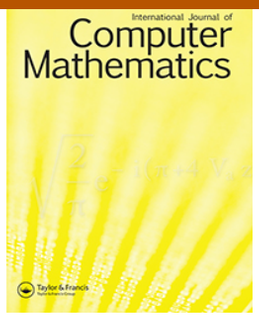

International Journal of Computer Mathematics

ISSN: 0020-7160 (Print) 1029-0265 (Online) Journal homepage: http://www.tandfonline.com/loi/gcom20

GPU acceleration of the stochastic grid bundling method for early-exercise options

Álvaro Leitao & Cornelis W. Oosterlee

To cite this article: Álvaro Leitao & Cornelis W. Oosterlee (2015) GPU acceleration of the stochastic grid bundling method for early-exercise options, International Journal of Computer Mathematics, 92:12, 2433-2454, DOI: 10.1080/00207160.2015.1067689

To link to this article: http://dx.doi.org/10.1080/00207160.2015.1067689

Accepted author version posted online: 29 Jun 2015.
Published online: 02 Sep 2015.

International Journal of Computer Mathematics, 2015
Vol. 92, No. 12, 2433–2454, http://dx.doi.org/10.1080/00207160.2015.1067689

REVIEW

GPU acceleration of the stochastic grid bundling method for early-exercise options

Álvaro Leitao[a,b*] and Cornelis W. Oosterlee[a,b]

[a]*Delft Institute of Applied Mathematics (DIAM), TU Delft, Delft, The Netherlands;* [b]*Centrum Wiskunde & Informatica (CWI), Amsterdam, The Netherlands*

(*Received 30 October 2014; revised version received 4 June 2015; accepted 4 June 2015*)

In this work, a parallel graphics processing units (GPU) version of the Monte Carlo stochastic grid bundling method (SGBM) for pricing multi-dimensional early-exercise options is presented. To extend the method's applicability, the problem dimensions and the number of bundles will be increased drastically. This makes SGBM very expensive in terms of computational costs on conventional hardware systems based on central processing units. A parallelization strategy of the method is developed and the general purpose computing on graphics processing units 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, a general approach for computing the early-exercise policy is proposed. Comparisons between sequential and GPU parallel versions are presented.

Keywords: computational finance; early-exercise derivatives; basket Bermudan options; high-dimensional pricing; stochastic grid bundling method (SGBM); Monte Carlosimulation; least-squares regression; high performance computing; parallel programming; GPGPU; compute unified device architecture (CUDA)

2010 AMS Subject Classifications: 91G60; 91G70; 65Y05; 65C05; 65C10

1. Introduction

In recent years, different techniques for pricing *early-exercise option contracts*, as they appear in computational finance, were developed. In the wake of the recent financial crisis, accurately modelling and pricing these kinds of options gained additional importance, as they also form the basis for incorporation of the so-called *counterparty risk premium* to an option value, which can be seen as an option covering the fact that a counterparty may go into default before the end of a financial contract, and thus cannot pay possible contractual obligations.

The early-exercise pricing methods can be classified into different kinds of techniques depending on whether they are based on *simulation and regression*, *transition probabilities* or *duality* approaches. Focusing on the first class of methods, Monte Carlo (MC) path generation and dynamic programming to determine an optimal early-exercise policy and the option price are typically combined. Some representative methods were developed by Longstaff and Schwartz [23] and Tsitsiklis and Van Roy [33].

*Corresponding author. Email: a.leitaorodriguez@tudelft.nl

The pricing problem becomes significantly more complex when the early-exercise options are *multi-* or even *high*-dimensional. One of the recent simulation–regression pricing techniques for early-exercise (Bermudan) options with several underlying assets, on which we will focus, is the *stochastic grid bundling method* (SGBM), proposed by Jain and Oosterlee [17]. The method is a hybrid of *regression* and *bundling* approaches, as it employs regressed value functions, together with bundling of the state space to approximate continuation values at different time steps. A high biased *direct estimator* (DE) and an early-exercise policy are first computed in SGBM. The early-exercise policy is then used to determine a lower bound to the true option price which is called the *path estimator* (PE). The early-exercise policy computation involves the computation of expectations of certain *basis functions*. Usually, these basis functions are chosen by experience in such a way that an analytic solution for the expectations appearing in the pricing method is available.

In this paper, we extend SGBM's applicability by drastically increasing the problem dimensionality, the number of bundles and, hence, the number of MC asset paths. As the method becomes much more time-consuming then, we parallelize SGBM taking advantage of the *general purpose computing on graphics processing units* (GPGPU) paradigm. It is known that GPGPU is very well suitable for financial purposes and MC techniques. In the case of pricing early-exercise options, several contributions regarding graphics processing units (GPU) implementations of different techniques appeared recently [1,2,11,12,15,26]. All these papers are based on a combination of a MC method and dynamic programming, except the Dang et al. paper, which uses partial differential equation (PDE) based pricing methods and the Pagès and Wilbertz paper, which employs MC simulation together with a quantization method. Our GPU version of SGBM is based on parallelization in two steps, according to the method's stages, i.e. forward in time (MC path generator) followed by the backward stage (early-exercise policy computation). This is a novelty with respect to other methods since, although the parallelization of a MC method is immediate, the parallelization of the backward stage is not trivial for other methods, for example not for the least-squares MC method (LSM) by Longstaff and Schwartz [23] where some load balancing issues for parallel systems can appear as only the in-the-money paths are considered. Due to some timing and distribution issues observed in the original bundling procedure caused by an increasing number of bundles, we present another bundling technique for SGBM which is much more efficient and more suitable on parallel hardware.

Thanks to the performance of our SGBM GPU version, we can explore different possibilities to compute the expectations appearing within the SGBM formulation, generalize towards different option contracts and, in particular, more general underlying asset models. A general approach to compute the expectations needed for the early-exercise policy is proposed and evaluated. This approach is based on the computation of a characteristic function for the discretized underlying stochastic differential system. Once this *discrete* characteristic function is determined, we can use it for the expectations calculation, for example, for multi-dimensional local volatility asset models.

The paper is organized as follows. In Section 2, the Bermudan option pricing problem is introduced. Section 3 describes the SGBM. The new approach to compute the expectations for the early-exercise policy is explained in Section 4. Section 5 gives details about the GPU implementation. In Section 6, numerical results and central processing unit (CPU) /GPU time comparisons are shown. Finally, we conclude in Section 7.

2. Bermudan options

This section defines the Bermudan option pricing problem and sets up the notations used. A Bermudan option is an option where the buyer has the right to exercise at a number of

times, $t \in [t_0 = 0, \ldots, t_m, \ldots, t_M = T]$, before the end of the contract, T. $\boldsymbol{S}_t = (S_t^1, \ldots, S_t^d) \in \mathbb{R}^d$ defines a d-dimensional underlying process which here follows the dynamics given by the system of stochastic differential equations (SDEs)

$$\begin{aligned} \mathrm{d}S_t^1 &= \mu_1(\boldsymbol{S}_t)\,\mathrm{d}t + \sigma_1(\boldsymbol{S}_t)\,\mathrm{d}W_t^1, \\ \mathrm{d}S_t^2 &= \mu_2(\boldsymbol{S}_t)\,\mathrm{d}t + \sigma_2(\boldsymbol{S}_t)\,\mathrm{d}W_t^2, \\ &\vdots \\ \mathrm{d}S_t^d &= \mu_d(\boldsymbol{S}_t)\,\mathrm{d}t + \sigma_d(\boldsymbol{S}_t)\,\mathrm{d}W_t^d, \end{aligned} \tag{1}$$

where $W_t^\delta, \delta = 1, 2, \ldots, d$, are correlated standard Brownian motions. The instantaneous correlation coefficient between W_t^i and W_t^j is $\rho_{i,j}$. The correlation matrix is thus defined as

$$\mathbf{C} = \begin{pmatrix} 1 & \rho_{1,2} & \cdots & \rho_{1,d} \\ \rho_{1,2} & 1 & \cdots & \rho_{2,d} \\ \vdots & \vdots & \ddots & \vdots \\ \rho_{1,d} & \rho_{2,d} & \cdots & 1 \end{pmatrix},$$

and the Cholesky decomposition of $\mathbf{C}$ reads

$$\mathbf{L} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ \rho_{1,2} & L_{2,2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \rho_{1,d} & L_{2,d} & \cdots & L_{d,d} \end{pmatrix}. \tag{2}$$

