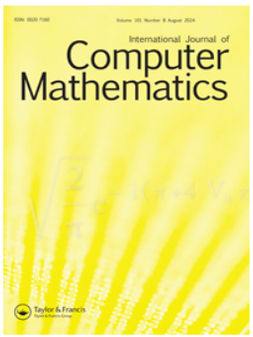

International Journal of Computer Mathematics

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

Efficient parallel Monte-Carlo techniques for pricing American options including counterparty credit risk

Íñigo Arregui, Álvaro Leitao, Beatriz Salvador & Carlos Vázquez

To cite this article: Íñigo Arregui, Álvaro Leitao, Beatriz Salvador & Carlos Vázquez (2024) Efficient parallel Monte-Carlo techniques for pricing American options including counterparty credit risk, International Journal of Computer Mathematics, 101:8, 821-841, DOI: 10.1080/00207160.2023.2172322

To link to this article: https://doi.org/10.1080/00207160.2023.2172322

INTERNATIONAL JOURNAL OF COMPUTER MATHEMATICS
2024, VOL. 101, NO. 8, 821–841
https://doi.org/10.1080/00207160.2023.2172322

Taylor & Francis
Taylor & Francis Group

RESEARCH ARTICLE

Efficient parallel Monte-Carlo techniques for pricing American options including counterparty credit risk

Íñigo Arregui [a,b], Álvaro Leitao [a,b], Beatriz Salvador [c] and Carlos Vázquez [a,b]

[a]M2NICA research group, University of A Coruña, A Coruña, Spain; [b]CITIC research centre, A Coruña, Spain; [c]Banco Santander, Boadilla del Monte, Spain

ABSTRACT

In this article we mainly propose a numerical scheme, based on the novel Stochastic Grid Bundling Method (SGBM), to price American options in the presence of counterparty credit risk. More precisely, we consider the regression techniques (regress later) employed in the SGBM method and take advantage of the bundling structure to develop an efficient parallel strategy that is implemented on a GPU architecture. Also, a novel interpolation-based technique is efficiently applied in the XVA computation. Besides the advantages obtained in the sequential version, when compared with the more classical Least Squares Method, we show the relevant speedup of the parallel GPU-based version with respect to the sequential CPU-based one.

ARTICLE HISTORY
Received 15 October 2022
Revised 17 January 2023
Accepted 18 January 2023

KEYWORDS
Counterparty credit risk; Total value adjustment; American options; Monte Carlo methods; GPU computing

2020 AMS SUBJECT CLASSIFICATIONS
65C05; 65C30; 65Y05

1. Introduction

One of the consequences of the 2007 financial crisis, when important entities went bankrupt, was the relevance given since that moment to the counterparty risk, that became an important issue to consider in all financial contracts. The counterparty risk can be described as the risk, for each party of a contract, that the counterparty will not live up to its contractual obligations. Many financial analysts and institutions consider that the crisis was due to the mistakes made in the financial system, namely in the management of the different risks. More precisely, the consideration of a low probability of default and the complexity of the financial derivatives were two of the keys that led to the crisis. Therefore, a review of the counterparty risk consideration has been addressed, and different adjustments on the risk–free value are being included in the derivative pricing. The most common adjustments are Credit Value Adjustment (CVA), to compensate the credit risk due to the counterparty, Debit Value Adjustment (DVA) to hedge the issuer probability of default, Funding Value Adjustment (FVA), to ensure that a dealer recovers its average funding costs when it trades and hedges derivatives, or Collateral Value Adjustment (CollVA) in case a collateral is present in the contract. Nowadays, additional adjustments such as Margin Value Adjustment (MVA) or Capital Value Adjustment (KVA) are also introduced in the pricing of derivatives. The set of this adjustments is referred to as total value adjustment and denoted by XVA. Moreover, the derivative value including the previous adjustments is referred to as risky derivative value.

Three methodologies are mainly used to include the XVA in the price of different products. The first one starts from the works by Piterbarg [48] (where funding costs are included) and Burgard and Kjaer [14] (that considers funding costs and bilateral counterparty risk), and lead to partial differential equations (PDE) models. In both works the PDE formulation is based on the use of appropriate hedging arguments combined with Itô's lemma for jump–diffusion processes [47], and mainly deal

CONTACT Álvaro Leitao alvaro.leitao@udc.es

with European style options. Recent works also based in PDEs to compute XVA are [4,6,7,20]. In such works, different assumptions are made about the intensity of default of the counterparties, thus posing problems depending on one, two or more stochastic underlying factors. A second methodology to compute the XVA is based on the solution of backwards stochastic differential equations (BSDE) [9,23,24]. And the last one involves Monte Carlo methods as the XVA can be expressed in terms of conditional expectations [11,16,17,31,46], that can be approximated by a dynamic programming formulation. By far, this approach is the most used by banks and financial entities since it offers much more flexibility, straightforward extension to higher dimensions and easy interpretation/implementation with respect to PDE- or BSDE-based approaches. We therefore focus on Monte Carlo methods in this work.

Among the Monte Carlo-based techniques, the so-called least square method (LSM) introduced by Longstaff and Schwartz in [44] is the most popular and widely utilized methodology to address pricing problems of derivatives with early exercise features. Basically, the method approximates recursively, moving backwards in time, the so-called continuation value (a conditional expectation) by regressing the future optimal discounted cash flows against a set of basis functions of the underlying state variables at that time epoch. In [35], a variant of LSM is proposed, where the basis for regression is constructed by using the state variables at the *next time step*, as opposed to the current time step. In the literature, this approach has commonly been referred to as *regression later* (or regress-later), leading to the term *regress-now* to describe the original strategy. In addition, LSM was already extended in [5] to price American options including XVA adjustments.

Jain and Oosterlee [38] have recently introduced an alternative to LSM, namely the Stochastic Grid Bundling Method (SGBM), which combines several advantageous existing features and provides stable and accurate estimations. This methodology is also based on the dynamic programming formulation, by recursively computing the option price moving backwards. Furthermore, the method employs regressed value functions within local partitions (bundles) of the state space to approximate continuation values at the next time step. Thus, SGBM is a regress-later technique where the regression is locally performed. In yet another advantage, SGBM is well-known for allowing the efficient computation of stable and precise financial sensitivities (known as Greeks), both at the initial time and *pathwise*, as shown in [37]. Besides the mentioned advantages, SGBM presents another desirable property: it is fully parallelizable. In general, the methodologies based on backward dynamic programming are not suitable candidates for parallelization, due to the intrinsic dependency between consecutive steps. However, thanks to the bundling, a parallelization opportunity appears in the space 'direction'. This fact has been already exploited in [43], where the authors employ parallel GPU computing to accelerate the early-exercise option valuation in a high-dimensional setting and introduce an efficient (in terms of parallelism) procedure to determine the space partitions. Here, we extend that work to tackle with the American option pricing considering counterparty risk and valuation adjustments. That extension is not trivial, since the inclusion of XVA component introduces several difficulties caused by the nested nature of the counterparty credit risk computations. Especially relevant is the GPU memory management when Monte Carlo subsimulations are required. Here we combine an interpolation-based pricing alternative with a fully parallelized algorithm, to come up with a methodology that decouples the most expensive parts from the practical application. In a sense, this approach provides similar benefits (in terms of combining offline and online phases) as the neural network-based techniques, but keeping the interpretability of the whole algorithm. Other recent developments employing parallel GPU computing in the XVA context can be found in [1,2,19]. Further applications of GPU-based approaches in computational finance include [32,33], for example.

The plan of the paper is the following. In Section 2, the derivative model for pricing American options is deduced. In Section 3, we present different methodologies using Monte Carlo techniques in order to compute the risky derivative value; moreover, a comparison between the methodologies is provided. Section 4 proposes some alternatives based on GPU computation in order to improve the computational time. With the aim of validating the advantages of applying the Stochastic Grid

Bundling Method to price risky American options, some tests have been also carried out. Finally, conclusions are presented in Section 5.

2. XVA for American options

In this section, we formulate the problem of pricing American options taking into account the XVA. A more detailed deduction can be found in [4,5].

Firstly, we assume the classical scenario. We consider a derivative trade between two defaultable counterparties, the issuer B and the investor C. From the point of view of the issuer, the risky derivative value at time t is denoted by $W_t = W(t, S_t, J_t^B, J_t^C)$, where S_t represents the price of the underlying asset, while J_t^B and J_t^C refer to two independent jump processes changing from 0 to 1 on default of B or C, respectively. The risk-free derivative (i.e. without counterparty risk) value is denoted by $V_t = V(t, S_t)$, and can be computed by the classical Black-Scholes problem for American options. In this setting, the stock price is modelled as a geometric Brownian motion, thus satisfying the following stochastic differential equation (SDE):

$$\mathrm{d}S_t = r_R S_t \,\mathrm{d}t + \sigma S_t \,\mathrm{d}Z_t, \tag{1}$$

where r_R is the underlying asset short term repurchase agreement (REPO) rate, σ is the volatility and Z_t is a Wiener process under the risk-neutral probability measure. Note that we are assuming that there exists a liquid REPO market on the underlying asset (see [13], for example).

