Chapter 26
Modern Monte Carlo Methods and GPU Computing

Álvaro Leitao and Cornelis W. Oosterlee

Abstract Pricing early-exercise options under multi-dimensional stochastic processes is a major challenge in the financial sector. In Leitao and Oosterlee (Int J Comput Math 92(12):2433–2454, 2015), a parallel GPU version of the Monte Carlo based Stochastic Grid Bundling Method (SGBM) (Jain and Oosterlee, Appl Math Comput 269:412–431, 2015) for pricing multi-dimensional Bermudan options is presented. The method is based on a combination of simulation, dynamic programming, regression and bundling of Monte Carlo paths. To extend the method's applicability, the problem dimensionality and the number of bundles is increased drastically. This makes SGBM very expensive in terms of computational costs on conventional hardware systems. A parallelization strategy of the method is developed and the GPGPU paradigm is used to reduce the execution time. An improved technique for bundling asset paths, which is more efficient on parallel hardware, is introduced. Thanks to the performance of the GPU version of SGBM, we can fully exploit the method and deal with very high-dimensional problems. Pricing results and comparisons between sequential and GPU parallel versions are presented.

26.1 Introduction

Monte Carlo methods are intensively employed in the financial field. Their simplicity in both interpretation and implementation makes them very attractive for practitioners. However, the main drawback usually attributed to this technique is the high computational cost. Although this fact has been improved in the last decades (due to the rapid evolution of the software and hardware), there still are some particular problems where the application of Monte Carlo methods can

Á. Leitao (✉) • C.W. Oosterlee
TU Delft, Delft Institute of Applied Mathematics, Delft, The Netherlands
CWI—Centrum Wiskunde & Informatica, Amsterdam, The Netherlands
e-mail: A.LeitaoRodriguez@tudelft.nl; c.w.oosterlee@cwi.nl

M. Ehrhardt et al. (eds.), *Novel Methods in Computational Finance*, Mathematics in Industry 25, DOI 10.1007/978-3-319-61282-9_26

be considered "expensive". That is the case when we deal with *multi-* or *high-*dimensional problems. In many cases, the use of Monte Carlo is the only option for this type of problems since other methodologies like PDE- or Fourier-based approaches can not be applied for very high dimensions (more than three), due to the curse of dimensionality. In this chapter, we will focus our analysis on *high-dimensional early-exercise option contracts* and how to combine a Monte Carlo based pricing method with parallel GPU computing resulting in an efficient technology.

In recent years, different Monte Carlo simulation techniques for pricing multi-dimensional early-exercise options were developed. Some representative methods were developed by Longstaff and Schwartz [7] and Tsitsiklis and Van Roy [8]. One of the recent Monte Carlo pricing techniques is the Stochastic Grid Bundling Method (SGBM), proposed by Jain and Oosterlee in [5] for pricing Bermudan options with several underlying assets. The method is a hybrid of *regression-* and *bundling-* based approaches, and uses regressed value functions, together with bundling of the state space to approximate continuation values at different time steps. In [6], the method's applicability has been extended by increasing the number of bundles and the problem dimensionality, which, together, also imply a drastic increase of the number of Monte Carlo paths. As the method becomes much more time-consuming then, we have parallelized the SGBM method taking advantage of the General-Purpose computing on Graphics Processing Units (GPGPU) paradigm. In this chapter, this work is summarized.

26.2 Problem Formulation

This section defines the Bermudan option pricing problem and sets up the notations used in this chapter. A Bermudan option is an option where the buyer has the right to exercise at a set number of times, $t \in [t_0 = 0, \ldots, t_m, \ldots, t_M = T]$, before the end of the contract, T. $\mathbf{S}_t = (S_t^1, \ldots, S_t^d) \in \mathbb{R}^d$ defines the d-dimensional underlying process. Let $h_t := h(\mathbf{S}_t)$ be an adapted process representing the intrinsic value of the option, i.e. the holder of the option receives $\max(h_t, 0)$, if the option is exercised at time t. The value of the option at the terminal time T is equal to the option's payoff, i.e.,

$$V_T(\mathbf{S}_T) = \max(h(\mathbf{S}_T), 0).$$

The conditional continuation value Q_{t_m}, i.e. the expected payoff at time t_m, is

$$Q_{t_m}(\mathbf{S}_{t_m}) = D_{t_m}\mathbb{E}\left[V_{t_{m+1}}(\mathbf{S}_{t_{m+1}})|\mathbf{S}_{t_m}\right],$$

se when we deal with *multi-* or *high-*
use of Monte Carlo is the only option
iodologies like PDE- or Fourier-based
gh dimensions (more than three), due
iapter, we will focus our analysis on
ntracts and how to combine a Monte
PU computing resulting in an efficient

imulation techniques for pricing multi-
veloped. Some representative methods
z [7] and Tsitsiklis and Van Roy [8].
niques is the Stochastic Grid Bundling
Oosterlee in [5] for pricing Bermudan
method is a hybrid of *regression-* and
ressed value functions, together with
e continuation values at different time
been extended by increasing the number
, which, together, also imply a drastic
ns. As the method becomes much more
the SGBM method taking advantage of
s Processing Units (GPGPU) paradigm.