Let $h_t := h(\boldsymbol{S}_t)$ 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. With the risk-free savings account process $B_t = \exp(\int_0^t r_s\,\mathrm{d}s)$, where r_t denotes the instantaneous risk-free rate of return, we define

$$D_{t_m} = \frac{B_{t_m}}{B_{t_{m+1}}}.$$

We consider the special case where r_t is constant. The problem is then to compute

$$V_{t_0}(\boldsymbol{S}_{t_0}) = \max_\tau \mathbb{E}\left[\frac{h(\boldsymbol{S}_\tau)}{B_\tau}\right],$$

where τ is a stopping time, taking values in the finite set $\{0, t_1, \ldots, T\}$. The value of the option at the terminal time T is equal to the option's payoff

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

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

$$Q_{t_m}(\boldsymbol{S}_{t_m}) = D_{t_m}\mathbb{E}[V_{t_{m+1}}(\boldsymbol{S}_{t_{m+1}})|\boldsymbol{S}_{t_m}]. \tag{3}$$

The Bermudan option value at time t_m and state $\boldsymbol{S}_{t_m}$ is then given by

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

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

3. Stochastic grid bundling method

The SGBM [17] is a simulation-based MC method for pricing early-exercise options (such as Bermudan options). SGBM first generates MC paths forward in time, 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 paragraphs:

Step I: Generation of stochastic grid points

The grid points in SGBM are generated by MC sampling, i.e. by simulating independent copies of sample paths, $\{\boldsymbol{S}_{t_0}(n), \ldots, \boldsymbol{S}_{t_M}(n)\}$, $n = 1, \ldots, N$, of the underlying process $\boldsymbol{S}_t$, all starting from the same initial state $\boldsymbol{S}_{t_0}$. The nth grid point at exercise time step t_m is then denoted by $\boldsymbol{S}_{t_m}(n)$, $n = 1, \ldots, N$. Depending upon the underlying stochastic process an appropriate discretization scheme (Δt, the discretization step), e.g. the Euler scheme, is used to generate sample paths. Sometimes the diffusion process can be simulated directly, essentially because it appears in closed form, like for the regular multi-dimensional Black–Scholes model.

Step II: Option value at terminal time

The option value at the terminal time $t_M = T$ is given by

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

with $\max(h(\boldsymbol{S}_{t_M}), 0)$ the payoff function.

This relation is used to compute the option value for all grid points at the final time step.

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 $\mathcal{B}_{t_{m-1}}(1), \ldots, \mathcal{B}_{t_{m-1}}(\nu)$ non-overlapping sets or partitions. SGBM employs bundling to approximate the conditional distribution using simulation. The method 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, see also in [17]. Different approaches for partitioning can be considered. Due to its importance in the parallel case, this decision is discussed in more detail in Section 3.1.

Step IV: Mapping high-dimensional state space to a low-dimensional space

Corresponding to each bundle $\mathcal{B}_{t_{m-1}}(\beta)$, $\beta = 1, \ldots, \nu$, a parameterized value function $Z : \mathbb{R}^d \times \mathbb{R}^K \mapsto \mathbb{R}$, which assigns values $Z(\boldsymbol{S}_{t_m}, \alpha_{t_m}^{\beta})$ to states $\boldsymbol{S}_{t_m}$, is computed. Here $\alpha_{t_m}^{\beta} \in \mathbb{R}^K$ is a vector of free parameters. The objective is then to choose, for each t_m and β, a parameter vector $\alpha_{t_m}^{\beta}$ so that

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

After some approximations, $Z(\boldsymbol{S}_{t_m}, \alpha_{t_m}^{\beta})$ can be computed by using ordinary least-squares regression.

Step V: Computing the continuation and option values at t_{m-1}

The continuation values for $S_{t_{m-1}}(n) \in \mathcal{B}_{t_{m-1}}(\beta)$, $n = 1, \ldots, N$, $\beta = 1, \ldots, \nu$, are approximated by

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

The option value is then given by

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

3.1 *Bundling*

One of the techniques proposed (in [17]) to partition the data into ν non-overlapping sets is the *k-means* clustering technique. The algorithm uses an iterative refinement algorithm, where, given an initial guess of cluster means, first of all the algorithm assigns each data item to one specific set (i.e. bundle) by calculating the distance between the item and the cluster mean (under some measure) and subsequently updates the sets and the cluster means. This process is repeated until some stopping condition is satisfied. The procedure needs a pre-bundling step to set approximated initial cluster means. The bundles obtained by using the k-means technique can be very irregular, i.e. the number of data items within each bundle can vary a lot. This fact can be a problem when many bundles are considered, and parallel load balancing can be an issue too.

Since our goal is to drastically increase the number of bundles used and, in particular, the problem dimensionality, the k-means algorithm becomes too expensive in terms of computational time and memory usage in high dimensions because the iterative process of searching new sets (of high dimension) takes much time and d-dimensional data points for each MC path and for each time step have to be stored. In addition, it may happen that some bundles do not contain a sufficient number of data points to compute an accurate regression function when the number of bundles increases. In order to overcome these two problems of the k-means clustering, we propose another bundling technique in the SGBM context which is much more efficient when taking into account our goal of efficiency in high dimensions: it does not involve an iterative process, distributes the data equally and does not need to store the d-dimensional points. The details are given in the next section.

3.1.1 *Equal-partitioning technique*

The equal-partitioning bundling technique is particularly well-suited for parallel processing; it involves 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. A schematic representation of this technique is shown in Figure 1.

With this simple approach, the drawbacks of the iterative bundling in the case of very high dimensions and an enormous number of bundles can be avoided. The sorting process is independent of the dimension of the problem, more efficient and less time-consuming than an iterative

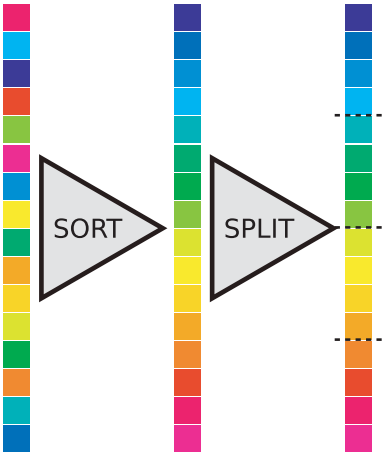

Figure 1. Equal-partitioning scheme.

search and, furthermore, it is highly parallelizable. In addition, the storage of all MC simulation data points can be avoided since only a reduced part is needed in the bundling stage with equal-partitioning. This is possible because the criterion is known in advance and its computation inside the MC generator allows to reduce a dimension of the stored data, i.e. storing a 2D matrix ($N \times M$) instead of storing a 3D matrix ($N \times M \times d$). 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.

3.2 *Parameterizing the option values*

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

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

As in the original SGBM paper, we follow the approach of Tsitsiklis and Van Roy [33] and we use 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, $\phi_1, \phi_2, \ldots, \phi_K$, that are typically chosen based on experience, as in the case of the LSM method [23], aiming to represent relevant properties of a given state, $\boldsymbol{S}_{t_m}$. In Section 3.2.2 and Section 4, we present two possible choices, one specific and the other one more generally applicable. In our case, the form of $Z(\boldsymbol{S}_{t_m}, \alpha_{t_m}^{\beta})$ depends on $\boldsymbol{S}_{t_m}$ only through $\phi_k(\boldsymbol{S}_{t_m})$. Hence, for some function $f : \mathbb{R}^K \times \mathbb{R}^K \mapsto \mathbb{R}$, we can write $Z(\boldsymbol{S}_{t_m}, \alpha_{t_m}^{\beta}) = f(\phi_k(\boldsymbol{S}_{t_m}), \alpha_{t_m}^{\beta})$, where

$$Z(\boldsymbol{S}_{t_m}, \alpha_{t_m}^{\beta}) = \sum_{k=1}^{K} \alpha_{t_m}^{\beta}(k)\phi_k(\boldsymbol{S}_{t_m}). \tag{4}$$

An exact computation of the vector of free parameters, $\alpha_{t_m}^{\beta}$, is generally not feasible. Hence, Equation (4) can be approximated by

$$Z(\boldsymbol{S}_{t_m}, \hat{\alpha}_{t_m}^{\beta}) = \sum_{k=1}^{K} \hat{\alpha}_{t_m}^{\beta}(k)\phi_k(\boldsymbol{S}_{t_m}), \tag{5}$$

satisfying

$$\arg\min_{\hat{\alpha}_{t_m}^{\beta}} \sum_{n=1}^{|\mathcal{B}_{t_{m-1}}(\beta)|} \left(V_{t_m}(\boldsymbol{S}_{t_m}(n)) - \sum_{k=1}^{K} \hat{\alpha}_{t_m}^{\beta}(k)\phi_k(\boldsymbol{S}_{t_m}(n)) \right)^2,$$

for the corresponding bundle $\mathcal{B}_{t_{m-1}}(\beta)$. The parameterized function in Equation (5) is computed by using ordinary least-squares regression, so that

$$V_{t_m}(\boldsymbol{S}_{t_m}(n)) = Z(\boldsymbol{S}_{t_m}(n), \hat{\alpha}_{t_m}^{\beta}) + \epsilon_{t_m}^{\beta},$$

where $\boldsymbol{S}_{t_{m-1}}(n) \in \mathcal{B}_{t_{m-1}}(\beta)$. The regression error, $\epsilon_{t_m}^{\beta}$, can be neglected here, as we have a sufficiently large number of paths in each bundle.