When considering counterparty risk, the variation of the derivative value in case one counterparty defaults is given in terms of the mark–to–market (denoted by $\mathcal{M}$) close out as

$$W(t, S, 1, 0) = \mathcal{M}^+(t, S) + R_B \mathcal{M}^-(t, S)$$

if counterparty B defaults first, or

$$W(t, S, 0, 1) = R_C \mathcal{M}^+(t, S) + \mathcal{M}^-(t, S),$$

if counterparty C defaults first. In the previous expressions, $R_B, R_C \in [0, 1]$ represent the recovery rates on the derivatives position of parties B and C, respectively, while $z^+ = \max\{z, 0\}$ and $z^- = \min\{z, 0\}$. Applying classical hedging arguments to a self-financing portfolio, and using Itô's Lemma for jump diffusion processes, we can deduce the inequality that models the American option price, W, including counterparty risk:

$$\frac{\partial W}{\partial t} + \mathcal{A}W - rW \leq (\lambda_B + \lambda_C)W + s_F \mathcal{M}^+ - \lambda_B(\mathcal{M}^+ + R_B \mathcal{M}^-) - \lambda_C(\mathcal{M}^- + R_C \mathcal{M}^+), \tag{2}$$

where r is the risk-free interest rate, λ_B and λ_C are the default intensities of B and C, respectively, and s_F is the funding cost of the entity. Moreover, the differential operator $\mathcal{A}$ is given by:

$$\mathcal{A} = \frac{\sigma^2 S^2}{2} \frac{\partial^2}{\partial S^2} + r_R S \frac{\partial}{\partial S}.$$

Note that the previously introduced REPO rate r_R and the risk free discount rate r are different, the latter being used to discount the collateralized derivative in cash.

Remark 2.1: In the previous model statement we have mainly followed [4,5,7,13–15] using PDE-based formulations. Alternative presentations of XVA, based on BSDEs, are also possible [1,2,11,12,23,24,46]. Nevertheless, we prefer to work on this simple formulation and focus on the advantageous SGBM methodology and its efficient computational implementation, which can be extended to more complex models.

As in previous works [4,5,7], we consider two possible values for the mark–to–market: the risk–free value and the risky derivative value. Next, we introduce the models for both cases. A more detailed deduction can be found in [5]. For the first one a nested Monte Carlo algorithm should be addressed, while for the second one an iterative algorithm is used to solve the nonlinearity. Then, different methodologies are proposed according to the mark–to–market value.

2.1. The linear problem ($\mathcal{M} = V$)

Firstly, we focus on the case where the mark–to–market is equal to the risk–free value, V. We mainly follow [5]. We are pricing an American option, which can be exercised at any time $t \in (0, T]$. We denote its exercise value at any time $t \in (0, T]$ as

$$h^*(t, S_t) = H(S_t), \tag{3}$$

where $H(S_t)$ represents the payoff of the option and S_t is a Markovian process.

In a first step, we consider $\mathcal{M} = V$ in (2) and denote by g the function given by:

$$g(V) = (R_B\lambda_B + \lambda_C)V^- + (R_C\lambda_C + \lambda_B)V^+ - s_F V^+.$$

Next, following [47] and applying the Feynman-Kac theorem, the risky derivative value can be written in terms of expectations. Then, at time $t = 0$ and for an underlying value S_0, the risk-free value and the risky value are given by

$$V_0(S_0) = \sup_{\tau\in\mathcal{T}_0} \mathbb{E}_0\left[e^{-r\tau}h^*(\tau, S_\tau)\right],$$

$$W_0(S_0) = \sup_{\tau\in\mathcal{T}_0} \mathbb{E}_0\left[e^{-r_0\tau}h^*(\tau, S_\tau) + \int_0^\tau e^{-r_0 u}g(V(u, S(u)))\,\mathrm{d}u\right],$$

respectively, where $r_0 = r + \lambda_B + \lambda_C$ and $\mathcal{T}_t$ denotes the set of admissible stopping times in $[t, T]$.

We first discretize the time interval by introducing a finite and increasing set of instants $0 = t_0 < t_1 < t_2 < \ldots < t_M = T \subset [0, T]$, so that we approximate the American option by a Bermudan one, considering that the option can be only exercised in t_i $(i = 0, 1, \ldots, M)$. Then, we denote by $S_i = S(t_i)$, $i = 1, 2, \ldots, M$, the asset price at the i-th exercise opportunity and compute such values using the analytical solution for the SDE (1). Thus, for an arbitrary initial value S_0, the asset price at time t is given by:

$$S_t = S_0 \exp\left(\left(r_R - \frac{\sigma^2}{2}\right)t + \sigma Z_t\right). \tag{4}$$

Taking into account this time discretization, the American option with counterparty risk can be priced through the dynamic programming approach developed in [5]. Thus,

$$\begin{aligned} W_M(s) &= h(T, s), \quad S_M = s \\ W_{i-1}(s) &= \max\left\{h_{i-1}(s),\ \mathbb{E}_{t_{i-1}}\left[W_i(S_i) + \int_{t_{i-1}}^{t_i} e^{-r_0 u}g(V(u, S(u)))\,\mathrm{d}u \,\middle|\, S_{i-1} = s\right]\right\}, \end{aligned} \tag{5}$$

where $h_i(s) = D_{0,i}h^*(t_i, s)$, for $i = M, M-1, \ldots, 1$.

In American options pricing, it is also interesting to state stopping rules and the exercise region. In that sense, any stopping time τ determines the sub-optimal value

$$W_0^\tau(S_0) = \mathbb{E}_0\left[h_\tau(S_\tau) + \int_0^\tau e^{-r_0 u}g(V(u, S(u)))\,\mathrm{d}u\right].$$

Therefore, our goal is to choose the optimal stopping time, which will be determined by

$$\tau^* = \min\left\{\tau_i \in \{t_1, \ldots, t_M\} : h_i(S_i) \geq W_i(S_i)\right\}, \tag{6}$$

so that the exercise region associated to W_i at the i-th exercise date is the set $\{s : h_i(s) = W_i(s)\}$. We introduce the continuation value C_i, which can be computed in an iterative way as:

$$\begin{aligned} &C_M(s) = 0, \\ &C_i(s) = \mathbb{E}_{t_i}\left[W_{i+1}(S_{i+1}) + \int_{t_i}^{t_{i+1}} e^{-r_0 u} g(V(u, S(u)))\, \mathrm{d}u \,\middle|\, S_i = s \right], \end{aligned} \tag{7}$$

for $i = M-1, \ldots, 0$. We deduce in [5] that the optimal stopping rule can be rewritten as

$$\tau^* = \min\left\{ \tau_i \in \{t_1, \ldots, t_M\} : h_i(S_i) \geq C_i(S_i) \right\} \tag{8}$$

and, in terms of this optimal stopping time, the option value is determined by

$$W_0^{\tau^*}(S_0) = \mathbb{E}_0\left[h_{\tau^*}(S_{\tau^*}) + \int_0^{\tau^*} e^{-r_0 u} g(V(u, S(u)))\, \mathrm{d}u \right].$$

2.2. The nonlinear problem ($\mathcal{M} = W$)

We now consider the mark–to–market equal to the risky derivative value, W. Then, the risky American option value at time $t = 0$ satisfies the nonlinear equation:

$$W_0(S_0) = \sup_{\tau \in \mathcal{T}_0} \mathbb{E}_0\left[e^{-r\tau} h^*(\tau, S_\tau) + \int_0^{\tau} e^{-ru} f(W(u, S(u)))\, \mathrm{d}u \right],$$

where the function f is given by:

$$f(W) = -(1 - R_B)\lambda_B W^- - (1 - R_C)\lambda_C W^+ - s_F W^+.$$

Recall that the asset prices follow the geometric Brownian motion process defined in (1). Once again, the time period is discretized with $M + 1$ time steps to simulate a continuously exercisable American option. Thus, the asset price at each time step is computed as the exact value (4).

Now, using a dynamic programming formulation, the value of the American option with counterparty risk can be computed by the recursive formula:

$$\begin{aligned} &W_M(s) = h(T, s), \quad S_M = s \\ &W_{i-1}(s) = \max\left\{ h_{i-1}(s),\ \mathbb{E}_{t_{i-1}}\left[W_i(S_i) + \int_{t_{i-1}}^{t_i} e^{-ru} f(W(u, S(u)))\, \mathrm{d}u \,\middle|\, S_{i-1} = s \right] \right\}, \end{aligned}$$

for $i = M, M-1, \ldots, 1$, and the expression of the continuation value can be written as follows:

$$C_i(s) = \mathbb{E}_{t_i}\left[W_{i+1}(S_{i+1}) + \int_{t_i}^{t_{i+1}} e^{-ru} f(W(u, S(u)))\, \mathrm{d}u \,\middle|\, S_i = s \right].$$

See [5] for further details.

Remark 2.2: In the derivations above (as well as the numerical methods introduced in the following), we restrict ourselves to the one-dimensional case. However, the algorithms presented in this work (since they rely on Monte Carlo methods) can be easily extended to a multi-dimensional setting.

