Pricing Bermudan Interest Rate Swaptions via Parallel Simulation under the Extended Multi-factor LIBOR Market Model

Zhang Nan, Ka Lok Man, Eng Gee Lim

Pricing Bermudan Interest Rate Swaptions via Parallel Simulation under the Extended Multi-Factor LIBOR Market Model

Nan Zhang[1], Ka Lok Man[1], and Eng Gee Lim[2]

[1] Department of Computer Science and Software Engineering,
Xi'an Jiaotong-Liverpool University, China
{nan.zhang, ka.man}@xjtlu.edu.cn
[2] Department of Electrical and Electronic Engineering,
Xi'an Jiaotong-Liverpool University, China
enggee@xjtlu.edu.cn

Abstract. We present a parallel algorithm and its implementation that computes lower and upper bounds for prices of Bermudan swaptions. The evolving of the underlying forward rates is assumed to follow the extended multi-factor LIBOR market model. We follow the Longstaff-Schwartz least-squares approach in computing a lower bound and the Andersen-Broadie duality-based procedure in computing an upper bound. Parallelisation in the implementation is achieved through POSIX threading. High-performance Intel MKL functions are used for regression and linear algebra operations. The parallel implementation was tested using Bermudan swaptions with different parameters on Intel multi-core machines. In all the tests the parallel program produced close results to those reported in the previous studies. Signif cant speedups were observed against an eff cient sequential implementation built for comparison.

Keywords: Parallel computing; Bermudan swaption pricing; LIBOR market model; Monte Carlo simulation

1 Introduction

An interest rate swap is an agreement between two parties to exchange cash f ows equal to interest at a pre-determined f xed rate on a notional principal for cash f ows equal to interest at a f oating rate on the same principal for a pre-determined number of years. The exchanges of cash f ows take place on pre-scheduled dates which usually span equal length of periods known as the accrual period, often three or six months. The f oating rate in most interest rate swaps is the LIBOR (London Interbank Offered Rate) rate. The party who pays out f xed-rate interests and receives f oating-rate interests is known as the f xed payer of the swap. A Bermudan swaption is an option which gives the holder the right to enter an interest rate swap at each date in a series of exercise dates, provided this right has not been exercised before. Bermudan swaptions are one of the most important and widely traded derivatives in f xed-income markets.

The LIBOR market model framework [4, 9, 11] and its extensions [2] are often used to compute the prices of Bermudan swaptions. Due to its large number of state variables,

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

lattice-based methods are not feasible for this model class, and the pricing generally requires Monte Carlo simulations. To make Monte Carlo simulation be able to handle the built-in early exercise feature in Bermudan-style options Longstaff and Schwartz [10] proposed the least-squares Monte Carlo approach, which determines the early exercise boundary through linear regressions. Prices for Bermudan options computed using this regression-based method are biased low to the true values, because the exercise strategies generated by the regressions are inferior to the optimal ones. For upper bounds on Bermudan options Andersen and Broadie [3] proposed a Monte Carlo algorithm that uses the duality representation of the options' value function. Combining the low- and high-biased pricing algorithms gives valid conf dence intervals for the true price of a Bermudan swaption.

However, the lower and upper bound pricing algorithms are computationally demanding. The upper bound algorithm requires nested simulations. To accelerate the algorithms we developed parallel solutions to the Longstaff-Schwartz method for lower bound and the Andersen-Broadie method for upper bound, and applied them to the pricing of Bermudan swaptions in the extended multi-factor LIBOR market model. The parallel algorithms are designed to work on shared-memory x86 multi-core processors that dominate today's PC markets. The parallelisation was achieved through f ne-tuned POSIX threads. Mathematical functions from Intel's highly optimised Math Kernel Library (MKL) [7, 8] were used for linear algebra operations. Various source code optimisations, such as common sub-expression elimination, were hard coded into the programs. The parallel implementation was tested on common Intel multi-core processors using the same examples as in [3]. The longest execution time was found in the valuation of the 8% f xed rate 11-year contract. On the quad-core 3.4GHz Intel Core i7-2600 used in the tests it took about eleven seconds to f nd the swaption's lower and upper bound prices using eight threads.