3.2.1 *Computing the continuation value*

Using the parameterized option value function $Z(\boldsymbol{S}_{t_m}, \hat{\alpha}^{\beta}_{t_m})$ corresponding to bundle $\mathcal{B}_{t_{m-1}}(\beta)$, the continuation values for the grid points that belong to this bundle are approximated by

$$\hat{Q}_{t_{m-1}}(\boldsymbol{S}_{t_{m-1}}(n)) = D_{t_{m-1}}\mathbb{E}[Z(\boldsymbol{S}_{t_m}, \hat{\alpha}^{\beta}_{t_m})|\boldsymbol{S}_{t_{m-1}} = \boldsymbol{S}_{t_{m-1}}(n)],$$

where $\boldsymbol{S}_{t_{m-1}}(n) \in \mathcal{B}_{t_{m-1}}(\beta)$. Using Equation (5), this can be written as

$$\begin{aligned}\hat{Q}_{t_{m-1}}(\boldsymbol{S}_{t_{m-1}}(n)) &= D_{t_{m-1}}\mathbb{E}\left[\left(\sum_{k=1}^{K}\hat{\alpha}^{\beta}_{t_m}(k)\phi_k(\boldsymbol{S}_{t_m})\right)\middle|\boldsymbol{S}_{t_{m-1}} = \boldsymbol{S}_{t_{m-1}}(n)\right] \\ &= D_{t_{m-1}}\sum_{k=1}^{K}\hat{\alpha}^{\beta}_{t_m}(k)\mathbb{E}[\phi_k(\boldsymbol{S}_{t_m})|\boldsymbol{S}_{t_{m-1}} = \boldsymbol{S}_{t_{m-1}}(n)].\end{aligned} \tag{6}$$

The continuation value will give us a reference value to compute the early-exercise policy and it will be used by the estimators in Section 3.3.

3.2.2 *Choice of basis functions*

The basis functions ϕ_k should be chosen such that the expectations $\mathbb{E}[\phi_k(\boldsymbol{S}_{t_m})|\boldsymbol{S}_{t_{m-1}} = \boldsymbol{S}_{t_{m-1}}(n)]$ in Equation (6) are easy to calculate, i.e. they are preferably known in closed form or otherwise have analytic approximations. The intrinsic value of the option, $h(\cdot)$, is usually an important and useful basis function, especially in high problem dimensions. In this section, we describe the choices of basis functions for different basket Bermudan options with the underlying processes following geometric Brownian motion ; we thus choose in Equation (1)

$$\mu_\delta(\boldsymbol{S}_t) = (r_t - q_\delta)S^\delta_t, \quad \sigma_\delta(\boldsymbol{S}_t) = \sigma_\delta S^\delta_t,$$

where r_t is the risk-free rate and q_δ and σ_δ, $\delta = 1, 2, \ldots, d$, are the dividend yield rates and the volatility, respectively.

In the case of a *geometric* basket Bermudan option, the intrinsic value of the option is defined by

$$h(\boldsymbol{S}_{t_m}) = \left(\prod_{\delta=1}^{d} S^\delta_{t_m}\right)^{1/d} - X, \quad h(\boldsymbol{S}_{t_m}) = X - \left(\prod_{\delta=1}^{d} S^\delta_{t_m}\right)^{1/d},$$

for *call* or *put* options, respectively, where X is the strike value of the option contract.

Then, basis functions that make sense are given by

$$\phi_k(\boldsymbol{S}_{t_m}) = \left(\left(\prod_{\delta=1}^{d} S^\delta_{t_m}\right)^{1/d}\right)^{k-1}, \quad k = 1, \ldots, K. \tag{7}$$

The expectation in Equation (6) requires the computation of the expectations of the expression in Equation (7), given $\boldsymbol{S}_{t_{m-1}}$,

$$\mathbb{E}[\phi_k(\boldsymbol{S}_{t_m})|\boldsymbol{S}_{t_{m-1}}(n)] = (P_{t_{m-1}}(n)\mathrm{e}^{(\bar{\mu}+(k-1)\bar{\sigma}^2/2)\Delta t})^{k-1},\ k = 1, \ldots, K, \tag{8}$$

where

$$P_{t_{m-1}}(n) = \left(\prod_{\delta=1}^{d} S^\delta_{t_{m-1}}(n)\right)^{1/d}, \quad \bar{\mu} = \frac{1}{d}\sum_{\delta=1}^{d}\left(r_t - q_\delta - \frac{\sigma^2_\delta}{2}\right), \quad \bar{\sigma}^2 = \frac{1}{d^2}\sum_{i=1}^{d}\left(\sum_{j=1}^{d} L^2_{ij}\right)^2,$$

with L_{ij}-matrix element i,j of $\mathbf{L}$, the Cholesky decomposition of the correlation matrix, given by Equation (2).

For the *arithmetic* basket Bermudan options, the intrinsic values of the call and put options are

$$h(\boldsymbol{S}_{t_m}) = \left(\frac{1}{d}\sum_{\delta=1}^{d} S_{t_m}^{\delta}\right) - X, \quad h(\boldsymbol{S}_{t_m}) = X - \left(\frac{1}{d}\sum_{\delta=1}^{d} S_{t_m}^{\delta}\right).$$

The basis functions are chosen as

$$\phi_k(\boldsymbol{S}_{t_m}) = \left(\frac{1}{d}\sum_{\delta=1}^{d} S_{t_m}^{\delta}\right)^{k-1}, \quad k = 1,\ldots,K.$$

Again, the continuation value, as given by Equation (6), requires us to compute

$$\mathbb{E}[\phi_k(\boldsymbol{S}_{t_m})|\boldsymbol{S}_{t_{m-1}}(n)] = \mathbb{E}\left[\left(\frac{1}{d}\sum_{\delta=1}^{d} S_{t_m}^{\delta}\right)^{k-1}\middle|\boldsymbol{S}_{t_{m-1}}(n)\right], \quad k = 1,\ldots,K. \tag{9}$$

The expectation in Equation (9) can be expressed as a linear combination of moments of the geometric average of the assets [17], i.e.

$$\left(\sum_{\delta=1}^{d} S_{t_m}^{\delta}\right)^{k} = \sum_{k_1+k_2+\cdots+k_d=k} \binom{k}{k_1,k_2,\ldots,k_d} \prod_{1\le\delta\le d} (S_{t_m}^{\delta})^{k_\delta},$$

where

$$\binom{k}{k_1,k_2,\ldots,k_d} = \frac{k!}{k_1!\,k_2!\cdots k_d!},$$

which can be computed in a straightforward way by Equation (8).

3.3 *Estimating the option value*

The estimation of the option value is the final step in SGBM. In this work, we follow the original proposed idea in [17] and we consider the so-called *DE* and *PE*, which can give us a confidence interval for the option price. SGBM has been developed as a so-called duality-based method, as originally introduced by Haugh and Kogan [16] and Rogers [27]. Using a duality-based methods an upper bound on the option value for a given exercise policy can be obtained, by adding a non-negative quantity that penalizes potentially incorrect exercise decisions made by the sub-optimal policy. The SGBM DE is typically biased high, i.e. it is often an upper bound estimator. The definition of the DE is

$$\hat{V}_{t_{m-1}}(\boldsymbol{S}_{t_{m-1}}(n)) = \max(h(\boldsymbol{S}_{t_{m-1}}(n)), \hat{Q}_{t_{m-1}}(\boldsymbol{S}_{t_{m-1}}(n))),$$

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

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

The DE corresponds to Step V in the initial description.

Once the optimal early-exercise policy has been obtained, the PE, 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 PE and sometimes the DE is superior. The obtained confidence intervals are generally small, indicating accurate SGBM results. In order to compute the low-biased estimates, we generate a new set of paths, as in common for duality-based MC methods, $\boldsymbol{S}(n) = \{\boldsymbol{S}_{t_1}(n), \ldots, \boldsymbol{S}_{t_M}(n)\}$, $n = 1, \ldots, N_{\mathrm{L}}$, using the same scheme as followed for generating the paths in the case of the DE and the bundling stage is performed considering the new set of paths. Along each path, the approximate optimal policy exercises at

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

where $\hat{Q}_{t_m}(\boldsymbol{S}_{t_m}(n))$ is previously computed using Equation (6). The PE is then defined by $v(n) = h(\boldsymbol{S}_{\hat{\tau}^*(\boldsymbol{S}(n))})$. Finally, the low-biased estimate given by the PE is

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

where $V_{t_0}(\boldsymbol{S}_{t_0})$ is the true option value.