3. Pricing with Monte Carlo-based methods: LSM and SGBM

Monte Carlo-based techniques to solve these types of early-exercise problems rely on the dynamic programming formulation and the approximation of the continuation value by regression. Following the same idea, in the previous section we have described how to estimate continuation values when considering counterparty risk. Next, we propose different ways of computing approximations, $\widehat{C}_i(s)$, of the real continuation values, $C_i(s)$, by employing Monte Carlo-based approaches. In this section we introduce two alternatives, which will be compared in Section 3.3. Although we focus on the linear case ($M = V$), similar procedures can be followed in the nonlinear one, if the mark–to–market is equal to the risky derivative value.

From (7), the value $C_i(s)$ can be computed by regressing the term,

$$W_{i+1}(S_{i+1}) + \int_{t_i}^{t_{i+1}} e^{-r_0 u} g(V(u, S(u)))\, \mathrm{d}u,$$

on the current state of the asset price s. Thus, C_i can be approximated by a linear combination of known functions of the current state using a least-squares regression, which leads to the approximations $\widehat{C}_i$. This is the starting point of the regression-based methodologies.

In Sections 3.1 and 3.2, we briefly describe two of the most successful methods addressing this problem, namely the classical *Least-Squares Method* (LSM) by Longstaff and Schwartz [44] and the recently introduced *Stochastic Grid Bundling Method* (SGBM) by Jain and Oosterlee [39]. For the reasons previously mentioned in Section 1, we do not present any comparison with PDE-based methods (except for obtaining the reference solutions used in the numerical experiments).

Note that for the case where $M = W$, the computation of the risky derivative value requires also such value which, following the approach proposed in [5], is handled by implementing a fixed-point iteration. Then, the regression might need to be computed several times although, in practice, the number of iterations is very low (up to two or three).

Both methods start by generating N independent copies of sample paths, $s_{1,j}, s_{2,j}, \ldots, s_{M,j}$ (for $j = 1, 2, \ldots, N$), of the underlying process. Ideally, these sample paths are obtained by means of an exact simulation scheme given by (4). If an exact scheme is not available, discretization techniques like *Euler-Maruyama* or *Milstein* can be employed [42]. As the underlying process considered here (see (5)) is driven by the *Geometric Brownian Motion* (log-normally distributed), the sample generation is exact.

3.1. LSM: lower bounds estimator using least-squares regressions

The least-squares Monte Carlo method (LSM), proposed in 2001 by Longstaff and Schwartz [44], is the most widely used method for pricing early-exercise options. Basically, the method approximates recursively, moving backwards in time, the continuation value (a conditional expectation) by regressing the future optimal discounted cash flows against a set of basis functions of the underlying state variables at that time epoch. The obtained approximation provides a quasi-optimal exercise policy, resulting in a lower bounds estimation.

Thus, the continuation value is firstly written as a linear combination of basis functions as follows [5]:

$$\begin{aligned} C_i(s_{i,j}) &= \mathbb{E}_{t_i}\left[W_{i+1}(S_{i+1}) + \int_{t_i}^{t_{i+1}} e^{-r_0 u} g(V(u, S(u)))\, \mathrm{d}u \,\middle|\, S_i = s_{i,j} \right] \\ &\approx \sum_{k=1}^{K} b_{ik} \psi_k(s_{i,j}) = b_i^T \psi(s_{i,j}), \end{aligned} \tag{9}$$

where $b_i = (b_{i1}, \ldots, b_{iK})^T$ are the regression coefficients at time t_i and $\psi(s) = (\psi_1(s_{i,j}), \ldots, \psi_K(s_{i,j}))^T$ is the vector of basis functions. Different bases can be used to approximate the continuation

value. We focus on the weighted Laguerre polynomials:

$$\psi_k(x) = e^{-x/2} L_{k-1}(x), \quad j = 1, 2, \ldots$$

where L_k is the $k-$th Laguerre polynomial.

As described in [5], we determine the expression of the regression coefficients b_i using a least-squares optimization technique. More precisely, the function to minimize is given by:

$$\varphi(b_i) = \mathbb{E}_{t_i}\left[\left(\psi(S_i)^T b_i - \mathbb{E}_{t_i}\left[W_{i+1}(S_{i+1}) + \int_{t_i}^{t_{i+1}} e^{-r_0 u} g(V(u, S(u)))\,\mathrm{d}u \,\middle|\, S_i = s_{i,j}\right]\right)^2\right].$$

Thus, we vanish the derivatives of φ with respect to b_i, so that the expression of coefficient b_i is approximated by $\widehat{b}_i$, which satisfies the linear system of equations:

$$A_i^{\psi} \widehat{b}_i = d_i^{\psi},$$

where A_i^{ψ} and d_i^{ψ} are described in [5] and can be easily estimated by Monte Carlo simulations. Finally, the continuation value C_i can be approximated by:

$$\widehat{C}_i(s_{i,j}) = \widehat{b}_i^T \psi(s_{i,j}), \tag{10}$$

and the risky derivative value can be replaced by its estimated value

$$\widehat{W}_{i+1}(s_{i+1,j}) = \max\left\{h_{i+1}(s_{i+1,j}),\, \widehat{C}_{i+1}(s_{i+1,j})\right\}.$$

Let us point out that in this procedure (sketched as Algorithm 1) we have to apply an *inner* Monte Carlo method at each time step and for each asset price path, thus making this approach very expensive from the computational point of view.

Algorithm 1 LSM

(1) Simulate N independent paths $\{s_{1,j}, s_{2,j}, \ldots, s_{M,j}\}$ $(j = 1, 2, \ldots, N)$ of the asset prices process.
(2) At maturity time t_M, $\widehat{W}_M(s_{M,j}) = h_M(s_{M,j})$.
(3) Apply backward induction for $i = M - 1, \ldots, 1$.
- Compute the classical Longstaff-Schwartz approximation with $s_0 = s_{i,j}$ for the time interval $[t_i, T]$ to obtain $W_{i,j}$.
- Given the estimated value $\widehat{W}_{i+1,j}$ and $W_{i,j}$ $(j = 1, \ldots, N)$, compute $\widehat{b}_i$ as the solution of the linear system $A_i^{\psi} \widehat{b}_i = d_i^{\psi}$.
- Estimate the continuation value $\widehat{C}_i(s_{i,j}) = \widehat{b}_i^T \psi(s_{i,j})$ $(j = 1, \ldots, N)$.
- Compute $\widehat{W}_{i,j} = \max\{h_i(s_{i,j}), \widehat{C}_i(s_{i,j})\}$.

(4) Save the regression coefficients $\widehat{b}_i$ to compute the risky derivative value.

Remark 3.1: Regarding the use of least squares for regression and the choice of weighted Laguerre polynomials, note that this is the proposal appearing in the seminal paper [44] where the LSM was introduced. Furthermore, rigorous theoretical convergence has been also studied in [44], subsequently improved in [51]. Concerning possible alternative regression methods, non-parametric regression techniques (like in [18]) could produce more stable results, although at much more computational cost. With respect to the selection of the basis functions, firstly note that the coefficients of orthogonal polynomials give rise to a non-singular matrix so that the LSM estimates should be the same for all orthogonal families of polynomials (Lengendre, Hermite, Laguerre, etc.) and monomials. The observed differences in the results can only be explained by numerical inaccuracies coming from the regression algorithm [51]. In the context of American option pricing, the family of weighted Laguerre polynomial exhibits more precision (see, for example, the numerical studies in [3]).

3.2. SGBM: bundling-based approach

As a second alternative, we consider the novel Stochastic Grid Bundling Method (SGBM), proposed in 2015 by Jain and Oosterlee [39], which combines several advantageous characteristics and provides stable and accurate estimations. Two specific features are particularly interesting. On the one hand, SGBM divides the state space into partitions called *bundles*, where the quantities of interest are locally regressed. On the other hand, that regression is performed on the next time step variables, i.e. SGBM employs the so-called *regress-later* approach, contrary to LSM, which relies on the *regress-now* strategy. As it has been numerically proven in [37] (and the references therein), the regress-later scheme, in combination with bundling the space, presents a more precise and robust behaviour.

As in the case of LSM, the SGBM algorithm starts by computing the option value at terminal time and on every simulated path, i.e. $h(s_{M,j})$ for $j = 1, \ldots, N$. Next, the following SGBM components are performed for each time step, t_i ($i < M$), moving backwards in time, starting from t_M:

- *Bundling*

  The underlying asset prices at t_{i-1}, i.e. $s_{i-1,j}$, are *bundled* into non-overlapping sets or partitions, $\mathcal{B}_{i-1,1}, \ldots, \mathcal{B}_{i-1,\nu}$, by using some prescribed scheme (*k-means clustering*, *recursive bifurcation* or *equal-partitioning* are some examples; see further details in [21,39,43]).