ricing problem and sets up the notations
an option where the buyer has the right
$_0 = 0, \ldots, t_m, \ldots, t_M = T]$, before the
$\mathbb{R}^d$ defines the d-dimensional underlying
ess representing the intrinsic value of the
$\max(h_t, 0)$, if the option is exercised at
al time T is equal to the option's payoff,

$(h(\mathbf{S}_T), 0)$.

i.e. the expected payoff at time t_m, is

$_{t_{m+1}}(\mathbf{S}_{t_{m+1}})|\mathbf{S}_{t_m}\big]$,

where D_{t_m} is the discount factor. The Bermudan option value at time t_m and state $\mathbf{S}_{t_m}$ is then given by

$$V_{t_m}(\mathbf{S}_{t_m}) = \max(h(\mathbf{S}_{t_m}), Q_{t_m}(\mathbf{S}_{t_m})).$$

We are interested in finding the option value at the initial state $\mathbf{S}_{t_0}$, i.e. $V_{t_0}(\mathbf{S}_{t_0})$.

26.3 Stochastic Grid Bundling Method

SGBM [5] is a simulation-based Monte Carlo method for pricing early-exercise options (such as Bermudan options). SGBM first generates Monte Carlo paths forward in time, which is followed by determining the optimal early-exercise policy, moving backwards in time in a dynamic programming framework, based on the Bellman principle of optimality. The steps involved in the SGBM algorithm are briefly described in the following:

- **Step I: Generation of stochastic grid points.** The grid points in SGBM are generated by Monte Carlo sampling, i.e., by simulating independent copies of sample paths, $\{\mathbf{S}_{t_0}(n), \ldots, \mathbf{S}_{t_M}(n)\}$, $n = 1, \ldots, N$, of the underlying process $\mathbf{S}_t$, all starting from the same initial state $\mathbf{S}_{t_0}$.
- **Step II: Option value at terminal time.** The option value at the terminal time $t_M = T$ is given by

$$V_{t_M}(\mathbf{S}_{t_M}) = \max(h(\mathbf{S}_{t_M}), 0),$$

with $\max(h(\mathbf{S}_{t_M}), 0)$ a multi-dimensional payoff function.

The following steps are subsequently performed for each time step, t_m, $m \leq M$, recursively, moving backwards in time, starting from t_M.

- **Step III: Bundling.** The grid points at t_{m-1} are clustered or *bundled* into $\mathscr{B}_{t_{m-1}}(1), \ldots, \mathscr{B}_{t_{m-1}}(\nu)$ non-overlapping sets or partitions. SGBM employs bundling to approximate the conditional distribution using simulation. It samples this distribution by bundling the grid points at t_{m-1} and then uses those paths that originate from the corresponding bundle to obtain a conditional sample for time t_m.
- **Step IV: Mapping high-dimensional state space to a low-dimensional space.** Corresponding to each bundle $\mathscr{B}_{t_{m-1}}(\beta)$, $\beta = 1, \ldots, \nu$, a parameterized value function $Z : \mathbb{R}^d \times \mathbb{R}^K \mapsto \mathbb{R}$, which assigns values $Z(\mathbf{S}_{t_m}, \alpha^{\beta}_{t_m})$ to states $\mathbf{S}_{t_m}$, is computed. Here $\alpha^{\beta}_{t_m} \in \mathbb{R}^K$ is a vector of free parameters. The objective is then to choose, for each t_m and β, a parameter vector $\alpha^{\beta}_{t_m}$ so that

$$Z(\mathbf{S}_{t_m}, \alpha^{\beta}_{t_m}) \approx V_{t_m}(\mathbf{S}_{t_m}).$$

- **Step V: Computing the continuation and option values at t_{m-1}.** The continuation values for $\mathbf{S}_{t_{m-1}}(n) \in \mathscr{B}_{t_{m-1}}(\beta)$, $n = 1, \ldots, N$, $\beta = 1, \ldots, \nu$, are approximated by

$$\widehat{Q}_{t_{m-1}}(\mathbf{S}_{t_{m-1}}(n)) = \mathbb{E}[Z(\mathbf{S}_{t_m}, \alpha_{t_m}^{\beta}) | \mathbf{S}_{t_{m-1}}(n)]. \tag{26.1}$$

The option value is then given by

$$\widehat{V}_{t_{m-1}}(\mathbf{S}_{t_{m-1}}(n)) = \max(h(\mathbf{S}_{t_{m-1}}(n)), \widehat{Q}_{t_{m-1}}(\mathbf{S}_{t_{m-1}}(n))).$$

Due to their importance, the last three steps are more extensively described in the following sections.