To summarize and clarify the general idea behind SGBM, in Algorithm 1 we show the corresponding algorithm considering both, DE and PE.

Algorithm 1: SGBM

Data: $\boldsymbol{S}_{t_0}, X, \mu_\delta, \sigma_\delta, \rho_{i,j}, T, N, M$
Pre-Bundling (only in k-means case).
Generation of the grid points (Monte Carlo). Step I.
Option value at terminal time $t = M$. Step II.
for *Time* $t = (M-1) \cdots 1$ **do**
  Bundling. Step III.
  for *Bundle* $\beta = 1 \cdots \nu$ **do**
    Exercise policy (Regression). Step IV.
    Continuation value. Step V.
    Direct estimator. Step V.
Generation of the grid points (Monte Carlo). Step I.
Option value at terminal time $t = M$. Step II.
for *Time* $t = (M-1) \cdots 1$ **do**
  Bundling. Step III.
  for *Bundle* $\beta = 1 \cdots \nu$ **do**
    Continuation value. Step V.
    Path estimator. Step V.

4. Continuation value computation: new approach

In Section 3.2.2, we showed that, for certain derivative contracts (geometric and arithmetic basket Bermudan options) and underlying processes (GBM), analytic solutions were available for the expectations that we needed to compute within SGBM. However, finding suitable basis functions resulting in analytic expressions for the expectations is not always possible for all

underlying models. For that, in this work, we propose a different way to compute the expected value in Equation (6) which is more generally applicable, for example, we can also work with local volatility models with this approach. The approach consists of, first, discretizing the asset process and then obtaining the multi-variate characteristic function of the discrete process. This procedure is based on the work of Ruijter and Oosterlee [28]. Once we have this *discrete characteristic function*, we can compute the required expectations for certain choices of basis functions.

4.1 *Discretization, joint discrete characteristic function and joint moments*

Taking into account the correlation between Brownian motions and the Cholesky decomposition given by Equation (2) and using an Euler scheme with time step $\Delta t = t_{m+1} - t_m$, we can discretize the general system of SDEs defined by Equation (1), with independent Brownian motions, $\Delta \tilde{W}^{\delta}_{t_{m+1}} = \tilde{W}^{\delta}_{t_{m+1}} - \tilde{W}^{\delta}_{t_m}$, $\delta = 1, 2, \ldots, d$, as follows:

$$
\begin{aligned}
S^1_{t_{m+1}} &= S^1_{t_m} + \mu_1(\boldsymbol{S}_{t_m})\Delta t + \sigma_1(\boldsymbol{S}_{t_m})\Delta \tilde{W}^1_{t_{m+1}},\\
S^2_{t_{m+1}} &= S^2_{t_m} + \mu_2(\boldsymbol{S}_{t_m})\Delta t + \rho_{1,2}\sigma_2(\boldsymbol{S}_{t_m})\Delta \tilde{W}^1_{t_{m+1}} + L_{2,2}\sigma_2(\boldsymbol{S}_{t_m})\Delta \tilde{W}^2_{t_{m+1}},\\
&\vdots\\
S^d_{t_{m+1}} &= S^d_{t_m} + \mu_d(\boldsymbol{S}_{t_m})\Delta t + \rho_{1,d}\sigma_d(\boldsymbol{S}_{t_m})\Delta \tilde{W}^1_{t_{m+1}} + L_{2,d}\sigma_d(\boldsymbol{S}_{t_m})\Delta \tilde{W}^2_{t_{m+1}} + \cdots + L_{d,d}\sigma_d(\boldsymbol{S}_{t_m})\Delta \tilde{W}^d_{t_{m+1}},
\end{aligned}
$$

where $\boldsymbol{S}_{t_m} = (S^1_{t_m}, S^2_{t_m}, \ldots, S^d_{t_m})$.

The d-variate discrete characteristic function of process $\boldsymbol{S}_{t_{m+1}}$, given $\boldsymbol{S}_{t_m}$, is given by

$$
\begin{aligned}
&\psi_{\boldsymbol{S}_{t_{m+1}}}(u_1, u_2, \ldots, u_d | \boldsymbol{S}_{t_m})\\
&\quad= \mathbb{E}\left[\exp\left(\sum_{j=1}^{d} i u_j S^j_{t_{m+1}}\right) | \boldsymbol{S}_{t_m}\right]\\
&\quad= \mathbb{E}\left[\exp\left(\sum_{j=1}^{d} i u_j \left(S^j_{t_m} + \mu_j(\boldsymbol{S}_{t_m})\Delta t + \sigma_j(\boldsymbol{S}_{t_m})\sum_{k=1}^{j} L_{k,j}\Delta \tilde{W}^k_{t_{m+1}}\right)\right) | \boldsymbol{S}_{t_m}\right]\\
&\quad= \exp\left(\sum_{j=1}^{d} i u_j (S^j_{t_m} + \mu_j(\boldsymbol{S}_{t_m})\Delta t)\right) \cdot \prod_{k=1}^{d}\left(\mathbb{E}\left[\exp\left(\sum_{j=k}^{d} i u_j L_{k,j}\sigma_j(\boldsymbol{S}_{t_m})\Delta \tilde{W}^k_{t_{m+1}}\right)\right]\right)\\
&\quad= \exp\left(\sum_{j=1}^{d} i u_j (S^j_{t_m} + \mu_j(\boldsymbol{S}_{t_m})\Delta t)\right) \cdot \prod_{k=1}^{d}\left(\psi_{\mathcal{N}(0,\Delta t)}\left(\sum_{j=k}^{d} u_j L_{k,j}\sigma_j(\boldsymbol{S}_{t_m})\right)\right),
\end{aligned}
$$

where i is the imaginary unit and $\psi_{\mathcal{N}(\mu,\sigma^2)}(u) = \exp(i\mu u - 0.5\sigma^2 u^2)$ is the characteristic function of a normal random variable with mean μ and variance σ^2.

The joint moments of the product of several random variables can be obtained by evaluating the following expression (more details in [22,34]), based on the discrete characteristic function

$$
\begin{aligned}
M_{\boldsymbol{S}_{t_{m+1}}} &= \mathbb{E}[(S^1_{t_{m+1}})^{c_1}(S^2_{t_{m+1}})^{c_2}\cdots(S^d_{t_{m+1}})^{c_d} | \boldsymbol{S}_{t_m}]\\
&= (-i)^{c_1+c_2+\cdots+c_d}\left[\frac{\partial^{c_1+c_2+\cdots+c_d}\psi_{\boldsymbol{S}_{t_{m+1}}}(\boldsymbol{u}|\boldsymbol{S}_{t_m})}{\partial u_1^{c_1}\partial u_2^{c_2}\cdots\partial u_d^{c_d}}\right]_{\boldsymbol{u}=0},
\end{aligned}
\tag{10}
$$

being $\boldsymbol{u} = (u_1, u_2, \ldots, u_d)$.

If we take the basis functions, as used in Equation (5), to be a product of the underlying processes to some power, i.e.

$$\phi_k(\boldsymbol{S}_{t_m}) = \left(\prod_{\delta=1}^{d} S_{t_m}^{\delta}\right)^{k-1}, \quad k = 1, \ldots, K, \tag{11}$$

the expected value in Equation (6) can be easily computed by means of Equation (10).

The approximation obtained in this way is, in general, worse than the analytic value because the characteristic function on which the expectations are based is related to the Euler SDE discretization and thus less accurate. However, since we can increase the number of bundles and, in particular, time steps drastically (due to our GPU implementation), we can employ a suitable combination of these to price products without analytic solution for the characteristic function. More involved models for the underlying asset dynamics can be chosen, for which we do not have analytic expressions for these expectations. In Section 6.4, more details are given.

5. Implementation details

5.1 *GPGPU: compute unified device architecture*

GPGPU is the use of graphics hardware (GPUs) in order to perform computations which are typically performed by CPUs. In this sense, the GPU can be seen as a co-processor of the CPU.

The compute unified device architecture, CUDA, is a parallel computing platform and programming model developed by NVIDIA (see [8]) for its GPUs. CUDA eases the coding of algorithms by means of an extension of traditional programming languages, like C. In this section, we give some basic details about the platform which are necessary to understand the parallel version of SGBM. More information about CUDA can be obtained from [5,7,14,19,20,29,36].

5.1.1 *Programming model*

The basic operational units in CUDA are the *CUDA threads*. The CUDA threads are processes that are executed in parallel. The threads are grouped into *blocks* of the same size and blocks of threads are grouped into *grids*. Such organization eases the adaptability to different problems. One grid executes special functions called *kernels* so that all threads of the grid execute the same instructions in parallel.