  A mapping $\mathcal{I}^{\beta}_{i-1} : \mathbb{N}^{[1,N_\beta]} \mapsto \mathbb{N}^{[1,N]}$ is defined, which maps ordered indices of paths in a bundle $\mathcal{B}_{i-1,\beta}$ to the original path indices, where $N_\beta := |\mathcal{B}_{i-1,\beta}|$ is the cardinality of the β-th bundle, $\beta = 1, \ldots, \nu$.
- *Regress-later within each bundle*

  Corresponding to each bundle $\mathcal{B}_{i-1,\beta}$ ($\beta = 1, \ldots, \nu$), a parameterized value function $G : \mathbb{R} \times \mathbb{R}^K \mapsto \mathbb{R}$, which assigns values $G(s_{i,j}, a^{\beta}_i)$ to states $s_{i,j}$, is computed. Here $a^{\beta}_i \in \mathbb{R}^K$ is a vector of free parameters. Assuming the square-integrability of the option value function, it is well-known that, under certain conditions, the option value function can be expressed as a linear combination of a countable number of orthonormal basis functions, ϕ_k. The goal is therefore to choose, for each t_i and β, a parameter vector a^{β}_i so that

$$W_i(s_{i,j}) = G(s_{i,j}, a^{\beta}_i) = \sum_{k=1}^{\infty} a^{\beta}_{i,k} \phi_k(s_{i,j}).$$

  In practice, the orthonormal basis is restricted to a finite set, i.e. $k \leq K$, and thus one may choose $\widehat{G} \approx G$ such that,

$$W_i(s_{i,j}) \approx \widehat{G}(s_{i,j}, a^{\beta}_i) = \sum_{k=1}^{K} a^{\beta}_{i,k} \phi_k(s_{i,j}). \tag{11}$$

  The vector of coefficients $a^{\beta}_{i,k}$ can be then estimated by using an ordinary least-squares regression, i.e.

$$\text{argmin}_{\widehat{a}^{\beta}_i} \sum_{n=1}^{N_\beta} \left(W_i\left(s_{i,\mathcal{I}^{\beta}_{i-1}(n)}\right) - \sum_{k=1}^{K} \widehat{a}^{\beta}_{i,k} \phi_k\left(s_{i,\mathcal{I}^{\beta}_{i-1}(n)}\right) \right)^2, \tag{12}$$

  where $n = 1, \ldots, N_\beta$, is the index of each path in the bundle β. Given a finite number of sample paths used for the least squares regression, the coefficients $\widehat{a}^{\beta}_{i,k}$ represent the approximation of $a^{\beta}_{i,k}$.
- *Computing the continuation and option values*

  The continuation values for $s_{i-1,\mathcal{I}^{\beta}_{i-1}(n)} \in \mathcal{B}_{i-1,\beta}$, (for $n = 1, \ldots, N_\beta$, $\beta = 1, \ldots, \nu$) are approximated by

$$\widehat{C}_{i-1}\left(s_{i-1,\mathcal{I}^{\beta}_{i-1}(n)}\right) = \mathbb{E}_{t_{i-1}}\left[\widehat{G}\left(S_i, a^{\beta}_i\right) \mid S_{i-1} = s_{i-1,\mathcal{I}^{\beta}_{i-1}(n)}\right].$$

Exploiting the linearity of the expectation operator by means of (11), the previous value can be written as:

$$\widehat{C}_{i-1}\left(s_{i-1,\mathcal{I}_{i-1}^{\beta}(n)}\right)=\sum_{k=1}^{K}\widehat{a}_{i,k}^{\beta}\mathbb{E}_{t_{i-1}}\left[\phi_k(S_i)\mid S_{i-1}=s_{i-1,\mathcal{I}_{i-1}^{\beta}(n)}\right]. \tag{13}$$

The vector of basis functions ϕ should ideally be chosen such that the expectations $\mathbb{E}[\phi_k(S_i)|S_{i-1}]$ are known in closed-form, or have analytic approximations.

The option value at each exercise time is then given by:

$$\widehat{W}_i\left(s_{i-1,\mathcal{I}_{i-1}^{\beta}(n)}\right)=\max\left(h\left(s_{i-1,\mathcal{I}_{i-1}^{\beta}(n)}\right),\widehat{C}_{i-1}\left(s_{i-1,\mathcal{I}_{i-1}^{\beta}(n)}\right)\right).$$

Note that the description above corresponds to the so-called *direct estimator*, as defined in [39], which comes from the 'direct' computation of the exercise policy and the application of the dynamic programming formulation. In Algorithm 2, we present a schematic representation of the SGBM method.

Algorithm 2 SGBM

(1) Simulate N independent paths $\{s_{1,j}, s_{2,j}, \ldots, s_{M,j}\}$ $(j = 1, 2, \ldots, N)$ of the asset prices process.
(2) At maturity time t_M, $\widehat{W}_M(s_{M,j}) = h_M(s_{M,j})$.
(3) Apply backward induction for $i = M-1, \ldots, 1$.
- Define a mapping $\mathcal{I}_{i-1}^{\beta}$ to distribute the asset prices $s_{i-1,j}$ into bundles $\mathcal{B}_{i-1,\beta}$.
- Approximate the next step's risky option value $W_i(s_{i,j})$ for each bundle $\mathcal{B}_{i-1,\beta}$.
- Compute the coefficients $\widehat{a}_i$ as the solution of a ordinary least squares regression.
- Estimate the continuation value for $s_{i-1,\mathcal{I}_{i-1}^{\beta}(n)}$ at each bundle $\mathcal{B}_{i-1,\beta}$ by

$$\widehat{C}_i(s_{i-1,\mathcal{I}_{i-1}^{\beta}(n)})=\sum_{k=1}^{K}\widehat{a}_{i,k}^{\beta}\mathbb{E}_{t_{i-1}}\left[\phi_k(S_i)\mid S_{i-1}=s_{i-1,\mathcal{I}_{i-1}^{\beta}(n)}\right].$$

(4) Compute $\widehat{W}_i(s_{i-1,\mathcal{I}_{i-1}^{\beta}(n)}) = \max\{h(s_{i-1,\mathcal{I}_{i-1}^{\beta}(n)}), \widehat{C}_{i-1}(s_{i-1,\mathcal{I}_{i-1}^{\beta}(n)})\}$.

3.3. LSM vs. SGBM

We now present a set of numerical experiments aiming to compare the two methods presented above for the American option valuation including counterparty credit risk. At a first look, a couple of important differences between them immediately arise. On the one hand, SGBM performs the regression locally, i.e. within a predefined number of partitions or bundles of the underlying asset space, while LSM regresses the data points globally, without grouping them. On other hand, LSM employs the so-called *regress-now* strategy, contrary to SGBM, which utilizes the *regress-later* approach. As we will show, the later turns out to be more precise and stable although it involves extra mathematical and computational complexity, since an expectation on the basis functions is required. In any case, if the application of interest prevents the use of the regress-later alternative (for example when the expectation in (13) is hard to compute), SGBM can be readily transformed into a regress-now fashion.

In the experiments in this section (and in Section 4.3.1), we consider the problem of pricing an American put option whose payoff function is $H(S_t) = \max\{K - S_t, 0\}$ with the following configuration:

$$\begin{aligned} &r=0.04, \quad r_R=0.06, \quad \sigma=0.25, \quad T=1, \quad K=15,\\ &R_B=0.3, \quad R_C=0.3, \quad \lambda_B=0.04, \quad \lambda_C=0.04, \end{aligned} \tag{14}$$

As it is common practice, the American option price is approximated by its Bermudan counterpart with a sufficiently high number of exercise opportunities. In this case, we select $M = 200$. As an initial underlying price we will generally employ $S_0 = K = 15$, except when its value is concretely specified, like in Section 4.4.1. The reference 'true' value of the American option considering XVA is obtained by the PDE-based methodology proposed in [4], taking an extreme configuration of time and space discretizations.

In order to isolate the estimation error caused by the Monte Carlo-based algorithms, in this section the risk-free inner option price required for the mark-to-market value is computed by means of the celebrated *COS*[1] *method* [29], a well-known Fourier inversion technique which provides very high accuracy (close to *machine epsilon*) in 1D early exercise pricing problems.

Given each methods' particularities described above, several numerical test cases are now carried out, aiming to asses the impact of their specific features on the XVA derivatives pricing context. First, a study on the SGBM convergence in terms of bundles is performed, considering both regress-now and regress-later approaches. The obtained convergence patterns are depicted in Figure 1. We can clearly observe that, by increasing the number of bundles (for a fixed number of Monte Carlo simulations), the approximation provided by SGBM significantly improves, indicating that the localized regression is beneficial regardless the regression strategy. However, another interesting observation comes from the fact that the convergence pattern of SGBM using regress-now gets stabilized after reaching a particular point (around $\nu = 2^4$), while SGBM with regress-later convergence presents a faster decay pattern, i.e, the estimation accuracy keeps improving for increasing number of bundles. In addition, a sufficiently high number of bundles prevents from numerical instabilities arising around the exercise boundary or in the kink produced by the payoff function at the strike.