26.3.1 Bundling

We propose a bundling technique in the SGBM context which is highly efficient when taking into account our goal of high dimensionality, called *equal-partitioning*. This technique is particularly well-suited for parallel processing: it does not involve an iterative process, distributes the data equally and does not need to store the d-dimensional points. Equal-partitioning has two steps: sorting and splitting. The general idea is to sort the data first under some convenient criterion and then split the sorted data items into sets (i.e. bundles) of equal size. The sorting process is independent of the dimension of the problem, efficient and, furthermore, it is highly parallelizable. In addition, the storage of all Monte Carlo simulation data points can be avoided since only a reduced part is needed in the bundling stage. The split stage assigns directly the portions of data to bundles which will contain the same number of *similar* (following some criterion) data items. Hence, the regression can be performed accurately even though the number of bundles increases in a significant way. Furthermore, the equally sized bundles allow for a better load balancing within the parallel implementation.

26.3.2 Parameterizing the Option Values

The high-dimensional option pricing problem become intractable and requires the approximation of the value function. This can be achieved by introducing a *parameterized value function* $Z : \mathbb{R}^d \times \mathbb{R}^L \mapsto \mathbb{R}$, which assigns a value $Z(\mathbf{S}_{t_m}, \alpha)$ to state $\mathbf{S}_{t_m}$, where $\alpha \in \mathbb{R}^L$ is a vector of free parameters. The goal is to choose, corresponding to each bundle β at time point t_{m-1}, a parameter vector $\alpha_{t_m}^{\beta} := \alpha$ so that,

$$V_{t_m}(\mathbf{S}_{t_m}) \approx Z\left(\mathbf{S}_{t_m}, \alpha_{t_m}^{\beta}\right).$$

ɔn and option values at t_{m-1}. The con-
$_1(\beta)$, $n = 1,\ldots,N$, $\beta = 1,\ldots,\nu$, are

$$\mathbb{E}[Z(\mathbf{S}_{t_m}, \alpha_{t_m}^{\beta})|\mathbf{S}_{t_{m-1}}(n)]. \tag{26.1}$$

$i(\mathbf{S}_{t_{m-1}}(n)), \widehat{Q}_{t_{m-1}}(\mathbf{S}_{t_{m-1}}(n)))$.

steps are more extensively described in the

ɜ SGBM context which is highly efficient
ɲ dimensionality, called *equal-partitioning*,
l for parallel processing: it does not involve
a equally and does not need to store the
g has two steps: sorting and splitting. The
ɜr some convenient criterion and then split
dles) of equal size. The sorting process is
ɔlem, efficient and, furthermore, it is highly
all Monte Carlo simulation data points can
eeded in the bundling stage. The split stage
ɯndles which will contain the same number
data items. Hence, the regression can be
umber of bundles increases in a significant
dles allow for a better load balancing within

on Values

problem become intractable and requires
ɔn. This can be achieved by introducing a
$\mathbb{R}^L \mapsto \mathbb{R}$, which assigns a value $Z(\mathbf{S}_{t_m}, \alpha)$
of free parameters. The goal is to choose,
point t_{m-1}, a parameter vector $\alpha_{t_m}^{\beta} := \alpha$ so

$Z\left(\mathbf{S}_{t_m}, \alpha_{t_m}^{\beta}\right)$.

SGBM follows the approach of Tsitsiklis and Van Roy [8] and it uses basis functions to approximate the values of the options. Hence, two important decisions have to be made: the form of the function Z and the basis functions. For each particular problem we define several basis functions, $\phi_1, \phi_2, \ldots, \phi_L$, that are typically chosen based on experience, as in the case of the LSM method [7], aiming to represent relevant properties of a given state, $\mathbf{S}_{t_m}$. In our case, the form of $Z\left(\mathbf{S}_{t_m}, \alpha_{t_m}^{\beta}\right)$ depends on $\mathbf{S}_{t_m}$ only through $\phi_k(\mathbf{S}_{t_m})$. Hence, for some function $f : \mathbb{R}^L \times \mathbb{R}^L \mapsto \mathbb{R}$, we can write $Z\left(\mathbf{S}_{t_m}, \alpha_{t_m}^{\beta}\right) = f\left(\phi_k(\mathbf{S}_{t_m}), \alpha_{t_m}^{\beta}\right)$, where

$$Z\left(\mathbf{S}_{t_m}, \alpha_{t_m}^{\beta}\right) = \sum_{k=1}^{L} \alpha_{t_m}^{\beta}(k)\phi_k(\mathbf{S}_{t_m}).$$

An exact computation of the vector of free parameters, $\alpha_{t_m}^{\beta}$, is generally not feasible for the corresponding bundle $\mathscr{B}_{t_{m-1}}(\beta)$. Thus, an approximation $\widehat{\alpha}_{t_m}^{\beta}$ is computed by using ordinary least squares regression.