5.1.2 *Memory hierarchy*

CUDA threads may access data from multiple memory spaces during their execution. Each thread manages its own *registers* and has private *local memory*. Each thread block has *shared memory* visible to all threads of the block and with the same lifetime as the block. All threads have access to the same *global memory*. There are also two additional read-only memory spaces accessible by all threads: the *constant and texture memory* spaces. In terms of speed, the registers represent the fastest memory (but also the smallest) while the global and local memories are the slowest. In recent GPU architectures (Kepler GK110 and GK110B, for example), several levels of cache are included to improve the accessibility.

5.1.3 *Memory transfers*

A key aspect in each parallel system is the memory transfer. Specifically in a GPU framework, the transfers between the CPU main memory and the GPU memory space are of great importance for the performance of the implementation. The number of transfers and the amount of data for each transfer are important. CUDA provides different ways to allocate and move data. In that sense, one of the CUDA 5.5 features is unified virtual addressing which allows asynchronous transfers and page-locked memory accesses by using a single address space for both CPU and GPU.

5.2 *Parallel SGBM*

The original SGBM implementation [17] was done in Matlab. The first step of our implementation was to code an efficient sequential version of the method in the C programming language, because it eases the parallel coding in CUDA. In addition, both implementations can be used to compare results and execution times. Once we obtained the C-version, we coded the CUDA version aiming to parallelize the suitable parts of the method. In addition, we also carried out the implementation of SGBM with the equal-partitioning bundling technique in C and CUDA.

Since SGBM is based on two clearly separated stages, we parallelize them separately. First of all, the MC path generation is parallelized (Step I). As is well known, MC methods are very suitable for parallelization, because of characteristics like a very large number of simulations and data independence. In Figure 2(a), 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 representations with two bundles are given in Figure 2(b) and 2(c) for the two considered bundling techniques, k-means and equal-partitioning. Note that, actually, several stages of parallelization are performed, one per time step. Between each parallel stage, the bundling (Step III) is carried out. This step can be also parallelized in the case of equal-partitioning. In the case of k-means, bundling has to be done sequentially.

As mentioned, the memory transfers between CPU and GPU are key since we have to move huge amounts of data. An increase of the number of bundles implies a drastic increase of the number of MC paths. Data has to be moved to CPU memory between the stages of parallelization, MC path simulation and bundle computations. For example, in case of the MC scenario generator, we need to store in and move from GPU global memory to CPU memory $N \times M \times d$ doubles[1] (8 bytes).

In the following subsections, we will show more specific details of the CUDA implementation of the two SGBM versions, the original one (with k-means bundling) and the new one (with equal-partitioning).

5.2.1 *Parallel MC paths*

In a GPU very large numbers of threads can be managed, so we launch one CUDA thread per MC path. The necessary random numbers are obtained 'on the fly' in each thread by means of the cuRAND library [10]. In addition, the intrinsic value of the option and also the computation of the expectation in Equation (6) are performed within the MC generator, decreasing the number of launched loops, i.e. we perform all necessary computations for the bundling and pricing stages taking advantage of the parallel MC simulation. The intermediate results are stored in an array defined inside the kernel which can be allocated in the registers, speeding up the memory accesses.

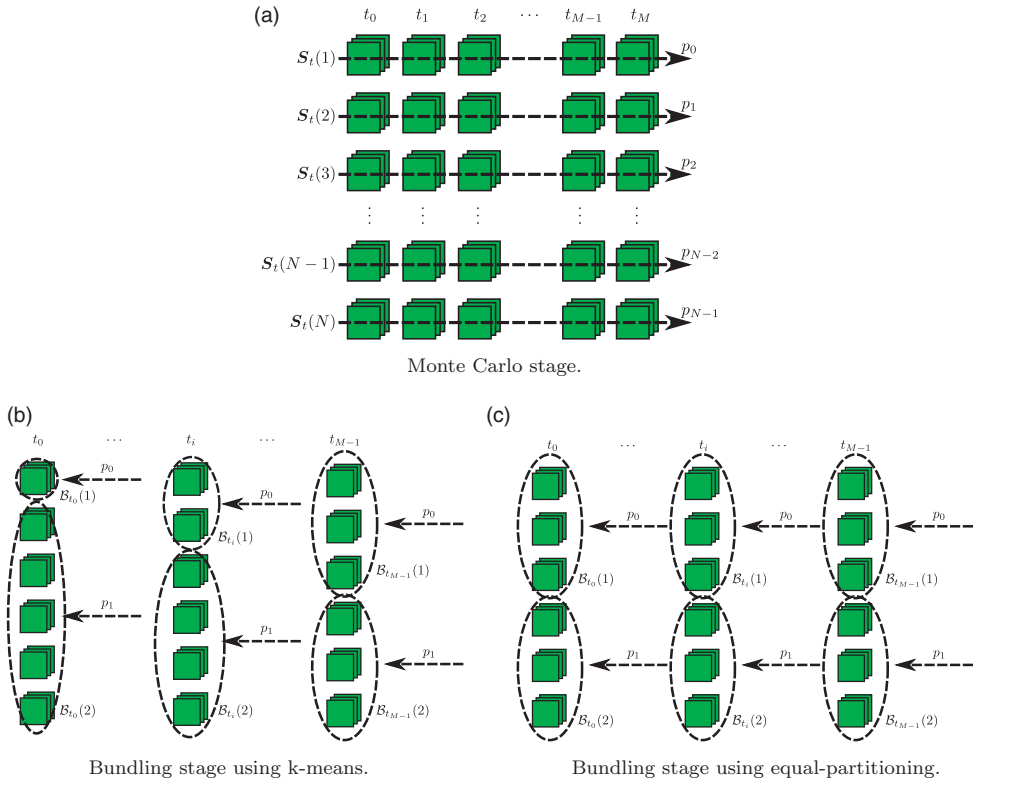

Figure 2. Parallel SGBM. MC and bundling stages: (a) MC stage. (b) Bundling stage using k-means. (c) Bundling stage using equal-partitioning.

In the original implementation (using k-means bundling), we need to store all MC data because it will be used during the bundling stage. This is a limiting factor since the maximum memory storage in global memory is easily reached. Furthermore, we have to move each value obtained from GPU global memory to CPU main memory and, with a significant number of paths, time steps and dimensions, this transfer can become really expensive.

When *equal-partitioning bundling* is considered, the main difference is that we now also perform calculations for the sorting criterion inside the MC generator, avoiding the storage of the complete MC simulation and the transfer of data from GPU global memory to CPU main memory. This approach gives us a considerable performance improvement and allows us to increase drastically the dimensionality and also the number of MC simulations (depending on the number of bundles).

5.2.2 *Bundling schemes*

For the k-means clustering the computations of the distances between the cluster means and all of the stochastic grid points have been parallelized. However, the other parts must be performed sequentially. The very large number of bundles makes this very expensive, since the bundling must be done in each time step.

As mentioned, equal-partitioning bundling involves two operations: sorting and splitting. It is well known that efficient sorting algorithms are a challenging research topic (see [21], for example). In parallel systems, like GPUs, this is even more important. In recent years, a great effort has taken place, see, for example, [18,24,30,31], trying to adapt classical algorithms to the

new parallel systems and developing new parallel techniques for sorting. Several libraries for GPU programming appeared in this field, like Thrust [32], CUB [6], Modern GPU [25], clogs [4] or VexCL [35].

In our work, we take advantage of the CUDA data-parallel primitives (CUDPP) library, described in [9]. CUDPP is a library of data-parallel algorithm primitives that are important building blocks for a wide variety of data-parallel algorithms, including sorting. The sorting algorithms are based on the work of Satish et al. [30]. The authors showed the performance of several sorting algorithms implemented on GPUs. Following the results of their work, we choose the parallel Radix sort which is included in version 2.1 of CUDPP. In addition, CUDPP provides a kernel-level API,[2] 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 global memory for each bundle. The global memory allocation is made for all bundles which means that the bundle's memory areas are adjacent and the accesses are faster.

5.2.3 *Estimators*

When the bundling stage is done, the exercise policy and the final option values can be computed by means of direct and PEs. In order to use the GPU memory efficiently, the obtained MC paths (once bundled) are sorted with respect to the bundles and stored in that way. We minimize the global memory accesses and the sorted version is much more efficient because, again, the bundle's memory areas are contiguous. For this purpose, we use again the CUDPP 2.1 library. Note that the data are already sorted in the case of equal-partitioning bundling.