As a second experiment, we study the convergence and the variance of the considered methodologies in terms of the number of simulations. In order to approximate standard deviation (the square root of the variance), we generate 100 option values produced by each methodology, taking the average of them as the methods' estimate. The obtained results are presented in Figure 2. From this experiment, two immediate insights can be extracted to demonstrate the benefits of the regress-later strategy. Firstly, the techniques based on regress-now converge much slower than SGBM with regress-later, in line with the outcomes of the previous experiment. Secondly, the variability is drastically reduced when the regress-later approach is employed, being almost negligible from $N = 2^{12}$ on. In Table 1, the corresponding relative errors (denoted by 'RE') and the standard deviations (denoted by 'SD') are reported. Again, the impressive reduction in the estimation variability when using regress-later is confirmed. Additionally, we observe that SGBM with regress-now becomes more precise than

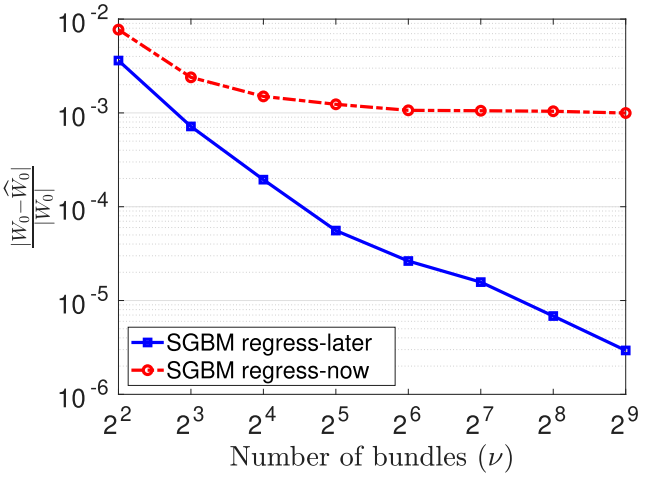

Figure 1. SGBM: convergence in bundles, ν. Setting: $N = 2^{18}$.

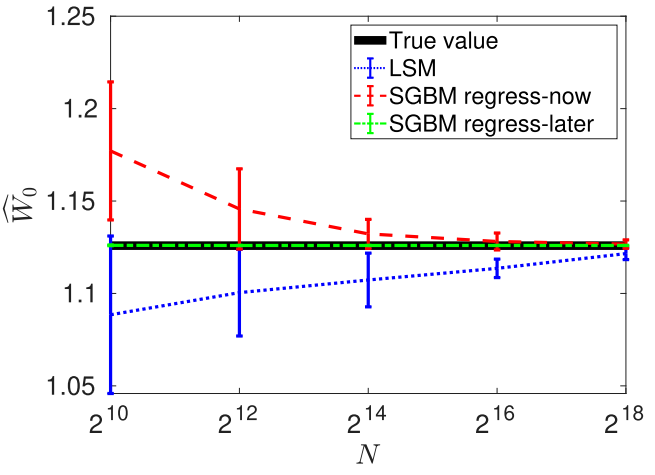

Figure 2. SGBM: convergence in Monte Carlo paths, *N*. Setting: $\nu = 512$.

Table 1. Relative errors and standard deviations.

	LSM		SGBM (regress-now)		SGBM (regress-later)	
N	RE	SD	RE	SD	RE	SD
2^{10}	3.32×10^{-2}	4.26×10^{-2}	4.55×10^{-2}	3.73×10^{-2}	1.46×10^{-4}	1.98×10^{-4}
2^{12}	2.25×10^{-2}	2.34×10^{-2}	1.76×10^{-2}	2.17×10^{-2}	7.92×10^{-5}	7.31×10^{-5}
2^{14}	1.65×10^{-2}	1.45×10^{-2}	5.65×10^{-3}	7.84×10^{-3}	8.15×10^{-5}	3.12×10^{-5}
2^{16}	1.09×10^{-2}	4.98×10^{-3}	1.99×10^{-3}	4.57×10^{-3}	8.28×10^{-5}	1.93×10^{-5}
2^{18}	3.82×10^{-3}	3.23×10^{-3}	7.52×10^{-4}	2.27×10^{-3}	8.68×10^{-5}	9.35×10^{-6}

Setting: $\nu = 512$.

LSM for sufficiently high *N*, probably thanks to the partition of the space into bundles and the accurate local regression inside them. As a final observation we also note that, as expected, LSM tends to underestimate the derivative price, due to its quasi-optimal exercise strategy (see Section 3.1). Several improvements to correct this behaviour have been proposed in the literature (relying on duality theory [36,49] or martingales [34], for example), although they are out of the scope of this work. Here we compare the basic implementations of both methods.

4. GPU enhancement

From the experiments in the previous section, we can clearly conclude that SGBM results in an advantageous estimator in terms of convergence speed and variance reduction with respect to the LSM alternative. For that reason, from now on we consider SGBM to be further optimized by including advanced computational components, like the parallel implementation on GPUs. The proposed computational improvements apply in the same manner with SGBM regardless the considered regression strategy.

Besides the benefits numerically demonstrated above, SGBM presents another desirable feature: it is fully parallelizable. It is well-known that Monte Carlo methods are embarrassingly parallel by definition. However, the Monte Carlo simulation is only the first stage in the early-exercise option pricing techniques relying on dynamic programming formulation (see Section 3). A backward stage, where the regression is performed, needs to be also addressed, in which a clear dependence between the computations in each time point appears. Thus, this stage is usually the most expensive in terms of computational cost, specially if many time steps are considered. Moreover, when the regression is conducted globally, like in the case of LSM, the parallelization of the backward stage becomes

highly non-trivial [8,30], due to the extra interdependence introduced among the data (asset simulations). In contrast, thanks to the structure of bundles, SGBM offers an opportunity of parallelization, since the regression is performed locally (within the partitions), and there is no dependence in the computational operations among different bundles. This remarkably important feature has already been exploited to address early-exercise pricing problems in a high-dimensional setting, see [43]. We will adapt some of the parallel formulations provided in that work to the current context of XVA calculations.

4.1. Parallel strategy

As previously mentioned, SGBM allows, thanks to its particular formulation, a complete parallel implementation. Actually, the two involved phases (the forward stage according the Monte Carlo simulation and the backward stage according to the dynamic programming solution) are suitable candidates for parallelization.

We therefore adopt the following parallel strategy. First, each Monte Carlo path is simulated in parallel, generating the underlying asset samples at each of the prescribed time steps. An estimation of the option value at the final time step (see (5)) is easily obtained by applying a parallel reduction. Secondly, we perform the SGBM components (described in Section 3.2) backwards in time, step by step, starting from the final time. Note that the involved computations, basically aiming to approximate the continuation value, are carried out at bundle level. Thus, the computations in each bundle can be done in parallel. Ideally, in order to avoid load-balancing problems, the number of asset samples within each bundle must be the same. For that reason, the *equal-partitioning* bundling technique is employed here (see [43] for further details). This procedure is repeated for all time steps following a recursive iteration. Thus, note that we actually perform M stages of parallelization, one per time step. However, thanks to an efficient GPU memory management, there is no need to perform neither any additional computation nor data transfers from GPU memory to CPU memory between the parallelization stages. Only data preparation (bundling) is executed here.

A schematic representation of the described parallel strategy is presented in Figure 3.

4.2. Computing the 'inner' option value

When addressing the option pricing problem for the case $\mathcal{M} = V$, i.e. when the mark–to–market is the value of the option without risk, the computational complexity becomes non-trivial, since the risk-free value has to be computed for each time step and for each Monte Carlo path. If this value is again calculated by Monte Carlo, we encounter a nested simulation, a non desirable component specially in

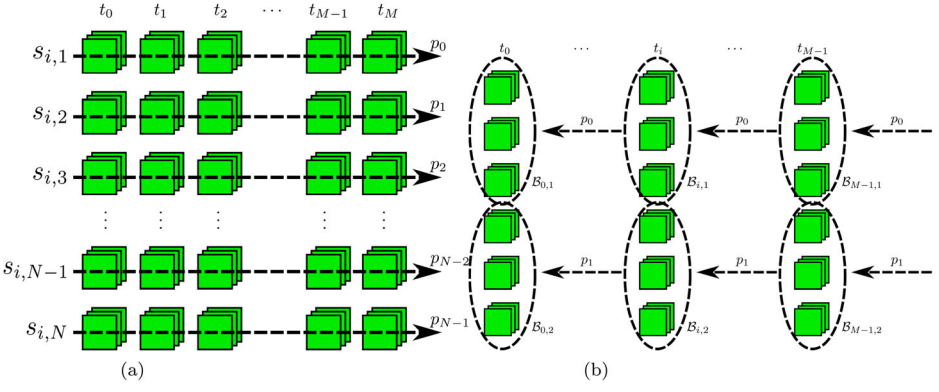

Figure 3. Parallel strategy. The GPU computation units (CUDA threads) are denoted by p_n. (a) Monte Carlo simulation (b) Bundle-based regression.

a GPU parallel implementation, because it entails an exponential growth in memory requirements. With the current underlying asset definition, many (faster) numerical techniques (Fourier inversion methods, as seen in previous sections) can be employed to obtain the option value; however, we aim to introduce a generically applicable approach, which must be also suitable to be run in parallel systems. Taking all these ideas into account, we propose the following alternative introduced in [5], based on interpolation:

(1) Given the initial spot prices, S_0, determine an asset domain $[S_{\min}, S_{\max}]$ sufficiently wide.
(2) Construct a vector of L potential 'initial' spot values, $S_0^1, S_0^2, \ldots, S_0^L$, within the previously defined asset domain.
(3) For each prescribed time point, t_i, compute the risk-free option values at that time, $V(t_i, s_0)$, with initial prices S_0^l, $l = 1, \ldots, L$, maturity $T = t_M$ and $M-i$ exercise opportunities. As a result, we obtain a grid (matrix) of option values with different initial (asset and time) points $[S_0^l, t_i]$, $l = 1, \ldots, L$, $i = 1, \ldots, M$.
(4) Pass these grid (matrix) to the risky option pricing engine.
(5) When required, compute the risk-free option price by interpolating over the constructed grid.