By using the parameterized option value function $Z\left(\mathbf{S}_{t_m}, \widehat{\alpha}_{t_m}^{\beta}\right)$ for the bundle $\mathscr{B}_{t_{m-1}}(\beta)$, the continuation values in Eq. (26.1) are approximated by

$$\begin{aligned}\widehat{Q}_{t_{m-1}}(\mathbf{S}_{t_{m-1}}(n)) &= D_{t_{m-1}}\mathbb{E}\left[\left(\sum_{k=1}^{L}\widehat{\alpha}_{t_m}^{\beta}(k)\phi_k(\mathbf{S}_{t_m})\right)|\mathbf{S}_{t_{m-1}} = \mathbf{S}_{t_{m-1}}(n)\right] \\ &= D_{t_{m-1}}\sum_{k=1}^{L}\widehat{\alpha}_{t_m}^{\beta}(k)\mathbb{E}\left[\phi_k(\mathbf{S}_{t_m})|\mathbf{S}_{t_{m-1}} = \mathbf{S}_{t_{m-1}}(n)\right],\end{aligned} \tag{26.2}$$

where $\mathbf{S}_{t_{m-1}}(n) \in \mathscr{B}_{t_{m-1}}(\beta)$. The continuation value will give us a reference value to compute the early-exercise policy. The basis functions ϕ_k should be chosen such that the expectations $\mathbb{E}\left[\phi_k(\mathbf{S}_{t_m})|\mathbf{S}_{t_{m-1}} = \mathbf{S}_{t_{m-1}}(n)\right]$ in Eq. (26.2) are easy to calculate, i.e. they are preferably known in closed form or otherwise have analytic approximations. In [6], several choices to determine the basis functions are described, either for particular cases or more general situations.

26.3.3 *Estimating the Option Value*

The estimation of the option value is the final step in SGBM. We consider the so-called *direct estimator* and *path estimator*. The direct estimator is typically biased high, i.e., it is often an upper bound. The definition of the direct estimator is

$$\widehat{V}_{t_{m-1}}(\mathbf{S}_{t_{m-1}}(n)) = \max\left(h\left(\mathbf{S}_{t_{m-1}}(n)\right), \widehat{Q}_{t_{m-1}}\left(\mathbf{S}_{t_{m-1}}(n)\right)\right),$$

where $n = 1, \ldots, N$. The final option value reads

$$\mathbb{E}[\widehat{V}_{t_0}(\mathbf{S}_{t_0})] = \frac{1}{N}\sum_{n=1}^{N} \widehat{V}_{t_0}(\mathbf{S}_{t_0}(n)).$$

The direct estimator corresponds to Step V in the initial description.

Once the optimal early-exercise policy has been obtained, the path estimator, which is typically biased low, can be developed based on the early-exercise policy. The resulting confidence interval is useful, because, depending on the problem at hand, sometimes the path estimator and sometimes the direct estimator is superior. The obtained confidence intervals are generally small, indicating accurate results. In order to compute the low-biased estimates, we generate a new set of paths, as in common for duality-based Monte Carlo methods, $\mathbf{S}(n) = \{\mathbf{S}_{t_1}(n), \ldots, \mathbf{S}_{t_M}(n)\}$, $n = 1, \ldots, N_{\mathrm{L}}$. Along each path, the approximate optimal policy exercises are

$$\widehat{\tau}^*(\mathbf{S}(n)) = \min\{t_m : h(\mathbf{S}_{t_m}(n)) \geq \widehat{Q}_{t_m}(\mathbf{S}_{t_m}(n)), m = 1, \ldots, M\},$$

where $\widehat{Q}_{t_m}(\mathbf{S}_{t_m}(n))$ is previously computed using Eq. (26.2). The path estimator is then defined by $v(n) = h\left(\mathbf{S}_{\widehat{\tau}^*(\mathbf{S}(n))}\right)$. Finally, the low-biased estimate given by the path estimator is

$$\underline{V}_{t_0}(\mathbf{S}_{t_0}) = \lim_{N_{\mathrm{L}}} \frac{1}{N_{\mathrm{L}}}\sum_{n=1}^{N_{\mathrm{L}}} v(n).$$

26.4 Parallel SGBM Method: Implementation Details

The GPU parallel implementation was performed by employing the *Compute Unified Device Architecture*, CUDA, a parallel computing platform and programming model developed by NVIDIA (see [2]).

