# Parallel Valuation of the Lower and Upper Bound Prices for Multi-asset Bermudan Options

Zhang Nan, Ka Lok Man

# Parallel Valuation of the Lower and Upper Bound Prices for Multi-Asset Bermudan Options

Nan Zhang[1] and Ka Lok Man[1]

Department of Computer Science and Software Engineering,
Xi'an Jiaotong-Liverpool University, China
{nan.zhang, ka.man}@xjtlu.edu.cn

**Abstract.** We present a parallel algorithm and its multi-threaded implementation for computing lower and upper bound prices of multi-asset Bermudan options. Our baseline sequential algorithm follows Longstaff and Schwartz's least-squares Monte Carlo method in computing the lower bound and Andersen and Broadie's simulation-based procedure with sub-optimality checking for the upper bound. The parallel implementation uses POSIX Threads for thread manipulation and Intel's MKL functions for random number generation and linear algebra operations. Tests were made on Intel x86 multi-core processors using the same option examples as the previous work, and the runtimes of the same computations were reduced from minutes to a few seconds.

**Keywords:** Parallel computing; option pricing; multi-asset Bermudan options; Monte Carlo simulation

## 1 Introduction

An equity call or put option is a financial contract that gives its holder the right without obligation to buy or sell a unit of the underlying asset at a pre-agreed strike price $K$ until a future expiration time $T$. In exchanging for this right, the buyer of such an option must pay the seller a certain amount of premium. The problem of option pricing is to compute the fair price of such an option contract. A buyer of a European-style option can only exercise the right of buying or selling at the expiration time $T$. In contrast, an American-style option can be exercised any time before $T$. Closed-form analytical formulas have been derived for the prices of many European options under a variety of models, the most widely-used being the Black-Scholes formula [3] under the geometric Brownian motion model. However, for American options, because of the early-exercise feature, closed-form expressions have not been found. For this reason, their prices are often computed using numerical procedures.

Various finite-difference schemes and lattice-based methods are such procedures often used to price American options. With these methods the early-exercise boundary of an American option can be easily determined. Whenever the payoff from exercising exceeds the option's continuation value at an exercise time it is assumed that the option should be exercised. However, when an option's value depends on a basket of assets it is not efficient to model the dynamics of the asset prices using either of these two

James J. Park et al. (Eds.): NPC 2012, LNCS 7513, pp. 000-000, 2012.

methods. Instead, simulation-based approaches can easily incorporate multiple assets, but the determining of continuation values at early exercise times is not as easy.

Regression-based methods [5, 13, 11] have been proposed to estimate the continuation values, and, thus, to enable Monte Carlo simulations to be applied in computing the prices of multi-asset American options. Longstaff and Schwartz [11] apply linear least-squares regression at every exercise time spot using observed continuation values on all in-the-money paths to find the coefficients of an assumed linear relationship between true continuation values and prices of the underlying assets. Algorithms like this produce low-biased estimations for the true price of an American option because the exercise strategies generated using the regressions are often sub-optimal. Hence, these algorithms compute lower bounds for the prices of multi-asset American options.

Andersen and Broadie [2] proposed a duality-based algorithm that computes upper bounds for multi-asset American options. The algorithm works with any lower bound estimator and presents the upper bound as the sum of the lower bound and a penalty term. The penalty term is a non-negative quantity representing a compensation for the potential incorrect exercise decisions made by following a sub-optimal strategy. Nested Monte Carlo simulations were used in [2] to compute the term. Optimisations were proposed in [4] for improving and accelerating the computing of the lower and upper bounds.

We present in this paper a parallel algorithm for computing the lower and upper bounds of multi-asset Bermudan[1] options. The algorithm parallelises the least-squares Monte Carlo lower bound estimator [11] and the primal-dual upper bound algorithm [2]. The sub-optimality checking discussed in [4] is also incorporated into the upper bound computation. The parallel algorithm was implemented via POSIX Threads, and works on shared-memory x86 multi-core processors. Highly-optimised functions from Intel's Math Kernel Library (MKL) [9, 10] were used in the implementation for linear algebra operations and random number generation. Experiments on an entry-level dual-core processor (2.4GHz Intel P8600) showed that the computational times for the same five-asset max-call Bermudan options were reduced from several minutes (as reported in [4] Table 3) to just a few seconds.[2] All the source codes for this work is freely available via email.[3]

In Section 2 we briefly present the lower and upper bound algorithms [11, 2, 4]. The parallel algorithm is presented in Section 3. Experimental results are reported in Section 4. Related work is discussed in Section 5. Conclusions and future work are found Section 6.