A schematic representation of this approach is depicted in Figure 4. The orange dashed vertical lines represent the interpolation direction. Each black dot represents the risk-free option price at particular time point, t_i, and with a specific initial asset value S_0^l.

This approach presents several advantages. On the one hand, we decouple the computation of the risky and risk-free option prices. We can therefore compute the grid points offline, before initiating the main algorithm for pricing the risky derivative value. Moreover, the computation of each grid point can be done in parallel, since there is no dependence between them. On the other hand, as the pricing routines are executed in a controlled number of points, important savings in memory requirements are obtained, a crucial aspect when considering GPU-based parallel systems. Furthermore, the interpolation (and, in particular, the linear interpolations used in this work) operates point-wise and can be done very efficiently, reducing to the minimum the computational overload due to the risk-free option pricing. This idea can be extended to multiple dimensions and/or to a range of model parameters by enlarging the interpolation space (considering a hypercube). Also, it can be straightforwardly adapted for the computation of the Greeks if required, following the algorithms developed in [37].

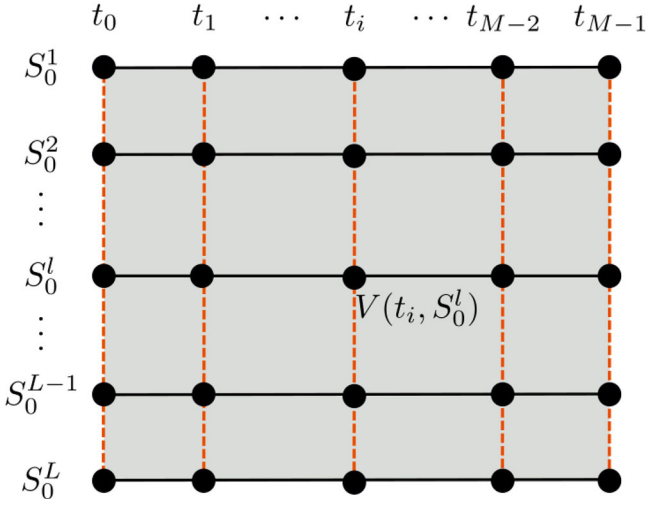

Figure 4. Schematic structure for the grid of option values.

Further, the collocation of the interpolation points can be selected following a previously analysed convenient pattern. This improvements are left for future research.

4.3. GPU implementation details

This section is devoted to briefly summarize some important implementation details for the proposed GPU-based parallel solution, i.e. the parallel version of SGBM for pricing early-exercise options including counterparty credit risk. We employ the well-known CUDA (Compute Unified Device Architecture) environment, see [27], suitable for NVIDIA GPUs. Note that the technicalities described here can be straightforwardly adapted to other similar platforms. Further details on CUDA can be found in excellent monographs like [22,26,41], for example.

CUDA relies on the computing paradigm called *Single Instruction, Multiple Data* (SIMD), whose basic operational units are the *CUDA threads*. The threads in CUDA are organized in several levels of hierarchy, where they are grouped into equal-size *blocks*, and the latter are grouped into the so-called *grids*. Such organization allows to construct a computing structure adapted to the problem at hand. Once this execution hierarchy is defined, the so-called *kernels* functions are run on the grid of CUDA threads, which execute the same instructions in parallel.

Another important issue to account for is the memory transfers. As mentioned, the GPU is often seen as a co-processor where the parallel instructions are executed on the same data. That data needs to be moved to the GPU memory. These memory transfers can potentially entail a significant impact in the algorithm's performance since, when the amount of data is considerable, the transfer penalizes the parallel execution. Thus, in any GPU-based parallel implementation is crucial to minimize/avoid the number of transfers and/or reduce the transferred amount of data.

Next, in the following subsections, we provide the particular features of the GPU implementation of the SGBM algorithm for option pricing with XVA.

4.3.1. Parallel Monte Carlo paths

Monte Carlo methods rely on the simulation of a large number of scenarios or paths. In the current models of NVIDIA GPUs, a huge number of CUDA threads can be handled, allowing to manage each Monte Carlo scenario by one single thread. The random numbers required in the algorithm are generated 'on the fly', by employing the cuRAND library [28]. We take advantage of these highly fine parallelization to compute all the necessary intermediate results, namely the intrinsic value of the option and the computation of the expectation in (13). Doing so, we make a more efficient use of the GPU memory hierarchy since those results can be stored in the memory with a fastest access, i.e. the registers, speeding up the overall computation. Further, we also perform calculations for the sorting criterion, required for the equal-partitioning bundling technique, inside the Monte Carlo generator, avoiding to move data from GPU global memory to CPU main memory. In this way, our approach is completely free of transfers, a fact that, as mentioned, gives us a remarkable performance improvement. All of these extra computations performed within the Monte Carlo algorithm are represented by including several green squares in Figure 3.

4.3.2. Parallel bundling

The bundling technique employed in this work, i.e, the equal-partitioning combines two operations: sorting and splitting. In the last decades, a new trend of sorting algorithms suitable for parallel systems have appeared in the literature as, for example, [40,50]. Along with these theoretical advances, a number of GPU-based sorting libraries have been released, with Thrust [52], CUB [25] or ModernGPU [45] being some of the most representative. In this work, we employ the CUB library, which provides an impressive performance since it is a close-to-production library. CUB is a library of collective primitives which are relevant building blocks for a wide variety of data-parallel algorithms, including sorting, at different GPU levels. Among the different sorting methods implemented in the library, we consider the parallel version of the Radix sort, which has demonstrated a remarkable efficiency

in parallel hardware. In addition, CUB includes a kernel-level Application Programming Interface (API) that, again, allows us to avoid all the transfers between parallelization phases.

Once the sorting stage has been completed, the splitting can be readily conducted. First, note that the bundle's size is trivially determined as $N_\beta = N/\nu, \forall \beta$. Then, each of the launched CUDA threads handles a pointer which references the beginning of the GPU global memory space where the corresponding Monte Carlo points are allocated. Thus, after the sorting stage, this approach entails a significantly more efficient memory access patterns since each bundle's memory areas are adjacent or, in GPU computing jargon, *coalesced*.

4.3.3. Parallel estimation

Following the dynamic programming approach, the exercise policy and the continuation (and option) values need to be computed for each bundle. For this purpose, we perform a parallelization at bundle level, namely the required computations within a bundle are managed by one CUDA thread. Then, the regression and intrinsic option value calculations are distributed among the available computational resources of the GPU, taking advantage of the pre-computed quantities from the Monte Carlo generator (see the subsection above). Thus, the presented algorithm entails M stages of parallelization, one every time step. In the final step, a reduction (an average) needs to be carried out to determine the actual option price. For that we utilize the well-established Thrust library [52], which does this task very efficiently in parallel on the GPU.

4.4. Computational performance

In this section the computational gains provided by the SGBM's parallel implementation are assessed throughout a set of numerical experiments. These experiments were carried out in a computational system with the following characteristics: CPU Intel Xeon E5-2620 v2 at 2.10 GHz, RAM 16 GB and GPU Nvidia Tesla V100 (Volta architecture). The codes were implemented in C language (sequential version) and CUDA (parallel version).

In all the experiments of this section, we again consider the American option valuation including XVA, with the parameter configuration in (14) and the settings described below it.

4.4.1. Convergence in grid points

This first experiment intents to numerically prove the convergence of the risky option price in terms of the number of grid points, L, employed for the interpolation-based approach presented in Section 4.2. Of course, this methodology introduces an interpolation error, which can be mitigated by including more points in the grid. We aim to determine an order of magnitude of this quantity and the impact to the targeted option price, i.e, the risky value of an American put option. Besides, as the grid construction is the most expensive part, it would be convenient to design it such that it can be re-utilized in most of the contexts. We therefore select a relatively wide range for the initial underlying prices, namely, from $S_0^1 = 5$ to $S_0^L = 25$, taking into account that the 'current' spot price is $S_0 = 15$. Then, to test the robustness of the proposed interpolation technique, three different spot prices are considered, $S_0 = 14$ (in-the-money), $S_0 = 15$ (at-the-money) and $S_0 = 16$ (out-of-the-money). In order to minimize the contribution of any other source of error, both the number of simulations and the number of bundles are taken sufficiently high, according to the results in Section 3.3, as $N = 2^{18}$ and $\nu = 512$, respectively. The obtained convergence results are presented in Table 2. We can clearly observe that, by just taking $L = 16$ points, a remarkable accuracy is achieved (around $10^{-4} \sim 10^{-5}$), provided that the SGBM estimate converges at the well-established (see [10], for example) Monte Carlo rate $N^{-\frac{1}{2}}$ (see Table 1), i.e. the precision of the approximation is bounded by the Monte Carlo theoretical convergence. Thus, we observe a fast numerical convergence (one order of magnitude) for lower values of L, which gets stabilized when the Monte Carlo precision is achieved. Moreover, it is worth to emphasize that this precision is preserved along the different spot prices, which allows to employ the same grid for multiple option pricing problems.

Table 2. Convergence (measured as the relative error of risky option values) in grid points, L.

	$S_0 = 14$	$S_0 = 15$	$S_0 = 16$
$L = 4$	5.88×10^{-3}	9.19×10^{-3}	1.24×10^{-2}
$L = 8$	7.59×10^{-4}	1.31×10^{-3}	1.82×10^{-3}
$L = 16$	8.92×10^{-5}	2.59×10^{-4}	4.39×10^{-4}
$L = 32$	5.74×10^{-5}	4.86×10^{-5}	1.37×10^{-4}
$L = 64$	3.92×10^{-5}	1.71×10^{-5}	9.16×10^{-5}

Setting: $N = 2^{18}$ and $\nu = 512$.

4.4.2. Performance analysis I: parts of the algorithm

The next set of experiments aim to compare the performance of a sequential version and a parallel version of SGBM, running on a CPU and a GPU, respectively. As described in Section 4.1, we exploit two opportunities of parallelization, the paths simulation required for the Monte Carlo algorithm in the forward stage and the regression computation within bundles at each time step in the backwards stage. Additionally, SGBM is also employed to construct the grid of option values without considering XVA, as proposed in Section 4.2, for which the risk-free original implementations (sequential and parallel) of the method are considered (see [43], for details).

Thus, the first test consists of assessing the individual contribution of each part of the algorithm and the (potential) gain provided by the parallel version compared against the sequential version. We denote by 'VG' the time consumed in the construction of the risk-free option values grid, by 'MC' the Monte Carlo simulation and by 'DE' the so-called *direct estimator* of SGBM, which corresponds to the regression computations within bundles and backwards in time. In order to offer a complete picture of the impact of the GPU-based parallel enhancement, several choices for the number of simulations, N, and the number of bundles, ν, are included in the analysis, providing this way a full range of combinations.

From the execution times reported in Tables 3 and 4, the following interesting observations can be extracted:

- As expected, the Monte Carlo algorithm greatly exploits the parallelism, obtaining an impressive reduction in the computational time (around $\times 1800$ in the best case). Besides the gain in speedup, the execution time magnitude is also remarkable, being in the order of milliseconds.
- The number of bundles have a crucial impact in the efficiency of the GPU parallel version. This fact is intuitive since the GPU parallelism needs a sufficiently high number of parallel threads to be competitive and take the parallel potential to the maximum. If we consider a low number of bundles, the performance of the GPU version is hampered. In any case, from $\nu = 32$ on, the parallel SGBM begins to provide important gains, specially in creating the grid option values, where the SGBM's backward stage dominates the computational cost.

Table 3. Parts of the algorithm: time (in seconds) and speedup.

	$\nu = 8$			$\nu = 32$		
	VG	MC	DE	VG	MC	DE
CPU	13560.6	2.80	1.61	13583.4	2.77	1.57
GPU	5399.9	8.77×10^{-3}	3.48	1376.7	8.83×10^{-3}	1.11
Speedup	2.51	319.27	0.46	9.86	313.70	1.41
		$\nu = 128$			$\nu = 512$	
	VG	MC	DE	VG	MC	DE
CPU	13646.2	2.77	1.60	13659.6	2.78	1.60
GPU	565.8	8.98×10^{-3}	0.36	535.4	8.81×10^{-3}	0.33
Speedup	24.12	308.46	4.44	25.5	315.55	4.84

Setting: $N = 2^{14}$.

Table 4. Parts of the algorithm: time (in seconds) and speedup.

	$\nu = 8$			$\nu = 32$		
	VG	MC	DE	VG	MC	DE
CPU	58834.1	11.65	8.66	57055.9	11.04	7.58
GPU	21726.2	6.69×10^{-3}	13.89	14124.8	6.02×10^{-3}	9.24
Speedup	2.71	1741.40	0.62	4.04	1833.88	0.82
	$\nu = 128$			$\nu = 512$		
	VG	MC	DE	VG	MC	DE
CPU	56346.8	10.88	7.31	56927.9	11.04	7.31
GPU	1857.7	5.94×10^{-3}	1.19	761.3	5.97×10^{-3}	0.47
Speedup	30.33	1831.65	6.14	74.78	1849.25	15.55

Setting: $N = 2^{16}$.

- Note as well that the number of bundles hardly affects the execution time of sequential version. This is because, given a specific number of Monte Carlo paths, the same workload needs to be processed regardless how those paths are distributed among the bundles.
- The performance improvement in the option value grid construction makes its use in practical applications feasible since the execution time is reduced from around 16 hours to around 10 minutes (for $N = 2^{16}$). Note that a process running for more than 10 hours (approx.) cannot be addressed overnight.
- Further, the significant reduction of the execution times of the value grid construction could also allow to explore more general grids, including other market or model parameters.
- In general, the more extreme is the configuration in number of simulations, N, and the number of bundles, ν, the greater benefits are provided by the GPU-based implementation. Thus, the gains are expected to be more important when one or both parameters are further increased.

Note that the performance of both, the parallel GPU-based and the sequential implementations can be further improved by a multi-core approach, using *OpenMP* for example. Here, we do not address such implementation because it would not provide any extra information on the computational efficiency since its impact would be proportional to the number of cores.

4.4.3. Performance analysis II: pricing for several spot prices

As shown in the numerical test in Section 4.4.1, once the option value grid is constructed, it can be safely employed for different initial spot prices, S_0, allowing this way the valuation of several market scenarios. In this experiment, we will assess the computational performance of the parallel version of SGBM in pricing various risky American puts with spot price ranging from $S_0 = 14$ to $S_0 = 16$. Besides, both the number of simulations, N, and the number of bundles, ν, are taken 'to the limit' aiming to highlight that, thanks to our GPU-based implementation, the order of the computational cost is kept in a reasonable and practically appealing magnitude. In Table 5, the execution times for pricing ten American put options are reported. Note that, although a bit counter-intuitive, increasing the number of bundles have a positive impact (i.e. the running time is reduced), particularly when the number of simulations is sufficiently high. Again, this is related to the fact of taking advantage of the full parallel power of the GPU computing, for what we should try to distribute the workload among

Table 5. Pricing ten American options: times (in seconds).

	$\nu = 512$	$\nu = 1024$	$\nu = 2048$	$\nu = 4096$
$N = 2^{16}$	4.86	4.82	5.31	6.97
$N = 2^{18}$	20.10	13.65	9.88	8.63
$N = 2^{20}$	87.78	58.91	46.96	42.82

Setting: $L = 16$.

Table 6. Pricing problem taking $\mathcal{M} = W$: time (in seconds) and speedup.

	$\nu = 8$			$\nu = 32$		
	$N = 2^{16}$	$N = 2^{18}$	$N = 2^{20}$	$N = 2^{16}$	$N = 2^{18}$	$N = 2^{20}$
CPU	30.29	130.72	571.57	29.99	131.14	551.17
GPU	40.59	236.22	962.93	11.78	67.79	280.02
Speedup	0.75	0.55	0.59	2.54	1.93	1.97
		$\nu = 128$			$\nu = 512$	
	$N = 2^{16}$	$N = 2^{18}$	$N = 2^{20}$	$N = 2^{16}$	$N = 2^{18}$	$N = 2^{20}$
CPU	29.43	128.69	545.83	29.16	127.05	538.53
GPU	3.32	18.09	73.77	1.16	5.33	21.59
Speedup	8.86	7.11	7.40	25.14	23.84	24.94
		$\nu = 1024$			$\nu = 4096$	
	$N = 2^{16}$	$N = 2^{18}$	$N = 2^{20}$	$N = 2^{16}$	$N = 2^{18}$	$N = 2^{20}$
CPU	29.27	126.54	534.40	29.51	125.98	521.28
GPU	1.12	3.36	13.13	1.88	2.12	8.75
Speedup	26.13	37.66	40.70	15.69	59.42	59.57

the maximum possible GPU threads (which match ν in this case). It is also worth to mention that the computational gains get stabilized from $\nu = 2048$. This might suggest that an optimal balance between the number of simulations, N, and number of bundles, ν, can be achieved but, it would be problem-dependent or determined in terms of the prescribed accuracy.

4.4.4. Performance analysis III: the non linear case $\mathcal{M} = W$

In the last experiment we study the performance of the parallel SGBM engine addressing the non-linear case, which arises when the mark-to-market is equal to the risky option price, i.e. $\mathcal{M} = W$. As previously mentioned, the non-linearity is handled by including a fixed-point scheme in the computation of the continuation value. The total execution times and the acceleration ratio provided by the parallel SGBM are reported in Table 6. Again, we observe a remarkable reduction of the computational cost thanks to the GPU-based parallelization. In particular, for practical[2] values of bundles' dimension, i.e. $\nu \geq 128$ and number of Monte Carlo paths, the obtained speedup ranges from around 8 times up to 60 times, approximately.

Remark 4.1: For the sake of conciseness, the obtained accuracy and convergence pattern in terms of bundles for the case $\mathcal{M} = W$ are not reported here. The precision provided by the SGBM method is of the same order as that reported in Section 3.3 for the case $\mathcal{M} = W$. For LSM, we refer to [5] for further details.

5. Conclusions

In this paper we have proposed the use of SGBM to price early-exercise options in the presence of counterparty credit risk. We have compared this technique with the classical LSM approach, highlighting the positive impact of the specific features of SGBM, namely the regress-later scheme (contrary to the regress-now alternative employed by LSM) and the space partition thanks to the bundling structure. Additionally, taking advantage of such organization in bundles, an efficient parallel version of the methodology is presented, which is particularly implemented on a GPU-based computational system. Moreover, a novel interpolation-based algorithm is developed to avoid the undesirable nested simulations usually appearing in XVA calculations. Furthermore, this interpolation technique has been also designed to be suitable for parallelization. As a result, we have come up with a fully parallelizable method, from the preprocesing stage to compute inner option values, passing through the Monte Carlo simulation, up to the risky option price estimation. This fact provides an impressive gain (speedup) in performance with respect to the sequential (CPU-based) implementations, supported here by a wide range of computational experiments.

Notes

1. The abbreviation derives from the fact that the method employs cosine series expansions to estimate the underlying density function.
2. Values which ensure a sufficiently high precision

Disclosure statement

No potential conflict of interest was reported by the author(s).

Funding

Í. Arregui, Á. Leitao, B. Salvador and C. Vázquez were funded by Spanish Ministry of Science and Innovation with [grant number PID2019-108584RB-I00]. Á. Leitao acknowledges the financial support from the Spanish Ministry of Science and Innovation, through the Juan de la Cierva-formación 2017 (FJC17) grant in the framework of the national programme for R&D 2013–2016. The authors wish to acknowledge the support received from the Centro de Investigación en Tecnologías de la Información y las Comunicaciones de Galicia, CITIC, funded by Xunta de Galicia and the European Union (European Regional Development Fund, Galicia 2014–2020 Program) by [grant number ED431G 2019/01]. Also the authors acknowledge the funding by Xunta de Galicia through [grant numbers ED431C2018/033 and ED431C 2022/47].

ORCID

Íñigo Arregui http://orcid.org/0000-0002-2456-4092
Álvaro Leitao http://orcid.org/0000-0002-3442-4587
Beatriz Salvador http://orcid.org/0000-0003-1443-6799
Carlos Vázquez http://orcid.org/0000-0001-6591-2252

References

[1] L.A. Abbas-Turki, S. Crépey, and B. Diallo, *XVA principles, nested Monte Carlo strategies, and GPU optimizations*, Int. J. Theoretical Appl. Finance 21 (2018), p. 1850030. https://doi.org/10.1142/S0219024918500309.
[2] C. Albanese, S. Caenazzo, and S. Crépey, *Credit, funding, margin, and capital valuation adjustments for bilateral portfolios*, Probability, Uncertainty Quantitative Risk 2 (2017), p. 7.
[3] N. Areal, A. Rodrigues, and M. Armada, *On improving the least squares monte carlo option valuation method*, Rev. Derivatives Res. 11 (2008), pp. 119–151.
[4] I. Arregui, B. Salvador, and C. Vázquez, *PDE models and numerical methods for total value adjustment in European and American options with counterparty risk*, Appl. Math. Comput. 308 (2017), pp. 31–53.
[5] I. Arregui, B. Salvador, and C. Vázquez, *A Monte Carlo approach to American options pricing including counterparty risk*, Int. J. Comput. Math. 96 (2019), pp. 2157–2176.
[6] I. Arregui, B. Salvador, D. Ševčovič, and C. Vázquez, *Total value adjustment for European options with two stochastic factors. Mathematical model, analysis and numerical simulation*, Computers Math. Appl. 76 (2018), pp. 725–740.
[7] I. Arregui, B. Salvador, D. Ševčovič, and C. Vázquez, *PDE models for American options with counterparty risk and two stochastic factors: Mathematical analysis and numerical solution*, Computers Math. Appl. 79 (2020), pp. 1525–1542. https://doi.org/10.1016/j.camwa.2019.09.014
[8] M. Benguigui and F. Baude, *Fast American basket option pricing on a multi-GPU cluster*, in *Proceedings of the 22nd High Performance Computing Symposium*, Tampa, FL, 2014, pp. 1–8.
[9] A. Borovykh, C.W. Oosterlee, and A. Pascucci, *Efficient XVA computation under local Lévy models*, SIAM J. Financial Math. 9 (2018), pp. 251–273.
[10] P. Boyle, M. Broadie, and P. Glasserman, *Monte Carlo methods for security pricing*, J. Economic Dyn. Control 21 (1997), pp. 1267–1321.
[11] D. Brigo and A. Capponi, *Bilateral counterparty risk valuation with stochastic dynamical models and applications to CDSs*, preprint (2009). Available at arXiv, math. 0812.3705.
[12] D. Brigo, M. Morini, and A. Pallavicini, *Counterparty Credit Risk, Collateral and Funding*, Wiley, 2013.
[13] C. Burgard and M. Kjaer, *In the balance*, Risk (2011), pp. 72–75. https://dx.doi.org/10.2139/ssrn.1785262.
[14] C. Burgard and M. Kjaer, *PDE representations of derivatives with bilateral counterparty risk and funding costs*, J. Credit Risk 7 (2011), pp. 1–19.

[15] C. Burgard and M. Kjaer, *Funding strategies, funding costs*, Risk (2013), pp. 82–87.
[16] L. Capriotti, Y. Jiang, and A. Macrina, *AAD and least-square Monte Carlo: Fast Bermudan-style options and XVA Greeks*, Algorithmic Finance 6 (2017), pp. 35–49.
[17] L. Capriotti, S. Lee, and J.M. Peacock, *Real time counterparty credit risk management in Monte Carlo*, Risk (2011), pp. 1–7.
[18] J.F. Carriere, *Valuation of the early-exercise price for options using simulations and nonparametric regression*, Insurance: Math. Econ. 19 (1996), pp. 19–30. https://www.sciencedirect.com/science/article/pii/S0167668796000042.
[19] K.W. Chau, J. Tang, and C.W. Oosterlee, *An SGBM-XVA demonstrator: A scalable Python tool for pricing XVA*, J. Math. Industry 10(7) (2020), pp. 1–19.
[20] Y. Chen and C.C. Christara, *Penalty methods for bilateral XVA pricing in European and American contingent claims by a partial differential equation model*, J. Comput. Finance 24 (2021), pp. 41–70.
[21] F. Cong and C.W. Oosterlee, *Pricing Bermudan options under Merton jump-diffusion asset dynamics*, Int. J. Comput. Math. 92 (2015), pp. 2406–2432.
[22] S. Cook, *CUDA Programming: A Developer's Guide to Parallel Computing with GPUs*, 1st ed., Morgan Kaufmann Publishers Inc., San Francisco, 2013.
[23] S. Crépey, *Bilateral counterparty risk under funding constraints–part I: Pricing*, Math. Finance 25 (2015), pp. 1–22.
[24] S. Crépey, *Bilateral counterparty risk under funding constraints–part II: CVA*, Math. Finance 25 (2015), pp. 23–50.
[25] *CUB webpage*. Available at https://nvlabs.github.io/cub/.
[26] *CUDA programming guide*. Available at https://docs.nvidia.com/cuda/cuda-c-programming-guide/.
[27] *CUDA webpage*. Available at https://developer.nvidia.com/cuda-zone.
[28] *cuRAND webpage*. Available at https://developer.nvidia.com/curand.
[29] F. Fang and C.W. Oosterlee, *Pricing early-exercise and discrete barrier options by Fourier-cosine series expansions*, Numerische Mathematik 114 (2009), pp. 27–62.
[30] M. Fatica and E. Phillips, *Pricing American options with least squares Monte Carlo on GPUs*, in *Proceedings of the 6th Workshop on High Performance Computational Finance, WHPCF '13*, ACM, New York, NY, 2013, pp. 1–6. http://doi.acm.org/10.1145/2535557.2535564.
[31] Q. Feng and C.W. Oosterlee, *Computing credit valuation adjustment for Bermudan options with wrong way risk*, Int. J. Theoretical Appl. Finance 20 (2017), p. 1750056.
[32] J. Fernández, A. Ferreiro, J. García-Rodríguez, A. Leitao, J. López-Salas, and C. Vázquez, *Static and dynamic SABR stochastic volatility models: Calibration and option pricing using GPUs*, Math. Comput. Simul. 94 (2013), pp. 55–75. https://www.sciencedirect.com/science/article/pii/S0378475413001389.
[33] A.M. Ferreiro-Ferreiro, J.A. García-Rodríguez, L. Souto, and C. Vázquez, *A new calibration of the Heston stochastic local volatility model and its parallel implementation on GPUs*, Math. Comput. Simul. 177 (2020), pp. 467–486. https://www.sciencedirect.com/science/article/pii/S0378475420301129.
[34] P. