hull, *Options, Futures, and Other Derivatives*, Prentice Hall, 6 ed., 2006.

[15] T. L. Lai and S. P. S. Wong, *Valuation of american options via basis functions*, IEEE TRANSACTIONS ON AUTOMATIC CONTROL, 49 (2004), pp. 374–385.

[16] N. B. Liberati and E. Platen, *On the efficiency of simplified weak taylor schemes for monte carlo simulation in finance*, in COMPUTATIONAL SCIENCE - ICCS 2004, PROCEEDINGS, vol. 3039 of LECTURE NOTES IN COMPUTER SCIENCE, 2004, pp. 771–778.

[17] F. A. Longstaff and E. S. Schwartz, *Valuing American options by simulation: A simple least-squares approach*, Review of Financial Studies, 14 (2001), pp. 113 – 147.

[18] N. Moreni, *A variance reduction technique for american option pricing*, PHYSICA A-STATISTICAL MECHANICS AND ITS APPLICATIONS, 338 (2004), pp. 292–295.

[19] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, *Incremental singular value decomposition algorithms for highly scalable recommender systems*, in Proceedings of the 5th International Conference in Computers and Information Technology, 2002.

[20] L. Stentoft, *Convergence of the least squares monte carlo approach to american option valuation*, MANAGEMENT SCIENCE, 50 (2004), pp. 1193–1203.