Since SGBM is based on two clearly separated stages, we parallelize them separately. First of all, the Monte Carlo path generation is parallelized (Step I). As is well-known, Monte Carlo methods are very suitable for parallelization, because of characteristics like a very large number of simulations and data independence. In Fig. 26.1a, we see schematically how the parallelization is done where $p_0, p_1, \ldots, p_{N-1}$ are the CUDA *threads*. The second main stage of SGBM is the regression and the computation of the continuation and option values (Steps IV and V) in each bundle, backwards in time. Due to the data dependency between time steps, the way to parallelize this stage of the method is by parallelizing over the bundles, performing the calculations in each bundle in parallel. Schematic and simplified representation with two bundles is given in Fig. 26.1b. Note that, actually,

where $n = 1, \ldots, N$. The final option value reads

$$\mathbb{E}[\widehat{V}_{t_0}(\mathbf{S}_{t_0})] = \frac{1}{N}\sum_{n=1}^{N} \widehat{V}_{t_0}(\mathbf{S}_{t_0}(n)).$$

direct estimator corresponds to Step V in the initial description. 2)
nce the optimal early-exercise policy has been obtained, the
h is typically biased low, can be developed based on the early-e
esulting confidence interval is useful, because, depending on t(3)
ometimes the path estimator and sometimes the direct estimat
tained confidence intervals are generally small, indicating acc
to compute the low-biased estimates, we generate a new se
ion for duality-based Monte Carlo methods, $\mathbf{S}(n) = \{\mathbf{S}_{t_l}(n),$
$\ldots, N_L$. Along each path, the approximate optimal policy exerc— 1)
$t(N)$

$^*(\mathbf{S}(n)) = \min\{t_m : h(\mathbf{S}_{t_m}(n)) \geq \widehat{Q}_{t_m}(\mathbf{S}_{t_m}(n)), m = 1, \ldots,$
$\mathbf{S}_{t_m}(n))$ is previously computed using Eq. (26.2). The path
by $v(n) = h\left(\mathbf{S}_{\hat{\tau}^*(\mathbf{S}(n))}\right)$. Finally, the low-biased estimate
r is

$$\underline{V}_{t_0}(\mathbf{S}_{t_0}) = \lim_{N_L} \frac{1}{N_L}\sum_{n=1}^{N_L} v(n).$$

SGBM Method: Implementation Details

nplementation was performed by employing the *Compu*
ture, CUDA, a parallel computing platform and progra
NVIDIA (see [2]).
ased on two clearly separated stages, we parallelize
, the Monte Carlo path generation is parallelized (S
onte Carlo methods are very suitable for parallel
tics like a very large number of simulations and
6.1a, we see schematically how the parallelization is
re the CUDA *threads*. The second main stage of S
computation of the continuation and option values (
backwards in time. Due to the data dependency bet
llelize this stage of the method is by parallelizing
calculations in each bundle in parallel. Schematic
h two bundles is given in Fig. 26.1b. Note that, actu

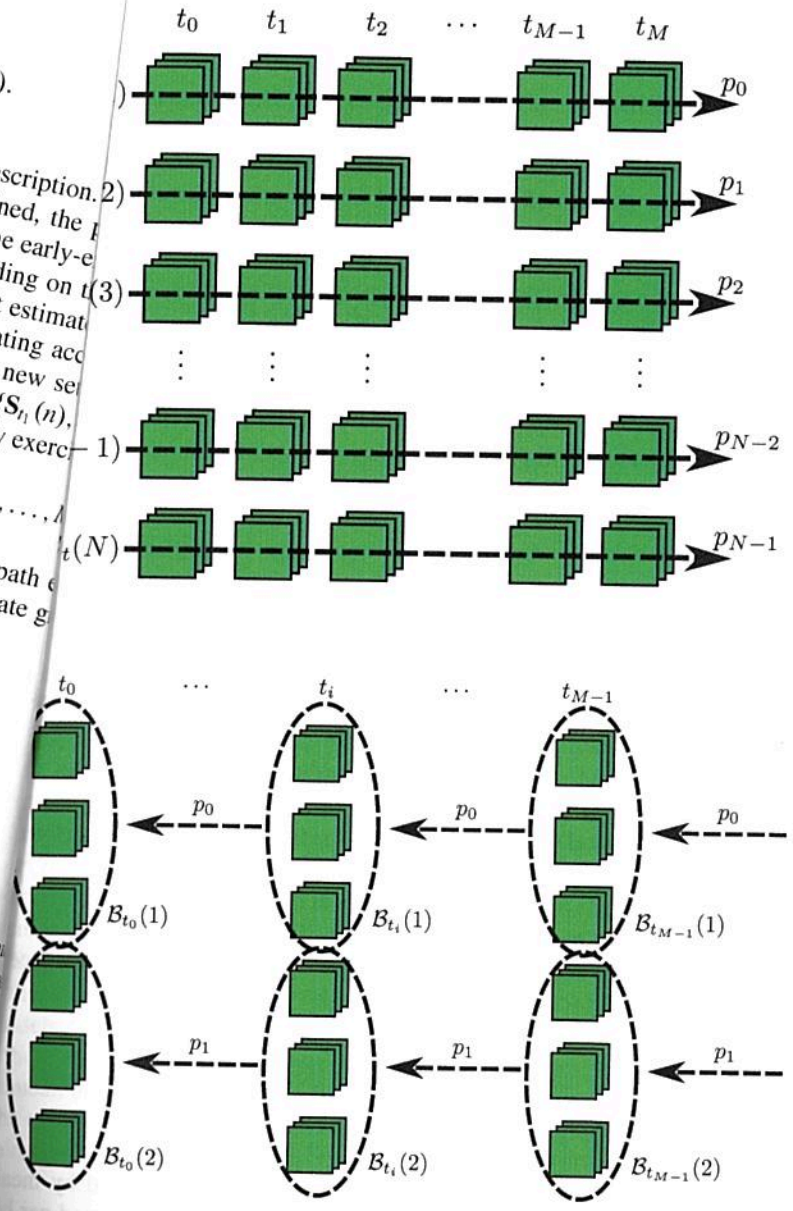


Fig. 26.1 Stages of the parallel SGBM method. (**a**) Monte Carlo stage. (**b**) Bundling stage

several stages of parallelization are performed, one per time step. Between each parallel stage, the bundling (Step III) is carried out.

In the following subsections, we will show more specific details of the CUDA implementation of the parallel SGBM method.

26.4.1 Parallel Monte Carlo paths

We launch one CUDA thread per Monte Carlo path. The necessary random numbers are obtained "on the fly" in each thread by means of the cuRAND standard library. In addition, the intrinsic value of the option and also the computation of the expectation in Eq. (26.2) are performed within the Monte Carlo generator, decreasing the number of launched loops and taking advantage of the parallel execution. The intermediate results are stored in an array defined inside the CUDA kernel which can be allocated in the registers, speeding up the memory accesses. We also perform calculations for the sorting criterion (required in the equal partitioning bundling) inside the Monte Carlo generator, avoiding the storage of the complete Monte Carlo grid points and the transfers of data from GPU global memory to CPU main memory in each time step. This approach gives us a considerable performance and allows us to increase drastically the dimensionality and also the number of Monte Carlo simulations (depending on the number of bundles).

26.4.2 Bundling Scheme

As mentioned, equal-partitioning bundling involves two operations: sorting and splitting. For the sort part, we take advantage of the CUDA Data-Parallel Primitives Library (CUDPP), described in [3]. We choose the parallel Radix sort which is included in version 2.1 of CUDPP. In addition, CUDPP provides a kernel-level Application Programming Interface (API), allowing us to avoid the transfers between stages of parallelization. Once the sorting stage has been performed, the splitting stage is immediate since the size of the bundles is known, i.e. N/ν. Each CUDA thread manages a pointer which points at the beginning of the corresponding region of GPU global memory for each bundle. The memory allocation is made for all bundles together which means that the bundle's memory areas are adjacent and the accesses are faster (*coalescing*).

26.4.3 Estimators

The exercise policy and the final option values can be computed by means of direct and path estimators. For the direct estimator, one CUDA thread per bundle is launched at each time step. For each bundle, the regression and option values are calculated on the GPU. All CUDA threads collaborate in order to compute the continuation value which determines the early-exercise policy. Once the early-exercise policy is determined, the path estimator can be executed. Its parallelization can be done over paths because the early-exercise policy is already known (given by the previous computation of the direct estimator) and is not needed to perform

is

arlo path. The necessary random numbers
means of the cuRAND standard library. In
nd also the computation of the expectation
Monte Carlo generator, decreasing the
dvantage of the parallel execution. The
y defined inside the CUDA kernel which
up the memory accesses. We also perform
uired in the equal partitioning bundling)
g the storage of the complete Monte Carlo
GPU global memory to CPU main memory
is a considerable performance and allows
ility and also the number of Monte Carlo
bundles).

ing involves two operations: sorting and
tage of the CUDA Data-Parallel Primitives
Ve choose the parallel Radix sort which
In addition, CUDPP provides a kernel-
e (API), allowing us to avoid the transfers
the sorting stage has been performed, the
ze of the bundles is known, i.e. N/v. Each
oints at the beginning of the corresponding
bundle. The memory allocation is made for
ne bundle's memory areas are adjacent and

on values can be computed by means of
ct estimator, one CUDA thread per bundle
h bundle, the regression and option values
A threads collaborate in order to compute
s the early-exercise policy. Once the early-
stimator can be executed. Its parallelization
rly-exercise policy is already known (given
ect estimator) and is not needed to perform

the regression. One CUDA thread per path is launched and it computes the optimal exercise time and the cash flows according to the policy.

26.5 Results

The implementation have been carried out in two programming languages: C and CUDA. This allows us to assess the improvement given by the parallel version compared with the sequential one. Experiments were performed on the Accelerator Island system of the Cartesius Supercomputer (more information in [1]). We consider the d-dimensional problem of pricing basket Bermudan options under the multi-dimensional Geometric Brownian Motion (GBM). The experiments setting is

- Initial state: $\mathbf{S}_{t_0} = (40, 40, \ldots, 40) \in \mathbb{R}^d$.
- Strike: $K = 40$.
- Risk-free interest rate: $r_t = 0.06$.
- Dividend yield rate: $q_\delta = 0.0, \delta = 1, 2, \ldots, d$.
- Volatility: $\sigma_\delta = 0.2, \delta = 1, 2, \ldots, d$.
- Correlation: $\rho_{i,j} = 0.25, j = 2, \ldots, d, i = 1, \ldots, j$.
- Maturity: $T = 1.0$.
- Exercise times: $M = 10$.
- Number of basis functions: $L = 3$.