In Section 2 we brief y discuss the extended multi-factor LIBOR market model. The lower and upper bound estimations for Bermudan swaptions are discussed in Section 3. The parallel algorithm is presented in Section 4. Experimental results are reported in Section 5. Conclusions are summarised in Section 6.

2 Model setup

Def ne an increasing maturity structure $0 = t_0 < t_1 < t_2, \ldots, < t_K$ and time t_i price $P(t_i, t_j)$ of a zero-coupon bond paying off \$1 at time t_j for $i \leq j$ and $i, j \in \{0, 1, 2, \ldots, K\}$. The LIBOR market model in general does not put any restriction on the increasing maturity structure, but our implementation assumed that any two successive time spots in the structure span an equidistant accrual period, often three or six months in practice. While the function $P(,)$ can be def ned on any time spots not necessarily coinciding with the dates in the maturity structure, we def ne it on dates in the maturity structure to simplify the problem and, thus, to serve the purpose of our implementation. The discrete forward rate $F_{t_j}(t_i)$ for any t_i and t_j when $i \leq j$ and $i, j \in \{0, 1, 2, \ldots, K-1\}$ that applies to period between t_j and t_{j+1} observed at time t_i is def ned as

$$F_{t_j}(t_i) = F_j(i) = \frac{1}{\delta_j}\left(\frac{P(t_i,t_j)}{P(t_i,t_{j+1})} - 1\right), \quad \delta_j = t_{j+1} - t_j \tag{1}$$

With this definition for forward rates, the definition for $P(t_i,t_j)$ can be written as

$$P(t_i,t_j) = P(i,j) = \prod_{k=i}^{j-1}\left(\frac{1}{1+\delta_k F_k(i)}\right), \quad i \le j, \quad i,j \in \{0,1,2,\ldots,K\} \tag{2}$$

Note that in Equation (2) the production is performed up to time t_{j-1}, because the forward rate $F_{j-1}(i)$ applies to the period from time t_{j-1} to time t_j. In the extended LIBOR market model [2], knowing the initial forward rates $F_j(0)$ for all $j \in \{0,1,2,\ldots,K-1\}$ forward rates observed at future times can be approximated by

$$\hat{F}_j(i+1) = \hat{F}_j(i)\exp\left(\frac{\varphi(\hat{F}_j(i))}{\hat{F}_j(i)}\boldsymbol{\lambda}_j^{\mathrm{T}}(i)\left[\left(\hat{\boldsymbol{u}}_j(i) - \frac{1}{2}\frac{\varphi(\hat{F}_j(i))}{\hat{F}_j(i)}\boldsymbol{\lambda}_j(i)\right)\Delta_i + \boldsymbol{\epsilon}_i\sqrt{\Delta_i}\right]\right) \tag{3}$$

Equation (3) is obtained by applying Euler scheme to the dynamics of the forward rate in continuous time. For the equation to hold we have the obvious condition $i+1 \le j \le K-1$. Function $\varphi(.)$ in the equation is the skew function, $\boldsymbol{\lambda}_j(i)$ is the m-dimensional volatility vector, and $\hat{\boldsymbol{u}}_j(i)$ is the m-dimensional drift vector defined as

$$\hat{\boldsymbol{u}}_j(i) = \sum_{k=i+1}^{j}\boldsymbol{\lambda}_k(i)\frac{\delta_k\varphi(\hat{F}_k(i))}{1+\delta_k\hat{F}_k(i)} \tag{4}$$

The $\boldsymbol{\epsilon}_i$ in Equation (3) is a m-dimensional vector of independent standard Gaussian variables and $\Delta_i = t_{i+1} - t_i$. Using Equation (3) with Monte Carlo simulation paths of forward rates can be generated. The equation guarantees the generated rates are positive. It should be pointed out that 1) in computing the drift term $\hat{\boldsymbol{u}}_j(i)$ the summation starts from time t_{i+1}, thus excluding the term $\boldsymbol{\lambda}_i(i)\delta_i\varphi(\hat{F}_i(i))/(1+\delta_i\hat{F}_i(i))$, and 2) the same m-dimensional vector $\boldsymbol{\epsilon}_i$ applies to the generation of all forward rates $F_j(i+1)$ for $j = i+1, i+2, \ldots, K-1$ from $F_j(i)$.