## 2 Sequential computing the lower and upper bounds

Assume a Bermudan option has $d$ exercise opportunities within time period 0 to $T$. These exercise times are denoted by $0 < t_1 < t_2 < \cdots < t_d = T$. The Bermudan

[1] Bermudan options are American-style options that can be exercised at a series of discrete time spots.

[2] Admittedly, the tests in [4] were made on a 2.0GHz Intel Pentium 4, a model much older than the processor we used in our tests.

[3] Email: nan.zhang@xjtlu.edu.cn

option's value depends on a basket of $n$ assets whose state $\boldsymbol{S}_i = (S_i^1, S_i^2, \ldots, S_i^n)$[4] at time $t_i$, $i \in [1, d]$, follows a vector-valued Markov process on $\mathbb{R}^n$ with initial value $\boldsymbol{S}_0$. Let $B_t$ denote the time $t$ value of one unit of cash invested in a risk-free money market account at time 0. Let $h_i$ denote the payoff from exercising the option at the stopping time $t_i$, where $t_i \in \mathcal{F} = \{t_1, t_2, \ldots, t_d\}$. The price $Q_0$ of the Bermudan option is, therefore, $Q_0 = \sup_{t_i \in \mathcal{F}} \mathbb{E}_0(h_i/B_i)$, where $\mathbb{E}_i$ denotes the expectation conditional on the information available until time $t_i$. The algorithm described in [11] computes not $Q_0$ but $L_0$, where $L_0 = \mathbb{E}_0(h_i/B_i) \leq Q_0$. The quantity $L_0$ is a low-biased price of the option obtained by following some specif c exercise policy rather than the optimal one.

To compute $L_0$ using Monte Carlo simulations we have to estimate the continuation values at each exercise time, so that on a path $j$ at time $t_i$, $i \in [1, d]$, if the immediate payoff exceeds the continuation value the option is assumed to be exercised. The computation starts from time $t_d = T$, at which it is assumed that the option is exercised for all in-the-money paths. The payoffs $h_d$ on all simulated paths are then used as the observed continuation values for the estimation of the continuation value at time $t_{d-1}$. The algorithm [11] assumes a linear relationship between the true continuation value $C_{d-1}$ at time $t_{d-1}$ and the state variable $\boldsymbol{S}_{d-1}$ such that $C_{d-1} = e_{d-1}^1 f_1(\boldsymbol{S}_{d-1}) + e_{d-1}^2 f_2(\boldsymbol{S}_{d-1}) + \ldots + e_{d-1}^m f_m(\boldsymbol{S}_{d-1})$. The $m$ functions $f_1, f_2, \ldots, f_m$ are the pre-def ned basis functions, and $e_{d-1}^1, e_{d-1}^2, \ldots, e_{d-1}^m$ are the coeff cients whose values are estimated by the following least-squares regression.

Assume totally $N_R$ simulation paths are launched and at time $t_{d-1}$ there are $p$ in-the-money paths. We use $1, 2, \ldots, p$[5] to index these in-the-money paths. We form a $p$-by-$m$ ($p$ rows and $m$ columns) design matrix $A_{d-1}$ whose $j$th row, $j \in [1, p]$, is $\boldsymbol{A}_{d-1}^j = (f_1(\boldsymbol{S}_{d-1}^j), f_2(\boldsymbol{S}_{d-1}^j), \ldots, f_m(\boldsymbol{S}_{d-1}^j))$, where $\boldsymbol{S}_{d-1}^j$ is the $n$-dimensional vector-valued state variable at time $t_{d-1}$ on the $j$th in-the-money path. Then we form the $p$-dimensional vector $\boldsymbol{V}_{d-1} = (B_{d-1}h_{d-1}^1/B_d, B_{d-1}h_{d-1}^2/B_d, \ldots, B_{d-1}h_{d-1}^p/B_d)$, where $B_{d-1}h_{d-1}^j/B_d$ is the observed discounted continuation value on the $j$th in-the-money path at time $t_{d-1}$. Now we can set up a linear system and solve the $m$-dimensional vector $\boldsymbol{E}_{d-1} = (e_{d-1}^1, e_{d-1}^2, \ldots, e_{d-1}^m)$ for the exercise time $t_{d-1}$ by minimising the merit function $\chi^2 = \Sigma_{i=1}^p (V_{d-1}^i - \boldsymbol{A}_{d-1}^i \boldsymbol{E}_{d-1}{}^{\mathrm{T}})^2$, where $V_{d-1}^i$ is the $i$th component of vector $\boldsymbol{V}_{d-1}$ and $\boldsymbol{A}_{d-1}^i$ is the $i$th row of the matrix $A_{d-1}$. The merit function is a measure for the aggregated difference between the observed continuation values and the estimated continuation values.