26.5.1 *Equal-Partitioning: Convergence Test*

First, we wish to test the convergence of the new equal-partitioning technique bundling. Regarding the sorting criterion, we choose the payoff criterion, i.e. we sort the Monte Carlo scenarios following the geometric or arithmetic average of the assets for geometric and arithmetic basket options, respectively. In Fig. 26.2, we show the convergence in option prices for geometric and arithmetic basket Bermudan options with different dimensionalities, i.e. $d = 5$, $d = 10$ and $d = 15$. In the case of the geometric basket option, we can also specify the reference price obtained by the COS method [4].

26.5.2 *Parallel SGBM: Performance Test*

With the convergence of the equal-partitioning technique shown numerically, we now increase drastically the number of bundles and, hence, the number of Monte Carlo paths, and perform a computational cost experiment, comparing the C and

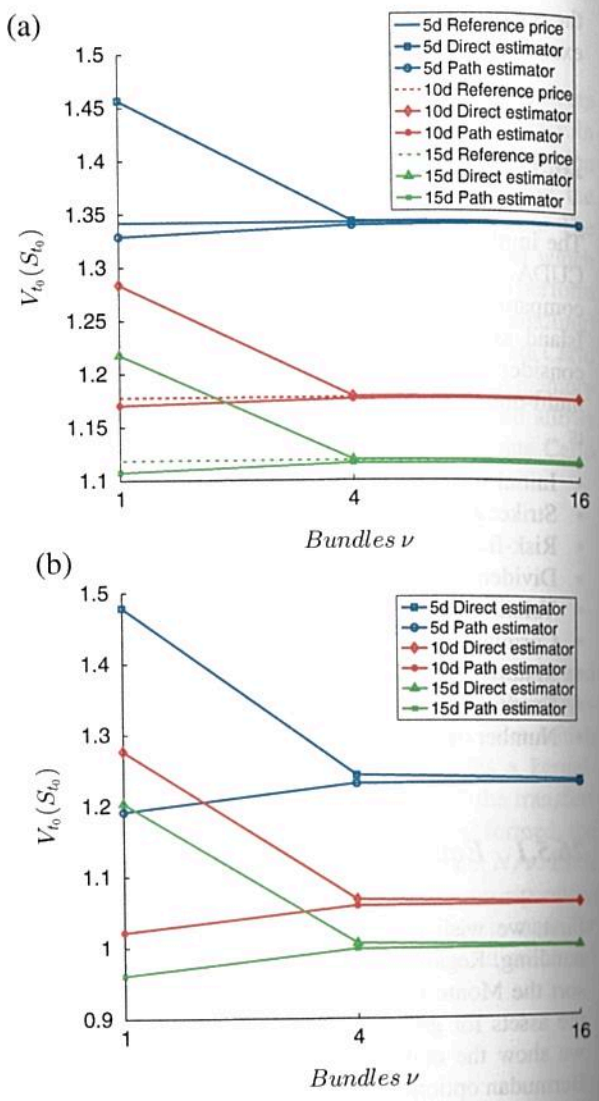
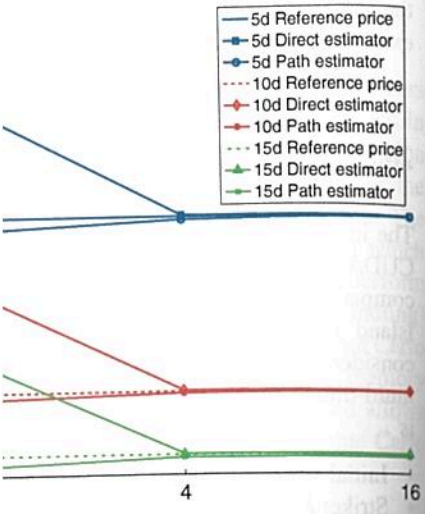
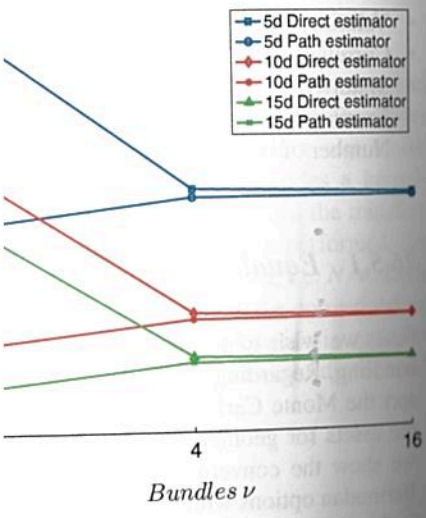