For the DE, 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 threads collaborate in order to compute the continuation value which determines the early-exercise policy. As we mentioned before, $T/\Delta t$ stages of parallelization are performed, i.e. one per time step. Hence, in each stage of parallelization, we need to transfer only the data corresponding to the current time step from CPU memory to GPU global memory. This data cannot be transferred at once because we wish to take advantage of the optimized parallel sorting libraries for the sorting step after the bundling stage. This allows us to minimize and improve the global memory accesses when the sequential bundling is employed, i.e. considering k-means clustering. Anyway, this fact does not imply a reduction of the performance since the total transferred amount is the same. Note again that, employing equal-partitioning bundling, these transfers are avoided since this bundling technique is fully parallelizable.

Once the early-exercise policy is determined, the PE can be executed. For that, a new set of grid points has to be generated following the procedure described in Section 5.2.1. In the case of the PE, the parallelization can be done over paths because the early-exercise policy is already known (given by the previous computation of the DE) 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. Another sorting stage is needed to assign the paths to the corresponding bundle. Again, we use the sorting functions of CUDPP 2.1 for that purpose.

In the final stage for both estimators, a summation is necessary to determine the final option price. We take advantage of the Thrust library [32] to perform this reduction on the GPU.

We present a schematic representation of parallel SGBM in Algorithm 2, highlighting the parts which have been parallelized and considering the equal-partitioning version. The variables payoffData, expData and critData correspond to three matrices in which we store the necessary computations for the pricing, regression and sorting stages, respectively, for each path and each

exercise time. The α_t^β values are computed in the regression step and determine the early-exercise policy, which is later used in the PE calculation.

Algorithm 2: Parallel SGBM

Data: $S_{t_0}, X, \mu_\delta, \sigma_\delta, \rho_{i,j}, T, N, M$
// Generation of the grid points (Monte Carlo). Step I.
// Option value at terminal time $t = M$. Step II.
[payoffData, critData, expData] = MonteCarloGPU($S_{t_0}, X, \mu_\delta, \sigma_\delta, \rho_{i,j}, T, N, M$);
for *Time* $t = M \cdots 1$ **do**
  // Bundling. Step III.
  SortingGPU(critData[t-1]);
  begin CUDAThread per bundle $\beta = 1 \cdots \nu$
    // Exercise policy (Regression). Step IV.
    α_t^β = LeastSquaresRegression(payoffData[t]);
    // Continuation value. Step V.
    CV = ContinuationValue(α_t^β, expData[t-1]);
    // Direct estimator. Step V.
    DE = DirectEstimator(CV, payoffData[t-1]);
return *DE*;
// Generation of the grid points (Monte Carlo). Step I.
// Option value at terminal time $t = M$. Step II.
[payoffData, critData, expData] = MonteCarloGPU($S_{t_0}, X, \mu_\delta, \sigma_\delta, \rho_{i,j}, T, N, M$);
for *Time* $t = M \cdots 1$ **do**
  // Bundling. Step III.
  SortingGPU(critData[t-1]);
  begin CUDAThread per path $n = 1 \cdots N$
    // Continuation value. Step V.
    CV[n] = ContinuationValue(α_t^β, expData[t-1]);
    // Path estimator. Step V.
    PE[n] = PathEstimator(CV[n], payoffData[t-1]);
return *PE*;

6. Results

Experiments were performed on the Accelerator Island system of the Cartesius Supercomputer (more information in [3]) with the following main characteristics:

- Intel Xeon E5-2450 v2.
- NVIDIA Tesla K40m.
- C-compiler: GCC 4.4.7.
- CUDA version: 5.5.

All computations are done in double precision, because a high accuracy is required both in the bundling as well as in the regression computations. We consider the d-dimensional problem of pricing basket Bermudan options with the following characteristics:

- Initial state: $S_{t_0} = (40, 40, \ldots, 40) \in \mathbb{R}^d$.
- Strike: $X = 40$.

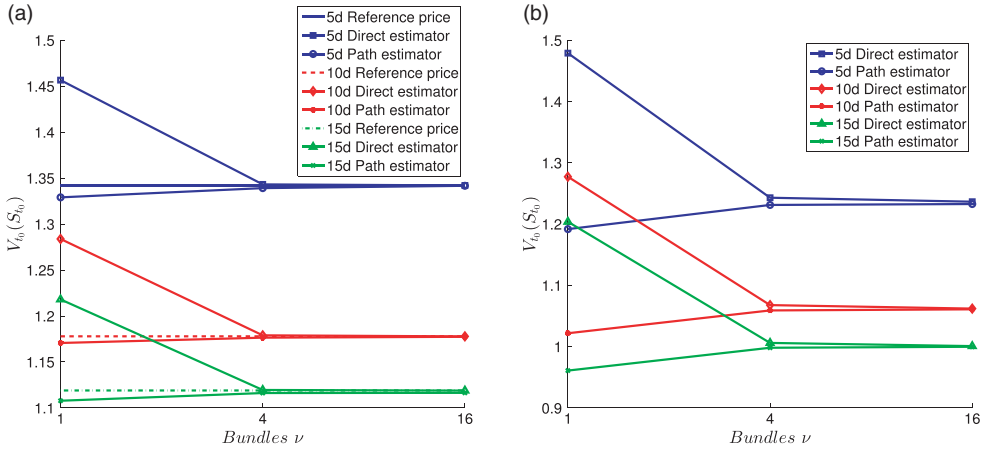

Figure 3. Convergence with equal-partitioning bundling technique. Test configuration: $N = 2^{18}$ and $\Delta t = T/M$: (a) geometric basket put option. (b) Arithmetic basket put option.

- 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: $M = 10$.

Specifically, geometric and arithmetic basket Bermudan put options are chosen in order to show the accuracy and performance. The number of basis functions is taken as $K = 3$. As the stochastic asset model, we first choose the multi-dimensional geometric Brownian motion (GBM), and for the discretization scheme we employ the Euler SDE discretization.

6.1 *Equal-partitioning: convergence test*

In the original SGBM paper, the authors have shown the convergence of SGBM using k-means bundling, in dependence of the number of bundles. For the equal-partitioning bundling, we perform a similar convergence study by pricing the previously mentioned options. Regarding the sorting criterion needed for the equal-partitioning technique, we choose the payoff criterion, i.e. we sort the MC scenarios following the geometric or arithmetic average of the assets for geometric and arithmetic basket options, respectively. Similar results can be obtained using other criteria like the product or the sum of the assets. In Figure 3, we show the convergence of the calculated option prices for both 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 [13], because a multi-dimensional geometric basket option can be reformulated as a one-dimensional option pricing problem. In the original paper for SGBM [17], we can also find a comparison with the LSM method [23].

6.2 *Bundling techniques: k-means vs. equal-partitioning*

With the convergence of the equal-partitioning technique shown numerically, we now increase drastically the number of bundles and, hence, the number of MC paths. For the two presented

Table 1. SGBM stages time (s) for the C and CUDA versions.

	k-Means			Equal-partitioning		
	MC	DE	PE	MC	DE	PE
Geometric basket Bermudan option						
C	82.42	234.37	203.77	101.77	41.48	59.16
CUDA	1.04	18.69	12.14	0.63	4.66	1.29
Speedup	79.25	12.88	16.78	161.54	8.90	45.86
Arithmetic basket Bermudan option						
C	78.86	226.23	203.49	79.22	39.64	58.65
CUDA	1.36	17.89	11.74	0.83	4.14	1.20
Speedup	57.98	12.64	17.33	95.44	9.57	48.87

Note: Test configuration: $N = 2^{22}$, $\Delta t = T/M$, $d = 5$ and $\nu = 2^{10}$.

Table 2. SGBM total time (s) for the C and CUDA versions.

	k-Means			Equal-partitioning		
	$d = 5$	$d = 10$	$d = 15$	$d = 5$	$d = 10$	$d = 15$
Geometric basket Bermudan option						
C	604.13	1155.63	1718.36	303.26	501.99	716.57
CUDA	35.26	112.70	259.03	8.29	9.28	10.14
Speedup	17.13	10.25	6.63	36.58	54.09	70.67
Arithmetic basket Bermudan option						
C	591.91	1332.68	2236.93	256.05	600.09	1143.06
CUDA	34.62	126.69	263.62	8.02	11.23	15.73
Speedup	17.10	10.52	8.48	31.93	53.44	72.67

Note: Test configuration: $N = 2^{22}$, $\Delta t = T/M$ and $\nu = 2^{10}$.

bundling techniques, we perform a time comparison between the C and CUDA implementations for multi-dimensional geometric and arithmetic basket Bermudan options. First of all, in Table 1, we show the execution times for the different SGBM stages, i.e. MC path generation, DE computation and PE computation, for the C and CUDA versions. We can see an overall improvement of the timing results for SGBM based on equal-partitioning bundling due to more efficient direct and PEs, which are around four times faster. The performance of the CUDA versions is remarkable, reaching a speedup of 160 for the MC simulation.