The linear system can be solved by using either QR or LQ factorisation, depending on whether it is over-determined or under-determined. Alternatively, the more robust singular value decomposition can be used at a higher computational cost. In our implementation the MKL function `LAPACKE_dgels` is used for solving the linear system. More details about the MKL functions for the factorisations can be found in the online reference [10]. Once $\boldsymbol{E}_{d-1}$ has been computed the $p$-dimensional vector $\hat{\boldsymbol{C}}_{d-1}{}^{\mathrm{T}} = A_{d-1}\boldsymbol{E}_{d-1}{}^{\mathrm{T}}$ can be derived, which contains the estimated continuation values for the $p$ in-the-money paths. This process proceeds backwards from time $t_{d-1}$ to $t_1$

[4] We use the subscript $i$ for $t_i$. So $\boldsymbol{S}_i$ is actually $\boldsymbol{S}_{t_i}$

[5] These are not their indexes in the whole $N_R$ simulated paths.

and outputs the $d$-by-$m$ coeff cient matrix $E$, whose $i$th row, $i \in [1, d-1]$, stores the coeff cients for the exercise time $t_i$. The $d$th row of $E$ is f lled by zeroes.

After the estimated exercise policy coeff cient matrix $E$ is obtained, another $N_L$ simulation paths are launched. On a $j$th path at the f rst exercise time $t_i$, $i \in [1, d]$, when $h_i^j > (f_1(\boldsymbol{S}_i^j), f_2(\boldsymbol{S}_i^j), \ldots, f_m(\boldsymbol{S}_i^j))\boldsymbol{E}_i{}^{\mathrm{T}}$, the option is assumed to be exercised, and the time 0 value of the resulted cashf ow is $h_i^j/B_i$. The low-biased option price $L_0$ is the mean of such cashf ows on all the $N_L$ simulated paths.

The upper bound $U_0$ of the option is def ned in [2] as $U_0 = L_0 + \Delta_0$, where $\Delta_0$ is a penalty term def ned as $\Delta_0 = \mathbb{E}_0(\max_{t_i \in \mathcal{F}}(h_i/B_i - \pi_i))$. The process $\pi_i$ is a martingale def ned in [4] by $\pi_{i+1} = \pi_i + L_{i+1}/B_{i+1} - Q_i/B_i$ at time $t_{i+1}$, $i \in [1, d-1]$. At time 0 and time $t_1$ it is def ned as $\pi_0 = L_0$ and $\pi_1 = L_1/B_1$. The process $L_{i+1}/B_{i+1}$ is the discounted lower bound price at time $t_{i+1}$ and is computed by $L_{i+1}/B_{i+1} = \mathbb{E}_{i+1}(h_{\tau_{i+1}}/B_{\tau_{i+1}})$, where $\tau_{i+1}$ is the f rst exercise time instance starting from time $t_{i+1}$ at which exercise is indicated. The algorithm in [4] with sub-optimality checking suggests computing $\pi_i$ only at those exercise times when exercise is suggested.

To estimate $\Delta_0$ we simulate $N_H$ paths, each of which has $d+1$ time steps, corresponding to the $n$-dimensional vector state variables $\boldsymbol{S}_0, \boldsymbol{S}_1, \ldots, \boldsymbol{S}_d$. On a $j$th simulated path, starting from time $t_1$ we ignore all exercise times at which continuation is suggested by the exercise policy. For a time $t_i$, $i \in [1, d]$, if exercise is suggested, we follow one of the two procedures listed below.

1. If time $t_i$ is the f rst exercise time on the $j$th path at which exercise is suggested, we set $\pi_i^j$ to the discounted option's payoff, which is also the discounted option value in this case, and so we have $\pi_i^j = h_i^j/B_i$. We then launch $N_S$ simulation trials to estimate $Q_i/B_i = \mathbb{E}_i(h_{\tau_i}/B_{\tau_i})$. This is the discounted option's continuation value from time $t_i$ if continuation is enforced. As above, stopping time $\tau_i$ is the f rst exercise time instance starting from time $t_i$ at which exercise is suggested. The penalty term $\Delta_i^j$ is set to 0.
2. If, otherwise, time $t_i$ is not the f rst exercise time on the $j$th path at which exercise is suggested, we set $\pi_i^j = \pi_l^j + h_i^j/B_i - Q_l/B_l$, where $t_l$ is the previous exercise time on the $j$th path at which exercise is suggested. By the time when $\pi_i^j$ is computed the values of $\pi_l^j$ and $Q_l/B_l$ are already available. We then launch $N_S$ simulation trials to estimate $Q_i/B_i = \mathbb{E}_i(h_{\tau_i}/B_{\tau_i})$ for later use. The penalty term $\Delta_i^j$ in this case is set to $\Delta_i^j = h_i^j/B_i - \pi_i^j$.