A Bermudan swaption gives the holder the right of exercising to enter a swap agreement in which the holder pays fixed cashflows $\theta\delta_{j-1}$ at time t_j for $j = s+1, s+2, \ldots, e$, in exchange for LIBOR on a \$1 notional, assuming t_s being the first exercise date and t_e being the last payment date. Payments for periods between t_j and t_{j+1} are exchanged at time t_{j+1} (paid in arrears). A Bermudan swaption is characterised by three dates: the lockout date t_s, the last exercise date t_x and the final swap maturity t_e. Our implementation assumed that all these three dates coincide with dates in the maturity structure and $t_s \le t_x = t_{e-1} = t_{K-1}$, that is, the last exercise date t_x is the second last date in the maturity structure and the swap matures at time t_K. A Bermudan swaption characterised by t_s, t_x and t_e (assuming $t_e = t_{x+1}$), can be exercised once at any time between t_s and t_x. The first payments are exchanged at time t_{s+1}, and the last at time $t_e = t_{x+1}$.

For such a Bermudan swaption exercised at t_j for $j \in \{s, s+1, \ldots, x = e-1\}$ we define a strictly positive process $B(t_i,t_j)$ for $i \le j$ as

$$B(t_i, t_j) = B(i, j) = \sum_{k=j}^{x} \delta_k P(i, k+1) \tag{5}$$

The par-rate $R(t_i, t_j)$ observed at time t_i if the swaption is exercised at time t_j assuming the swap maturing at time $t_e = t_{x+1}$ is

$$R(t_i, t_j) = R(i, j) = \frac{P(i, j) - P(i, x+1)}{B(i, j)} \tag{6}$$

This rate is also known as the swap rate at time t_i that makes the payoff of the swap exercised at time t_j and maturing at time $t_e = t_{x+1}$ equal to zero, thus, being fair to both the parties. With the above def nitions the payoff $S(t_j)$ of the swaption exercised at time t_j for the f xed payer is

$$S(t_j) = S(j) = B(j, j)(R(j, j) - \theta)^+ = B(j, j) \left(\frac{1 - P(j, x+1)}{B(j, j)} - \theta \right)^+ \tag{7}$$

where θ is the f xed coupon rate. The time 0 value of this payoff $S(j)$ received at exercise time t_j is $P(0, j)S(j)$.

Equipped with the above-mentioned def nitions Monte Carlo simulation can be used to evaluate lower and upper bounds for prices of Bermudan swaptions in an extended m-factor LIBOR market model.

3 Lower and upper bounds for Bermudan swaption

Use Q_0 to denote the fair time 0 price of a Bermudan swaption. The price equals to the discounted payoff of the swaption if it is exercised at an optimal exercise time between t_s and t_x. Using $\aleph$ to denote the set $\{t_s, t_{s+1}, \ldots, t_x\}$ of exercise times, Q_0 is computed by

$$Q_0 = \sup_{\tau \in \aleph} \mathbb{E}^Q \left(P(0, \tau) S(\tau) \right) \tag{8}$$

where $\mathbb{E}^Q$ denotes the expectation under the spot measure (see [2] for more details about the spot measure), $S(\tau)$ is the payoff to the f xed payer if exercise takes place at time τ and $P(0, \tau)$ discounts the payoff to time 0. If τ^* is the optimal exercise strategy that solves Equation (8) the task of computing Q_0 is down to f nding τ^*.

With Monte Carlo simulation various methods have been proposed to approximate the optimal exercise strategy τ^*. Andersen [1] used an optimisation procedure to determine the exercise boundary. Sided with the work in [6], we applied the regression-based Longstaff-Schwartz method [10] in searching for the approximations. This method computes low-biased price $L_0 = \mathbb{E}^Q(P(0, \tau)S(\tau)) \leq Q_0$ for a Bermudan swaption.

Knowing the initial forward curve, we f rst simulate N_R paths of forward rates, over which an exercise strategy is computed. On a particular path at an exercise time t_j for $j \in \aleph$ we use a constant c, the swap rate $R(j, j)$ observed at time t_j if exercise takes

place at that time, $R(j,j)^2$ and $R(j,j)^3$ as basis functions for the regression. After the strategy generation we launch another group of N_L paths with an $N_L \gg N_R$. The lower bound L_0 is estimated over these N_L paths based on the strategy.

The upper bound U_0 is computed by the simulation-based algorithm [3] proposed by Andersen and Broadie. The sub-optimality checking discussed in [5] was integrated into our implementation. The upper bound U_0 is set as the sum of the lower bound L_0 and a penalty term Δ_0. We launch N_H simulated paths to compute Δ_0 as $\Delta_0 = \mathbb{E}^Q(\max_{\tau\in\aleph}(P(0,\tau)S(\tau)-\pi_\tau))$. On any simulated path, at time t_i the computation for Δ_i takes place only if exercise is suggested by the exercise strategy. The computation follows one of the two procedures.

1. If time t_i is the first exercise time on the path at which exercise is suggested, π_i is set as $\pi_i = P(0,i)S(i)$, the discounted payoff from the exercise. We then launch N_S inner simulation trials to estimate the swaption's discounted continuation value $P(0,i)Q_i = \mathbb{E}^Q(P(0,j)S(j))$ where $t_j \in \{t_{i+1}, t_{i+2}, \ldots, t_x\}$ is the first exercise time instance at which exercise is suggested after time t_i. The penalty term Δ_i is initialised as 0.
2. If, otherwise, time t_i is not the first exercise time on the path at which exercise is suggested, we set $\pi_i = \pi_l + P(0,i)S(i) - P(0,l)Q_l$, where t_l is the previous exercise time on the path at which exercise is suggested. By the time when π_i is computed the values of π_l and $P(0,l)Q_l$ are already available. We then launch N_S inner simulation trials to estimate $P(0,i)Q_i = \mathbb{E}^Q(P(0,j)S(j))$, where $t_j \in \{t_{i+1}, t_{i+2}, \ldots, t_x\}$ is the first exercise time instance at which exercise is suggested after time t_i, for later use. The penalty term Δ_i is set as $\Delta_i = P(0,i)S(i) - \pi_i$.

Note that the above procedure is deduced from Proposition 4.1 in [5]. Applying Equation (15) in [5] to a Bermudan swaption, at exercise time t_i where exercise is suggested process π_i is defined as

$$\pi_i = \pi_{i-1} + P(0,i)S(i) - P(0,i-1)Q_{i-1} \tag{9}$$

If time t_i is the first exercise time on a path at which exercise is suggested, according to Proposition 4.1(1) in [5], $\pi_{i-1} = P(0,i-1)S(i-1)$. At an exercise time where continuation is suggested such as time t_{i-1}, the swaption's payoff is equal to its continuation value, and, so, we have $\pi_{i-1} = P(0,i-1)Q_{i-1}$. Substitute $P(0,i-1)Q_{i-1}$ for π_{i-1} in Equation (9) we get $\pi_i = P(0,i)S(i)$. The penalty term $\Delta_i = P(0,i)S(i) - \pi_i = 0$. If, however, time t_i is an exercise time, but not the first one, at which exercise is suggested, according to Proposition 4.1(2), $\pi_{i-1} = \pi_l - P(0,l)Q_l + P(0,i-1)S(i-1)$, where time t_l is the previous exercise time on the path at which exercise is suggested. Because at time t_{i-1} the swaption's payoff equals to its continuation value, we have $\pi_{i-1} = \pi_l - P(0,l)Q_l + P(0,i-1)Q_{i-1}$. Substitute this expression for π_{i-1} in Equation (9) we get $\pi_i = \pi_l + P(0,i)S(i) - P(0,l)Q_l$. Fig. 1 shows an example where inner simulations are launched at exercise times t_{s_1} and t_{s_4} where exercises are suggested.