Fig. 26.2 Convergence with equal-partitioning technique. Configuration: $N = 2^{18}$ and $\Delta t = T/M$. (**a**) Geometric basket put option. (**b**) Arithmetic basket put option

CUDA implementations. We now focus on pricing arithmetic basket Bermudan option since it is a more interesting and involved problem. In the first half of the Table 26.1, the execution times for the different SGBM stages, i.e. Monte Carlo path generation (MC), direct estimator computation (DE) and path estimator computation (PE) are shown. The total computational cost for $d = 5$, $d = 10$ and $d = 15$ problems are presented in the second half of the Table 26.1. In both experiments, we observe a significant acceleration of the CUDA versions. The impact of the dimensionality is much less important in the parallel version than in the sequential version, resulting in an increasing speedup in terms of the problem dimension.





on pricing arithmetic basket Bermudan
involved problem. In the first half of the
ferent SGBM stages, i.e. Monte Carlo path
ation (DE) and path estimator computation
l cost for $d = 5$, $d = 10$ and $d = 15$
lf of the Table 26.1. In both experiments,
the CUDA versions. The impact of the
the parallel version than in the sequential
dup in terms of the problem dimension.

Table 26.1 Time (s) for the C and CUDA versions

	Stages time ($d = 5$)			Total time		
	MC	DE	PE	$d = 5$	$d = 10$	$d = 15$
C	79.22	39.64	58.65	256.05	600.09	1143.06
CUDA	0.83	4.14	1.20	8.02	11.23	15.73
Speedup	95.44	9.57	48.87	31.93	53.44	72.67

Configuration: $N = 2^{22}$, $\Delta t = T/M$ and $\nu = 2^{10}$

Table 26.2 Time (s) for a high-dimensional problem

	$\nu = 2^{10}$			$\nu = 2^{14}$		
	$d = 30$	$d = 40$	$d = 50$	$d = 30$	$d = 40$	$d = 50$
C	993.96	1723.79	2631.95	992.29	1724.60	2631.43
CUDA	11.14	17.88	26.99	11.20	17.94	27.07
Speedup	89.22	96.41	97.51	88.60	96.13	97.21

Configuration: $N = 2^{20}$ and $\Delta t = T/M$

This is thank to equal-partitioning bundling technique, since the parallelism can be efficiently exploited.

26.5.3 *High-Dimensional Problems*

We now test a high-dimensional option pricing problem. In Table 26.2, the execution times for pricing arithmetic basket Bermudan put options in different dimensions and with different numbers of bundles, ν, are presented. Note that the number of bundles hardly influences the execution times and the performance is mainly dependent on the number of paths and the dimensionality. The obtained speedup reaches around 100 times for the 50-dimensional problem.

26.6 Conclusions

In this chapter, we have presented an efficient implementation of the Stochastic Grid Bundling Method on a GPU architecture. Through the GPU parallelism, we could speed up the execution times when the number of bundles and the dimensionality increase drastically. In addition, we have proposed a parallel bundling technique which is more efficient in terms of memory use and more suitable on parallel systems. These two improvements enable us to extend the method's applicability to high-dimensional problems.

Compared with other GPU parallel implementations of early-exercise option pricing methods, our parallel SGBM is very competitive in terms of computational time since we provided a new way to parallelize the backward stage, according to the bundles, which gave us a remarkable performance improvement.

Acknowledgements The first author is supported by the European Union in the FP7-PEOPLE-2012-ITN Program under Grant Agreement Number 304617 (FP7 Marie Curie Action, Project Multi-ITN STRIKE—Novel Methods in Computational Finance). The authors would like to thank Shashi Jain, ING Bank, for providing support and the original codes of the Stochastic Grid Bundling Method.

References

1. Cartesius webpage: https://www.surfsara.nl/systems/cartesius
2. CUDA webpage: http://www.nvidia.com/object/cuda_home_new.html
3. CUDPP webpage: http://cudpp.github.io/
4. Fang, F., Oosterlee, C.W.: Pricing early-exercise and discrete Barrier options by Fourier-cosine series expansions. Numer. Math. **114**(1), 27–62 (2009)
5. Jain, S., Oosterlee, C.W.: The stochastic grid bundling method: efficient pricing of Bermudan options and their Greeks. Appl. Math. Comput. **269**, 412–431 (2015)
6. Leitao, A., Oosterlee, C.W.: GPU acceleration of the stochastic grid bundling method for early-exercise options. Int. J. Comput. Math. **92**(12), 2433–2454 (2015)
7. Longstaff, F.A., Schwartz, E.S.: Valuing American options by simulation: a simple least-squares approach. Rev. Financ. Stud. **14**(1), 113–47 (2001)
8. Tsitsiklis, J.N., Van Roy, B.: Regression methods for pricing complex American-style options. IEEE Trans. Neural Netw. **12**(4), 694–703 (2001)