The penalty term $\Delta^j$ for the $j$th path is then set to $\Delta^j = \max(\Delta_1^j, \Delta_2^j, \ldots, \Delta_d^j)$. The penalty terms for the exercise times at which continuation is suggested are set to zeroes. After the penalty terms have been computed for all the $N_H$ paths, the upper bound increment $\Delta_0$ is set to the mean of all the penalty terms. The upper bound $U_0$ is therefore $U_0 = L_0 + \Delta_0$. Fig. 1 shows an example path of the simulation using a single-asset Bermudan call option.

Note that the above-listed procedure is derived from the simplif ed def nition of $\pi_i$ and Proposition 4.1 in [4]. If $\hat{L_0}$ is the lower bound estimation obtained by $N_L$ simulation trials with a sample standard deviation $\hat{s_L}$ and $\hat{\Delta_0}$ is the estimation for the increment using $N_H$ trials with a sample standard deviation $\hat{s_\Delta}$, as in [2], a 100(1-$\alpha$)%-probability conf dence interval for the price $Q_0$ of the Bermudan option can be

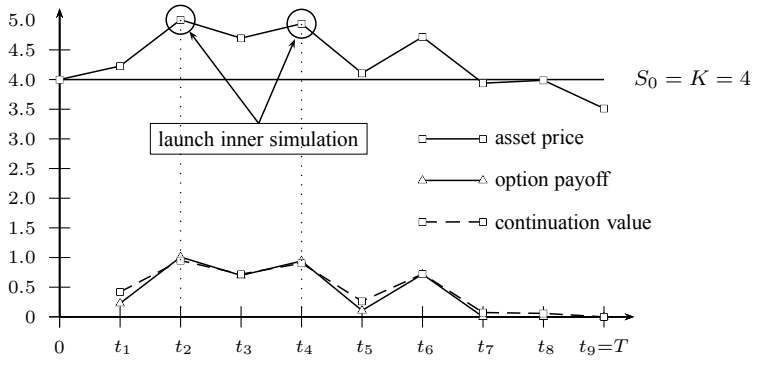



Fig. 1: A simulation path for a single-asset Bermudan call option, which can be exercised at times $t_1, t_2, \ldots, t_9 = T$. On this path, at times $t_2$ and $t_4$ the payoff exceeds the continuation value, and so the inner simulation is only launched at these two time spots.

computed as $[\hat{L_0} - z_{1-\alpha/2}\hat{s_L}/\sqrt{N_L},\ \hat{L_0} + \hat{\Delta_0} + z_{1-\alpha/2}\sqrt{\hat{s_L}^2/N_L + \hat{s_\Delta}^2/N_H}]$ with $z_x$ denoting the $x$th percentile of a standard Gaussian distribution.

## 3 The parallel computing of the lower and upper bounds

In computing the $L_0$ and $\Delta_0$, besides the procedures explained in Section 2, there are auxiliary steps, such as generating random numbers and constructing simulation paths for computing the coeff cient matrix $E$. A complete sequence of the steps for computing the $L_0$ and $\Delta_0$ is found in Fig. 2.

In a parallel computer system having $c$ processors, we denote the processors by $p_1, p_2, \ldots, p_c$. In our program, the parallelisation was achieved through POSIX Threads which works on shared-memory multi-processor systems. However, the parallel algorithm is not conf ned by such platforms. The construction of the $N_R$ simulation paths for estimating the coeff cient matrix $E$ needs $ndN_R$ standardised normally distributed random numbers ($\mathcal{N}(0,1)$). With $c$ processors, this task is equally divided such that each processor generates $ndN_R/c$ random numbers and then constructs $N_R/c$ paths. For each processor $p_i$, $i \in [1, c]$, it starts generating random numbers from the $(i-1)ndN_R/c$ position in the stream, so that there is no overlap between the processors.

The computing of the coeff cient vectors $\boldsymbol{E}_{d-1}, \boldsymbol{E}_{d-2}, \ldots, \boldsymbol{E}_1$ needs $d-1$ iterations. In each of the iterations, cross-sectional information needs to be collected, which makes this phase not as natural to be parallelised as the others. However, the MKL function `LAPACKE_dgels` and other functions for linear algebra we used support multithreading, and as a result of that this phase was not explicitly threaded.

After obtaining the matrix $E$, the lower bound $L_0$ is computed over $N_L$ paths where maximumly $ndN_L$ random numbers may be needed. Each of the $c$ processors completes an equal fraction of this task.

The estimation of $\Delta_0$ is also equally divided among the $c$ processors. Like the computation of $L_0$, the exact number of random numbers to be needed is unknown beforehand. However, in this case, the maximum number $nd^2N_HN_S$ is so large that, often, they cannot all be generated and stored as 8-byte double-precision f oats. Hence, we

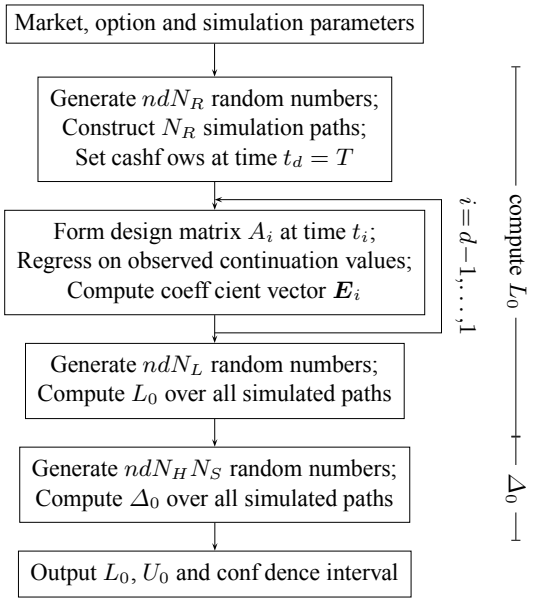



Fig. 2: Steps for computing lower bound $L_0$ and upper bound $U_0 = L_0 + \Delta_0$.

only generated $ndN_H N_S$ random numbers with each processor generating an equally-lengthed segment without overlapping. For any processor $p_i$, $i \in [1, c]$, in the simulation if the numbers in its segment are used up, it will roll back to the beginning of the segment.

When the program starts, the main thread is bound onto processor $p_1$. When using the $c$ processors for explicit threading, the main thread spawns $c - 1$ child threads and binds each of them onto one of the processors $p_2, p_3, \ldots, p_c$. After generating the child threads, the main thread is given an equal fraction of the task to complete. During this course we disable the multi-threading feature of the MKL functions to avoid resource contention [9]. This can be done by using the MKL function `mkl_set_num_threads` to set the number of threads used by MKL to 1. However, when estimating the coeff - cient vectors we enable the multi-threading feature by setting the number to $c$. Nevertheless, the MKL functions may dynamically adjust the number of threads they actually use during the computation. The sequence of the parallel computing is shown in Algorithm 1.

## 4 Experimental results

We used the same multi-asset symmetric Bermudan max call options [2, 4] to test our sequential and parallel implementations. Each of these Bermudan options depends on a basket of $n$ uncorrelated equity assets. The risk-neutral dynamics of their prices all follow geometric Brownian motion processes. For a $j$th, $j \in [1, n]$, asset in the basket, its price dynamic at time $t$ is modelled by the equation $dS_t^j = (r - \delta)S_t^j dt + \sigma S_t^j dW_t^j$, where $r$ is the annual risk-free interest rate, $\delta$ is the dividend yield of the equity asset,

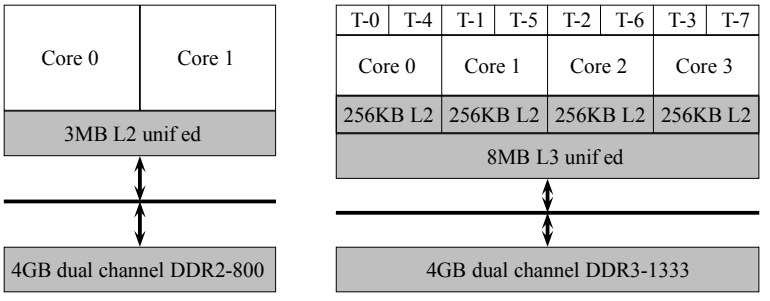
**Algorithm 1**: Parallel computing of $L_0$ and $U_0$ using $c$ processors $p_1, p_2, \ldots, p_c$.

**Input**: Market, option and simulation parameters.
**Output**: $L_0$, $\Delta_0$, $U_0$ and a 100(1-$\alpha$)%-probablity conf dence interval.

**begin**
  Bind main thread onto processor $p_1$.
  Set the number of threads used by MKL to 1.
  Spawn $c-1$ child threads and bind each of them onto one of $p_2, p_3, \ldots, p_c$.
  **with** $c$ threads **in parallel do**
    Generate $ndN_R$ standardised normally distributed random numbers.
    Construct $N_R$ simulation paths.
  Allow main thread to reside on any one of the $c$ processors.
  Set the number of threads used by MKL to $c$.
  Compute coeff cient matrix $E = (\mathbf{0}, \boldsymbol{E}_1, \ldots, \boldsymbol{E}_{d-1}, \mathbf{0})^{\mathrm{T}}$.
  Bind main thread onto processor $p_1$.
  Set the number of threads used by MKL to 1.
  Spawn $c-1$ child threads and bind each of them onto one of $p_2, p_3, \ldots, p_c$.
  **with** $c$ threads **in parallel do**
    Generate $ndN_L$ standardised normally distributed random numbers.
    Compute cashf ows on the $N_L$ simulated paths.
  Set $L_0$ to the mean of the cashf ows.
  Spawn $c-1$ child threads and bind each of them onto one of $p_2, p_3, \ldots, p_c$.
  **with** $c$ threads **in parallel do**
    Generate $ndN_H N_S$ standardised normally distributed random numbers.
    Compute increments of the $N_H$ simulated paths.
  Set $\Delta_0$ to the mean of the increments.
  Set $U_0 = L_0 + \Delta_0$.
  Compute the conf dence interval.
**end**

$\sigma$ is the annual volatility and $W_t^j$ is a standard Brownian motion process. The time $t$ value of one unit of cash invested at time 0 is $B_t = \exp(rt)$. The $d$ exercise times for the options are equally spaced at times $t_i = iT/d$, $i = 1, 2, \ldots, d$. On the simulated paths, from a time $t_i$ to time $t_{i+1}$, $i \in [0, d-1]$, the price of a $j$th asset is updated by $S_{i+1}^j = S_i^j \exp((r - \delta - \sigma^2/2)(t_{i+1} - t_i) + \sigma\epsilon\sqrt{t_{i+1} - t_i})$, where $\epsilon$ is a $\mathcal{N}(0,1)$ random number. The options' strike prices are $K$, and so the payoff of such an option at time $t_i$, $i \in [1, d]$, is $h_i(\boldsymbol{S}_i) = (\max(\boldsymbol{S}_i^1, \boldsymbol{S}_i^2, \ldots, \boldsymbol{S}_i^n) - K)^+$. For comparison purposes we used the same values for the market and simulation parameters as in [2] and [4]. We set $K = 100$, $r = 0.05$, $\delta = 0.1$, $T = 3$, $\sigma = 0.2$, $d = 9$, $N_R = 200,000$, $N_L = 2,000,000$, $N_H = 1,500$, $N_S = 1,000$ and $\boldsymbol{S}_0 = (S, S, \ldots, S)$. For the f ve-asset Bermudan options we used the same 18 basis functions found in Appendix A.2 [4]. For the three-asset Bermudan options we used the f rst 10 functions and the f rst 8 for the two-asset Bermudan options.

The sequential and parallel implementations were programed in C/C++. The NPTL (native POSIX thread library) 2.12.1 was used for the threading. The tests were made on Intel multi-core systems running Ubuntu Linux 10.10 (64 bit). The binary executables were compiled by the Intel compiler icpc 12.0 for Linux.




(a) The Intel P8600 system. (b) The Intel Core i7 2600 system.

Fig. 3: Multi-core machines used in tests.

We f rst present in Table 1 the results from three groups of tests using the Bermudan max call options when $n = 2, 3, 5$. Some data reported in Table 2 [2] and Table 3 [4] are included in the table for comparison. These groups of tests were made on a 2.4GHz dual-core Intel P8600 processor (Fig. 3(a)). We show the results on this entry-level multi-core processor to demonstrate that our sequential and parallel implementations are practical on common personal computers.

From the data reported in Table 1 we can see that the $\hat{L_0}$s and $\hat{U_0}$s produced by the parallel program are very close to the data reported in [2] and [4], yet the runtimes were much shorter. We also ran tests using the seven f ve-asset Bermudan options on a 3.4GHz (turbo boost to 3.8GHz) quad-core (8 threads with hyperthreading) Intel Core i7 2600 processor (Fig. 3(b)). It took the sequential program 27.19 seconds to compute the $\hat{L_0}$s and $\hat{U_0}$s for the seven options and the parallel program 11.66 seconds using eight threads.

## 5 Related work

Choudhury et al. [6] parallelised Longstaff and Schwartz's lower bound algorithm. The lower bound estimating algorithm was divided into three phases: path simulation, calibration (in which the least-squares regressions are performed) and valuation. They used singular value decomposition in preference of its robustness, and they explicitly parallelised the calibration phase, exploiting the fact that the decomposition of the design matrices is mutually independent. The parallel algorithm was implemented using the Quantlib open-source library [1]. A speedup of 18 times for the whole lower bound algorithm was achieved on a 32 processor Blue Gene system. Doan et al. [7] presented two parallel Monte Carlo based algorithms for pricing multi-asset Bermudan options. The f rst of the two algorithms parallelises the approach proposed by Ibanez and Zapatero [8] which computes the optimal exercise boundary of an option through parameterisation. The second parallelises the algorithm proposed by Picazo [12] which computes continuation values through classif cation. Their algorithms were evaluated in a desktop grid environment.

Table 1: Computational and timing results on Intel dual-core 2.4GHz P8600.

| $\boldsymbol{S}_0$ | $\hat{L}_0$ | $T_L$ | $\hat{U}_0$ | $T_U$ | $T_P$ | $T_S$ | $T'_S$ | 95% CI | 95% CI' |
|---|---|---|---|---|---|---|---|---|---|
| $n = 2, m = 8$ | | | | | | | | | |
| 90 | 8.048 | 1.80 | 8.051 | 0.48 | 2.28 | 2.59 | | [8.043, 8.068] | [8.053, 8.082] |
| 100 | 13.886 | 1.71 | 13.891 | 0.52 | 2.23 | 2.88 | | [13.881, 13.912] | [13.892, 13.934] |
| 110 | 21.341 | 1.82 | 21.345 | 0.59 | 2.41 | 3.14 | | [21.335, 21.370] | [21.316, 21.359] |
| Total runtime | | | | | 6.91 | 8.61 | | | |
| $n = 3, m = 10$ | | | | | | | | | |
| 90 | 11.254 | 2.06 | 11.257 | 0.70 | 2.75 | 4.00 | | [11.249, 11.277] | [11.265, 11.308] |
| 100 | 18.653 | 2.45 | 18.660 | 0.76 | 3.21 | 4.50 | | [18.647, 18.684] | [18.661, 18.728] |
| 110 | 27.546 | 2.66 | 27.556 | 0.83 | 3.49 | 4.82 | | [27.539, 27.584] | [27.512, 27.663] |
| Total runtime | | | | | 9.44 | 13.31 | | | |
| $n = 5, m = 18$ | | | | | | | | | |
| 70 | 3.899 | 2.74 | 3.901 | 1.02 | 3.76 | 5.63 | 46.2 | [3.895, 3.912] | [3.880, 3.913] |
| 80 | 9.024 | 3.30 | 9.029 | 1.07 | 4.37 | 6.52 | 54 | [9.019, 9.046] | [8.984, 9.033] |
| 90 | 16.612 | 4.25 | 16.618 | 1.17 | 5.42 | 7.88 | 85.2 | [16.607, 16.641] | [16.599, 16.686] |
| 100 | 26.125 | 5.10 | 26.143 | 1.23 | 6.33 | 9.03 | 121.8 | [26.119, 26.170] | [26.093, 26.194] |
| 110 | 36.746 | 5.30 | 36.771 | 1.28 | 6.58 | 9.33 | 151.2 | [36.740, 36.802] | [36.681, 36.819] |
| 120 | 47.890 | 5.31 | 47.909 | 1.33 | 6.63 | 9.44 | 168.6 | [47.883, 47.943] | [47.816, 48.033] |
| 130 | 59.352 | 5.33 | 59.379 | 1.33 | 6.66 | 9.47 | 198 | [59.345, 59.415] | [59.199, 59.437] |
| Total runtime | | | | | 39.75 | 57.29 | | | |

Notes: The estimations $\hat{L}_0$, $\hat{U}_0$ and the 95% CI were computed by our parallel program. $T_L$ is the parallel runtime for computing the $\hat{L}_0$ and $T_U$ is the parallel runtime for computing the $\hat{U}_0$. $T_P$ is the total parallel runtime, $T_P = T_L + T_U$. $T_S$ is the runtime of our sequential program in computing $\hat{L}_0$ and $\hat{U}_0$. $T'_S$ is the runtime reported in the middle panel of Table 3 [4]. There, the original runtimes were measured in minutes. Here, they are converted to seconds. The data of 95% CI' for $n = 2, 3$ are copied from Table 2 [2], and that for $n = 5$ are copied from the middle panel of Table 3 [4]. No timing result was found in Table 2 [2], and, therefore, data for $T'_S$ when $n = 2, 3$ are not included in the table.