If $\hat{L_0}$ is the lower bound estimation over N_L simulation trials with a sample standard deviation $\hat{s_L}$ and $\hat{\Delta_0}$ is the estimation for the penalty term using N_H trials with a sample standard deviation $\hat{s_\Delta}$, as in [3], a 100(1-α)%-probability confidence interval (CI) for the price Q_0 of the Bermudan swaption can be computed as

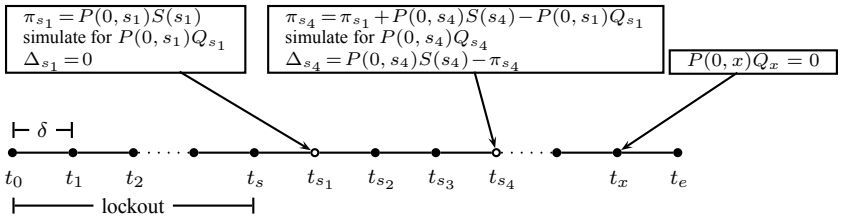


Fig. 1: Inner simulation on a constructed forward rate path. Note that time t_{s_1} is the first exercise time instance on the path at which exercise is suggested, and t_{s_4} is the second.

$$100(1-\alpha)\%\ \text{CI} = \left[\hat{L_0} - \frac{z_{1-\alpha/2}\hat{s_L}}{\sqrt{N_L}},\ \hat{L_0} + \hat{\Delta}_0 + z_{1-\alpha/2}\sqrt{\frac{\hat{s_L}^2}{N_L} + \frac{\hat{s_\Delta}^2}{N_H}}\right] \tag{10}$$

with z_x denoting the xth percentile of a standard Gaussian distribution.

4 The parallel computing

On a parallel computer with c processors, the computation for L_0 over N_L paths and that for Δ_0 over N_H paths are evenly distributed onto all processors through POSIX threads. In the generation of random numbers, each individual processor generates a segment for its own use, skipping a certain amount from the beginning of a random number stream. Take the ith processor (processor index starting from 0) as an example. In computing L_0 it will generate $m(N_L/c)(t_e/\delta)$ standardised normally distributed random numbers, m being the model factor. This segment starts from the $(im(N_L/c)(t_e/\delta))$th position of the random stream, assuming a constant accrual period divisible by the final swap maturity. The ith processor will then construct N_L/c complete paths and compute a $\hat{L_0}$ over these paths. The $\hat{L_0}$ over all N_L paths are obtained by averaging all $\hat{L_0}$s computed by each of the c processors.

The phase of optimal exercise strategy approximation is not explicitly threaded. The computation starts from the second last exercise time t_{x-1} and proceeds backwards until reaching exercise time instance t_s. At each time step payoffs and swap rates on all in-the-money paths are collected for the least-squares regression. The regression is performed by Intel's MKL function `LAPACKE_dgels`. Other MKL functions are used for linear algebra operations, such as vector-vector multiplication. These MKL functions by default support threading. During execution they are allowed to use any processor available in the system.

To minimise execution time of the approximation the necessary preprocessing that can be parallelised through threading is separated from the backward computation where the collection of the cross-sectional information makes explicit parallelisation difficult. The preprocessing includes random number generation, path construction and payoff calculation. These tasks are parallelised and evenly distributed onto all processors.

A path of forward rates conceptually is a two-dimensional structure. To save storage space in memory we map it onto an one-dimensional array. Fig. 2 illustrates the

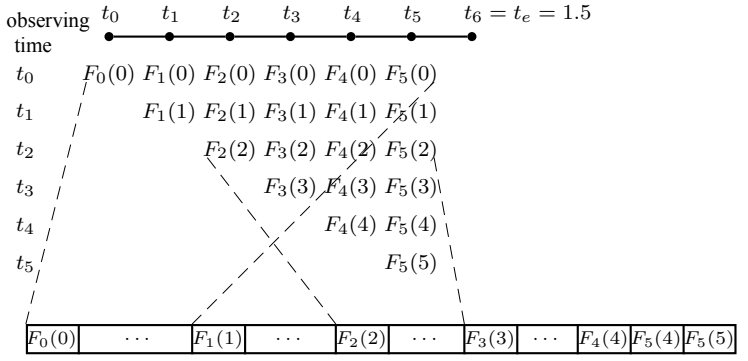


Fig. 2: A single forward rate path and its one-dimensional storage in memory.

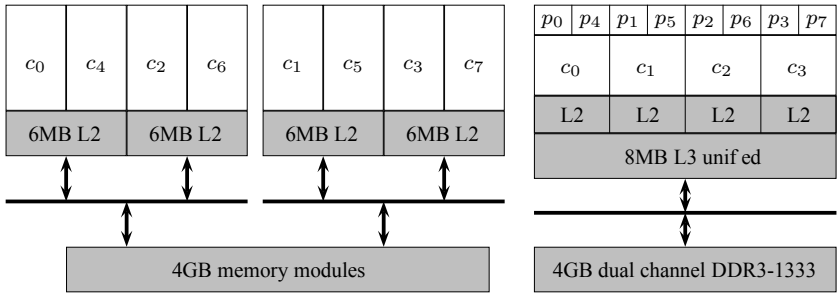
mapping using an example where $t_e = 1.5$ and $\delta = 0.25$. A mapping function $I(i,j) = \sum_{k=0}^{i-1}(t_e/\delta - k) + (j - i)$ is designed to return the index position in the array where rate $F_j(i)$ should be stored. For this function to work we need the obvious constraints $i \leq j$ and $i, j \in \{t_0, t_1, \ldots, t_x = t_{e-1}\}$.

When generating a forward rate using Equation (3), in computing the drift function $\hat{\boldsymbol{u}}$, repetitive evaluation of common sub-expressions should be avoided. For example, in generating $\hat{F}_3(3)$ drift function $\hat{\boldsymbol{u}}_3(2)$ needs to be evaluated according to Equation (4). Let $D_j(i)$ denote the term inside the summation in Equation (4), that is, $D_j(i) = \boldsymbol{\lambda}_j(i)(\delta_j\varphi(\hat{F}_j(i)))/(1 + \delta_j\hat{F}_j(i))$. Using this notation we have $\hat{\boldsymbol{u}}_3(2) = D_3(3)$. Now, when $\hat{F}_4(3)$ is computed from $\hat{F}_4(2)$, drift function $\hat{\boldsymbol{u}}_4(2) = D_3(3)+D_4(3)$ needs to be evaluated. However, by this time $D_3(3)$ has already been computed in the generation of $\hat{F}_3(3)$, and, so, its value should not be evaluated again. In our implementation, whenever $\hat{F}_{i+1}(i+1)$ is computed we have a buffer initialised for drift function $\hat{\boldsymbol{u}}_{i+1}(i) = D_{i+1}(i+1)$. Next, when generating $\hat{F}_j(i+1)$ for $t_j \in \{t_{i+2}, t_{i+3}, \ldots, t_x = t_{e-1}\}$ only $D_j(i+1)$ is computed and its value is added to the accumulated value in the buffer to form the value for drift function $\hat{\boldsymbol{u}}_j(i)$. This optimisation signif cantly reduced execution times of the path generations, especially for the nested simulations in computing the upper bound penalty term Δ_0. The routine in our implementation for computing the payoff $S(j)$ def ned in Equation (7) not only returns the payoff but also the swap rate $R(j,j)$ to save the rate being computed separately.

In computing the penalty term Δ_0 the inner simulation generates forward rates based on rates in existing path. For example, if on one of the N_H paths at exercise time t_i exercise is suggested, the inner simulation will be launched, which will then generate N_S paths to estimate the discounted continuation value. These N_S paths all originate from the time instance t_i on the original path, and so they all have the same forward rates observed between time t_0 and time t_i along the original path.

5 Experimental result

The parallel implementation was programed in C/C++. For comparison purpose an ef- f cient sequential program was also implemented. The NPTL (native POSIX thread



(a) Dual-socket E5405s (b) Core i7-2600 (256KB L2)

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

library) 2.12.1 was used for threading. Tests were made on a quad-core 3.4GHz Intel Core i7-2600 processor (Fig. 3(b)) and two quad-core 2.0GHz Intel Xeon E5405 processors (Fig. 3(a)). Both the systems ran Ubuntu Linux 10.10 64-bit version. The binary executables were compiled by Intel's C/C++ compiler icpc 12.0 for Linux with -O3 optimisation.

The implementations were tested using Bermudan swaptions under different lockouts t_s, final swap maturities t_e and fixed coupon rates θ. We used the same examples as in [3]. In all the tests, forward rates were generated in an extended two-factor LIBOR market model according to Equation (3). The parameters were set to the same values as in [3]: $\varphi(x) = x$, $\delta = 0.25$, $F_j(0) = 0.1$ for $j \in \{t_0, t_1, \ldots, t_{M-1}\}$, and $\boldsymbol{\lambda}_j(i) = (0.15, 0.15 - \sqrt{0.009(t_j - t_i)})$ for $i \leq j$. For the simulations we had $N_R = 5000$, $N_L = 50000$, $N_H = 750$ and $N_S = 300$. The testing results on the Intel Xeon E5405s are reported in Table 1. The runtimes were measured in seconds as wall clock times. Data from Table 4 in [3] are included for comparison. In all the tests we assumed $t_x = t_{e-1}$.

From the speedup (S_P) data it can be seen that the parallel program demonstrated significant accelerations against the sequential program. Comparing the data in the last two columns we can see that the figures are quite close, although not identical. While exercise strategies were generated by an optimisation procedure in [3], we generated them using the regression-based method. In computing $\hat{L}_0$ the work in [3] used antithetic sampling, but we did not use any variance reduction technique.

On the quad-core 3.4GHz Intel Core i7-2600 the same tests were made. For the 11-year contract with 0.08 fixed coupon rate the parallel program finished the whole computation in 11.63 seconds and demonstrated 5.2 times speedup against the sequential program.

6 Conclusions

We have presented a parallel algorithm that computes lower and upper bounds for prices of Bermudan swaptions under the extended multi-factor LIBOR market model. The algorithm uses the Longstaff-Schwartz least-squares Monte Carlo method in generating early exercise boundary for a swaption. This boundary is later used for estimating the lower and upper bounds. In computing the upper bound we follow the duality-based

Table 1: Lower and upper bounds of Bermudan swaptions and timing results on two quad-core 2.0GHz Intel Xeon E5405s.

t_s	t_e	θ	$\hat{L_0}$	T_L	$\hat{\Delta_0}$	T_U	T_P	S_P	95% CI	95% CI'
0.25	1.25	0.08	183.6	0.12	0.022	0.23	0.35	2.14	[183.6, 183.6]	[183.9, 184.1]
0.25	1.25	0.10	42.3	0.05	0.031	0.12	0.17	2.71	[42.3, 42.3]	[43.1, 43.6]
0.25	1.25	0.12	5.2	0.05	0.010	0.11	0.16	2.0	[5.2, 5.2]	[5.5, 5.7]
1.00	3.00	0.08	341.5	0.13	0.094	0.54	0.67	4.13	[341.5, 341.6]	[339.2, 340.6]
1.00	3.00	0.10	126.1	0.09	0.214	0.35	0.44	4.07	[126.1, 126.3]	[125.1, 127.2]
1.00	3.00	0.12	36.8	0.14	0.217	0.27	0.41	2.76	[36.8, 37.0]	[36.4, 37.6]
1.00	6.00	0.08	751.0	0.33	0.966	3.24	3.57	6.64	[751.0, 752.0]	[749.0, 755.2]
1.00	6.00	0.10	315.9	0.36	3.152	1.66	2.02	5.41	[315.8, 319.0]	[315.6, 323.5]
1.00	6.00	0.12	130.8	0.35	1.957	0.89	1.23	4.45	[130.8, 132.8]	[126.5, 131.6]
1.00	11.00	0.08	1236.5	0.95	11.310	19.02	19.98	7.14	[1236.5, 1247.9]	[1245.1, 1269.0]
1.00	11.00	0.10	613.3	1.11	17.619	9.62	10.73	6.29	[613.3, 631.0]	[618.4, 645.0]
1.00	11.00	0.12	334.2	1.05	13.022	4.20	5.26	6.70	[334.2, 347.3]	[324.7, 345.0]
3.00	6.00	0.08	458.0	0.29	0.218	1.23	1.52	5.17	[458.0, 458.3]	[443.6, 446.6]
3.00	6.00	0.10	234.4	0.34	0.403	0.86	1.20	4.53	[234.4, 234.8]	[225.5, 229.5]
3.00	6.00	0.12	110.8	0.32	0.776	0.65	0.97	3.90	[110.8, 111.6]	[105.9, 109.0]

Notes: The estimations $\hat{L_0}$ and $\hat{\Delta_0}$ and the conf dence intervals are reported in basis points. T_L is the parallel runtime in computing $\hat{L_0}$, T_U is the parallel runtime in computing $\hat{\Delta_0}$, and $T_P = T_L + T_U$ is the total parallel runtime. S_P is the speedup of the parallel implementation in computing $\hat{L_0}$ and $\hat{\Delta_0}$ against the sequential implementation running under the same settings. Data in the last column are copied from the seventh column of Table 4 in [3]. They are included for purpose of comparison.

procedure proposed by Andersen and Broadie. Sub-optimality checking is incorporated into the upper bound estimation to reduce its computational cost.

The implementation of the parallel algorithm was tested for its correctness and performance. The implementation works on shared-memory x86 multi-processor systems. Parallelisation was achieved through explicit POSIX threading, except in the generation of exercise boundary, in which Intel MKL functions' threading ability is exploited. Source code optimisations, such as elimination of common sub-expressions, were applied to the programs to speedup the simulations. Conceptually two-dimensional forward rate paths are mapped onto one-dimensional arrays in memory to save storage space. The implementation was tested on Intel multi-core systems. All the tests were completed by the parallel program in reasonable length of time periods. Signif cant speedups were observed against an eff cient sequential implementation.

References

1. Andersen, L.: A Simple Approach to the Pricing of Bermudan Swaptions in the Multi-Factor Libor Market Model. The Journal of Computational Finance 3(2) (1999)

2. Andersen, L., Andreasen, J.: Volatility Skews and Extensions of the Libor Market Model. Applied Mathematical Finance 7, 1–32 (2000)
3. Andersen, L., Broadie, M.: Primal-Dual Simulation Algorithm for Pricing Multidimensional American Options. Management Science 50(9), 1222–1234 (Sep 2004)
4. Brace, A., Gatarek, D., Musiela, M.: The Market Model of Interest Rate Dynamics. Mathematical Finance 7(2), 127–155 (Apr 1997)
5. Broadie, M., Cao, M.: Improved Lower and Upper Bound Algorithms for Pricing American Options by Simulation. Quantitative Finance 8(8), 845–861 (Dec 2008)
6. Hippler, S.: Pricing Bermudan Swaptions in the LIBOR Market Model. Master's thesis, University of Oxford (Jun 2008)
7. 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/`
8. 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/`
9. Jamshidian, F.: LIBOR and Swap Market Models and Measures. Finance and Stochastics 1(4), 293–330 (1997)
10. 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)
11. Miltersen, K.R., Sandmann, K., Sondermann, D.: Closed Form Solutions for Term Structure Derivatives with Log-Normal Interest Rates. The Journal of Finance 52, 409–430 (1997)