In Table 2, the total execution times for $d = 5$, $d = 10$ and $d = 15$ problems are presented. We observe a significant acceleration of the CUDA versions for both bundling techniques, with a special improvement in the case of the equal-partitioning. This is because the iterative process of k-means bundling penalizes parallelism and memory transfers, especially when the dimensionality increases, while equal-partitioning handles these issues in a more efficient way.

6.3 *High-dimensional problems*

The second goal is to drastically increase the problem dimensionality for basket Bermudan option pricing. In the case of the k-means bundling algorithm, this is not possible because of memory limitations. However, we save memory using the equal-partitioning technique which enables us to increase the problem dimensions. We can reduce the total number of MC paths, since by equal-partitioning each bundle has the same number of paths. This gives us accurate regressions in all bundles. The SGBM prices for geometric and arithmetic basket Bermudan put options are shown in Table 3. Again, the reference price for the geometric basket Bermudan option is obtained by the COS method [13] and we present the prices given by the DE and PE.

Table 3. Option price for a high-dimensional problem with equal-partitioning.

	$d = 30$	$d = 40$	$d = 50$
Geometric basket Bermudan option			
COS	1.057655	1.041889	1.032343
SGBM DE	1.057657	1.041888	1.032339
SGBM PE	1.057341	1.041545	1.031797
Arithmetic basket Bermudan option			
SGBM DE	0.937436	0.921009	0.911023
SGBM PE	0.934359	0.919695	0.909646

Note: Test configuration: $N = 2^{20}$, $\Delta t = T/M$ and $\nu = 2^{10}$.

Table 4. SGBM total time (s) for a high-dimensional problem with equal-partitioning.

	$\nu = 2^{10}$			$\nu = 2^{14}$		
	$d = 30$	$d = 40$	$d = 50$	$d = 30$	$d = 40$	$d = 50$
Geometric basket Bermudan option						
C	337.61	476.16	620.11	337.06	475.12	618.98
CUDA	4.65	6.18	8.08	4.71	6.26	8.16
Speedup	72.60	77.05	76.75	71.56	75.90	75.85
Arithmetic basket Bermudan option						
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

Note: Test configuration: $N = 2^{20}$ and $\Delta t = T/M$.

In Table 4, the execution times for pricing geometric and arithmetic basket Bermudan put options in different dimensions and with different numbers of bundles, ν, are shown. Note that the number of bundles hardly influences the execution times. With the equal-partitioning technique, the performance is mainly dependent on the number of paths and the dimensionality. For that reason, we can exploit the GPU parallelism arriving at a speedup of around 75 for the 50-dimensional problem in the case of geometric basket Bermudan option and around 100 for the 50-dimensional arithmetic basket Bermudan option.

Taking into account the accuracy and the performance of our CUDA implementation, we can even increase the dimension of the problem to higher values.

6.4 *Experiments with more general approach*

Parallel SGBM can thus be used to efficiently price high-dimensional basket Bermudan options under GBM or other processes for which the continuous characteristic function is available. However, as presented in Section 4, we can use SGBM also when the continuous characteristic function is not available. In this case, we first discretize the SDEs system and then determine the characteristic function, which changes with position and/or time. In order to get accurate approximations when using this discrete characteristic function, we need to improve the approximation of the expectations resulting from the use of the basis functions in Equation (11). For that, we can increase the number of time steps, i.e. we can reduce the differences between MC paths within each bundle (with this, the regression becomes more accurate). We perform a convergence test to show this behaviour. For the experiment, we choose the GPU version of SGBM with equal-partitioning bundling, because this is the most efficient implementation and the performance is

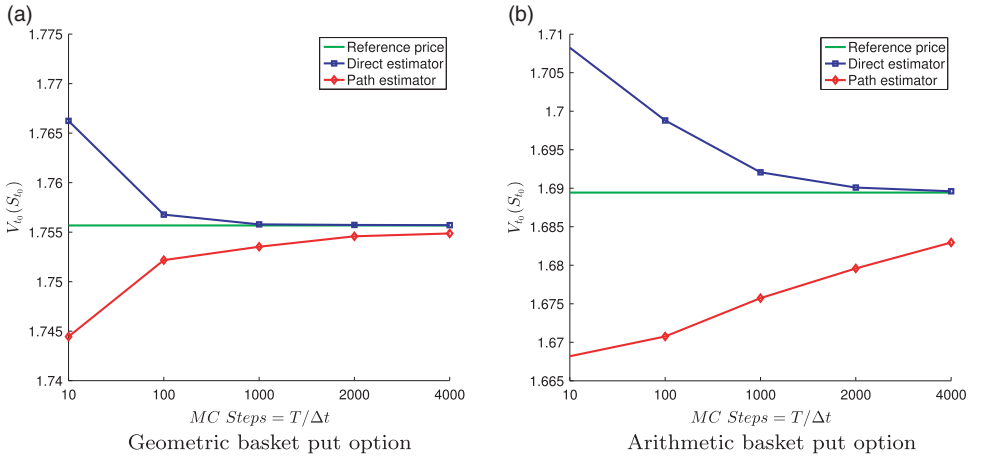

Figure 4. CEV model convergence, $\gamma = 1.0$. Test configuration: $N = 2^{16}$, $\nu = 2^{10}$ and $d = 2$: (a) geometric basket put option. (b) Arithmetic basket put option.

independent of the number of bundles. An increasing number of time steps makes the C-version too expensive again.

We consider a parametric local volatility model called the constant elasticity of variance (CEV) model. This model can be obtained by substituting

$$\mu_{\delta}(\boldsymbol{S}_t) = (r_t - q_{\delta})S_t^{\delta}, \quad \sigma_{\delta}(\boldsymbol{S}_t) = \sigma_{\delta}(S_t^{\delta})^{\gamma}, \tag{12}$$

in Equation (1). $\gamma \in [0, 1]$ is a free parameter called the variance elasticity, and q_{δ} and σ_{δ}, $\delta = 1, 2, \ldots, d$, are constant dividend yield and volatility, respectively.

The computation of the derivatives in Equation (10) is performed by using Wolfram Mathematica 8 [37].

In Figure 4, a convergence test regarding the number of MC time steps is presented. A two-dimensional problem is considered and $\gamma = 1.0$ is used in Equation (12) to compare our approximation with the reference price given by the COS method (in the case of a geometric basket Bermudan put option) and the original SGBM following geometric Brownian motion (for an arithmetic basket Bermudan put option). The figure shows that we can get improved approximations of the option price by increasing the number of MC time steps.

In Figure 5, the same convergence test as $d = 2$ case is presented for five-dimensional problem. Again, we can see that the approximations are better when the number of MC time steps is increased. In both cases, $d = 2$ and $d = 5$, the DE gives a better approximation compared with the reference price. It is also observed that the direct and PEs for the new approach perform similarly as in the case of the original SGBM and they can provide a confidence interval for the option price.

Once we find an appropriate combination of the number of MC paths, bundles and time steps, we can use our parallel SGBM method to price products under local volatility dynamics. In Tables 5 and 6, the results of pricing two-dimensional and five-dimensional geometric and arithmetic basket Bermudan put options are presented. We take a different values for γ in the CEV model here. Note that we did not encounter any problems with the Euler discretization of the CEV dynamics in our test cases. The execution times (s) are around 120 and 150 for two-dimensional and five-dimensional problems, respectively.

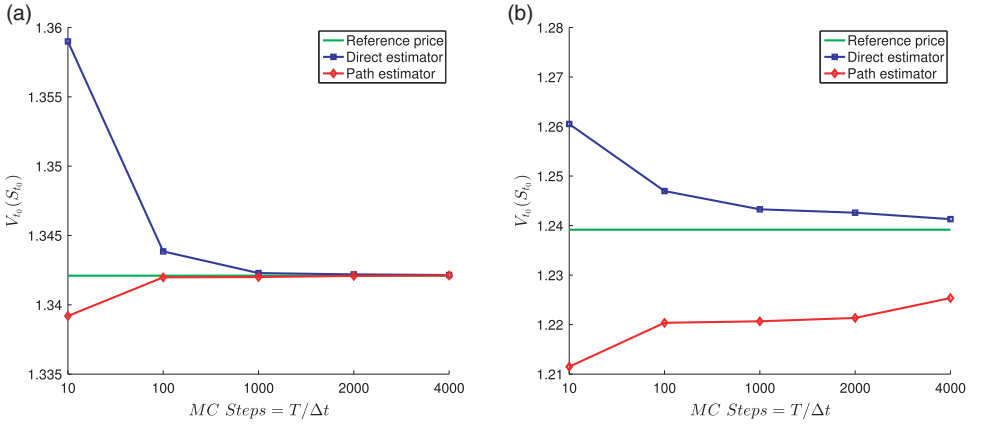

Figure 5. CEV model convergence, $\gamma = 1.0$. Test configuration: $N = 2^{16}$, $\nu = 2^{10}$ and $d = 5$: (a) geometric basket put option. (b) Arithmetic basket put option.