# 6 Conclusions

We have presented a parallel algorithm and its implementation that computes the lower and upper bound prices of multi-asset Bermudan options. The algorithm parallelises the lower bound estimator proposed in [11] and the upper bound estimator proposed in [2] with the sub-optimality checking discussed in [4]. The parallel implementation uses POSIX Threads for multi-threading and Intel's high-performance MKL functions for random number generation, QR or LQ factorisation and vector and matrix operations. Tests were made using the same symmetric f ve-asset Bermudan max call options as in [2, 4]. The results on an entry-level 2.4GHz dual-core Intel P8600 showed that it took the parallel program a few seconds to price anyone of the Bermudan options, while in [4], some of the same tasks needed several minutes on a 2.0GHz Intel Pentium 4.

The factorisation of the design matrix for different exercise times is independent of each other, although the overall procedure for the least-squares regressions is sequential. This fact can be exploited to explicit parallelise the factorisations as the approach presented in [6]. The reason we did not exploit this opportunity in our parallel algorithm was that through testing we found that in the tested examples the cost of the explicit

multi-threading exceeded that of the implicit multi-threading of the MKL functions. However, in our future work, for examples of larger scales we may explicit parallelise the factorisations at different exercise times to achieve better scalability.

Another possible direction for future work is that we may work on implementing the parallel algorithm on general-purpose GPUs using CUDA or OpenCL to achieve further better performance.

## References


1. Quantlib: A Free/Open-source Library for Quantitative Finance, http://quantlib.org/index.shtml
2. Andersen, L., Broadie, M.: Primal-Dual Simulation Algorithm for Pricing Multidimensional American Options. Management Science 50(9), 1222–1234 (Sep 2004)
3. Black, F., Scholes, M.: The Pricing of Options and Corporate Liabilities. The Journal of Political Economy 81(3), 637–659 (1973)
4. Broadie, M., Cao, M.: Improved Lower and Upper Bound Algorithms for Pricing American Options by Simulation. Quantitative Finance 8(8), 845–861 (Dec 2008)
5. Carriere, J.F.: Valuation of the Early-Exercise Price for Options using Simulations and Nonparametric Regression. Insurance: Mathematics and Economics 19(1), 19–30 (Dec 1996)
6. Choudhury, A.R., King, A., Kumar, S., Sabharwal, Y.: Optimizations in Financial Engineering: The Least-Squares Monte Carlo Method of Longstaff and Schwartz. In: Proceedings of the 2008 IEEE International Symposium on Parallel and Distributed Processing (IPDPS). pp. 1–11. IEEE (Apr 2008)
7. Doan, V., Gaikwad, A., Bossy, M., Baude, F., Stokes-Rees, I.: Parallel Pricing Algorithms for Multi-Dimensional Bermudan/American Options using Monte Carlo Methods. Mathematics and Computers in Simulation 81(3), 568–577 (2010)
8. Ibáñez, A., Zapatero, F.: Monte Carlo Valuation of American Options through Computation of the Optimal Exercise Frontier. Journal of Financial and Quantitative Analysis 39(2), 253–275 (2004)
9. Intel Corporation: Intel Math Kernel Library for Linux OS: User's Guide (2011), Document Number: 314774-018US. http://software.intel.com/en-us/articles/intel-math-kernel-library-documentation/
10. Intel Corporation: Intel Math Kernel Library Reference Manual (2011), Document Number: 630813-044US. http://software.intel.com/en-us/articles/intel-math-kernel-library-documentation/
11. Longstaff, F.A., Schwartz, E.S.: Valuing American Options by Simulation: A Simple Least-Squares Approach. The Review of Financial Studies 14(1), 113–147 (2001)
12. Picazo, J.A.: American Option Pricing: A Classif cation-Monte Carlo (CMC) Approach. In: Monte Carlo and Quasi-Monte Carlo Methods 2000: Proceedings of a Conference Held at Hong Kong Baptist University. pp. 422–433. Springer (Dec 2000)
13. Tsitsiklis, J.N., Roy, B.V.: Optimal Stopping of Markov Processes: Hilbert Space Theory, Approximation Algorithms, and An Application to Pricing High-Dimensional Financial Derivatives. IEEE Transactions on Automatic Control 44(10), 1840–1851 (Oct 1999)