Table 5. CEV option pricing.

	$\gamma = 0.25$	$\gamma = 0.5$	$\gamma = 0.75$	$\gamma = 1.0$
Geometric basket Bermudan option				
SGBM DE	0.001420	0.055636	0.411066	1.755705
SGBM PE	0.001395	0.055620	0.410758	1.754869
Arithmetic basket Bermudan option				
SGBM DE	0.001417	0.055340	0.404410	1.688609
SGBM PE	0.001346	0.055249	0.400400	1.682956

Note: Test configuration: $N = 2^{16}$, $\Delta t = T/4000$, $\nu = 2^{10}$ and $d = 2$.

Table 6. CEV option pricing.

	$\gamma = 0.25$	$\gamma = 0.5$	$\gamma = 0.75$	$\gamma = 1.0$
Geometric basket Bermudan option				
SGBM DE	0.000291	0.029395	0.276030	1.342147
SGBM PE	0.000274	0.029322	0.275131	1.342118
Arithmetic basket Bermudan option				
SGBM DE	0.000289	0.029089	0.267943	1.241304
SGBM PE	0.000288	0.028944	0.267214	1.225359

Note: Test configuration: $N = 2^{16}$, $\Delta t = T/4000$, $\nu = 2^{10}$ and $d = 5$.

7. Conclusions

In this paper, we have presented an efficient implementation of the SGBM 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 and explore more general ways to compute the continuation values. A general approach for approximating expectations based on the so-called discrete characteristic function was presented. This approach allowed us to use SGBM for underlying asset models for which the continuous characteristic function is not available.

Several results under the CEV local volatility model were presented to show the performance and the accuracy of the new proposed techniques.

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

We think that the parallel SGBM technique presented forms a fine basis to deal with credit valuation adjustment of portfolios with financial derivatives in the near future.

Acknowledgments

The authors would like to thank Shashi Jain, ING Bank, for providing support and the original codes of the SGBM.

Thanks to SURFsara for providing the access to Accelerator Island system of the Cartesius Supercomputer to perform our experiments.

Disclosure statement

No potential conflict of interest was reported by the authors.

Funding

The first author is supported by the European Union in the FP7-PEOPLE-2012-ITN Program [grant number 304617] (FP7 Marie Curie Action, Project Multi-ITN STRIKE – Novel Methods in Computational Finance).

Notes

1. Double-precision floating-point format.
2. Application programming interface.

References

[1] L.A. Abbas-Turki and B. Lapeyre, *American Options Pricing on Multi-core Graphic Cards*, 2009 International Conference on Business Intelligence and Financial Engineering, Beijing, 2009, pp. 307–311.
[2] M. Benguigui and F. Baude, *Fast American Basket Option Pricing on a Multi-GPU Cluster*, Proceedings of the 22nd High Performance Computing Symposium, April, Tampa, FL, 2014, pp. 1–8.
[3] Cartesius webpage, https://www.surfsara.nl/systems/cartesius.
[4] Clogs webpage, http://sourceforge.net/projects/clogs/.
[5] S. Cook, *CUDA Programming: A Developer's Guide to Parallel Computing with GPUs*, 1st ed., Morgan Kaufmann Publishers Inc., San Francisco, CA, 2013.
[6] CUB webpage, http://nvlabs.github.io/cub/.
[7] CUDA programming guide, http://docs.nvidia.com/cuda/cuda-c-programming-guide/.
[8] CUDA webpage, http://www.nvidia.com/object/cuda_home_new.html.
[9] CUDPP webpage, http://cudpp.github.io/.
[10] cuRAND webpage, https://developer.nvidia.com/curand.
[11] V. Cvetanoska and T. Stojanovski, *Using high performance computing and Monte Carlo simulation for pricing American options*, CoRR abs/1205.0106 (2012). Available at http://arxiv.org/abs/1205.0106.
[12] D.M. Dang, C.C. Christara, and K.R. Jackson, *An efficient graphics processing unit-based parallel algorithm for pricing multi-asset American options*, Concurr. Comput.: Pract. Exp. 24 (2012), pp. 849–866. Available at http://dx.doi.org/10.1002/cpe.1784.
[13] F. Fang and C.W. Oosterlee, *Pricing early-exercise and discrete barrier options by Fourier-cosine series expansions*, Numer. Math. 114 (2009), pp. 27–62.
[14] R. Farber, *CUDA Application Design and Development*, Morgan Kaufmann Publishers Inc., San Francisco, CA, 2011.
[15] M. Fatica and E. Phillips, *Pricing American Options with Least Squares Monte Carlo on GPUs*, Proceedings of the 6th Workshop on High Performance Computational Finance, WHPCF'13, Denver, CO. Available at http://doi.acm.org/10.1145/2535557.2535564, ACM, New York, NY, USA, 2013, pp. 5:1–5:6.

[16] M.B. Haugh and L. Kogan, *Pricing American options: A duality approach*, Oper. Res. 52 (2004), pp. 258–270.
[17] S. Jain and C.W. Oosterlee, *The Stochastic Grid Bundling Method: Efficient Pricing of Bermudan Options and Their Greeks*, 2013. Available at http://ssrn.com/abstract = 2293942.
[18] B. Jan, B. Montrucchio, C. Ragusa, F.G. Khan, and O. Khan, *Fast parallel sorting algorithms on GPUs*, Int. J. Distrib. Parallel Syst 14 (2012), pp. 107–118.
[19] V. Kindratenko (ed.), *Numerical Computations with GPUs*, Springer, 2014. Available at http://www.springer.com/us/book/9783319065472
[20] D.B. Kirk and W.m.W. Hwu, *Programming Massively Parallel Processors: A Hands-on Approach*, Elsevier, Burlington, 2010.
[21] D.E. Knuth, *The Art of Computer Programming, Sorting and Searching, Volume 3*, 2nd ed., Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, 1998.
[22] H. Kobayashi, B.L. Mark, and W. Turin, *Probability, Random Processes, and Statistical Analysis*, Cambridge University Press, Cambridge, 2012.
[23] F.A. Longstaff and E.S. Schwartz, *Valuing American options by simulation: A simple least-squares approach*, Rev. Financ. Stud. 14 (2001), pp. 113–147. Available at http://ideas.repec.org/a/oup/rfinst/v14y2001i1p113-47.html.
[24] M.J. Misic and M.V. Tomasevic, *Data sorting using graphics processing units*, Telfor J. 4 (2012), pp. 43–48.
[25] Modern GPU webpage, http://nvlabs.github.io/moderngpu/.
[26] G. Pagès and B. Wilbertz, *GPGPUs in computational finance: Massive parallel computing for American style options*, Concurr. Comput.: Pract. Exp. 24 (2012), pp. 837–848. Available at http://dx.doi.org/10.1002/cpe.1774.
[27] L.C.G. Rogers, *Monte Carlo valuation of American options*, Math. Financ. 12 (2002), pp. 271–286.
[28] M.J. Ruijter and C.W. Oosterlee, *Numerical Fourier Method and Second-order Taylor Scheme for Backward SDEs in Finance*, Working paper, 2014.
[29] J. Sanders and E. Kandrot, *CUDA by Example: An Introduction to General-purpose GPU Programming*, Addison-Wesley, Michigan, 2011.
[30] N. Satish, M. Harris, and M. Garland, *Designing Efficient Sorting Algorithms for Manycore GPUs*, Proceedings of the 23rd IEEE International Parallel and Distributed Processing Symposium, Rome, May 2009.
[31] K. Thouti and S. Sathe, *An OpenCL method of parallel sorting algorithms for GPU architecture*, Int. J. Exp. Algorithms 3 (2012), pp. 1–8.
[32] Thrust webpage, http://thrust.github.io/.
[33] J.N. Tsitsiklis and B. Van Roy, *Regression methods for pricing complex American-style options*, IEEE Trans. Neural Netw. 12 (2001), pp. 694–703.
[34] N.G. Ushakov, *Selected Topics in Characteristic Functions (Modern Probability and Statistics)*, Mouton De Gruyter, Utrecht, 1999.
[35] VexCL webpage, http://ddemidov.github.io/vexcl/.
[36] N. Wilt, *The CUDA Handbook: A Comprehensive Guide to GPU Programming*, Addison-Wesley, Indiana, 2013.
[37] Wolfram Mathematica webpage, http://www.wolfram.com/mathematica/.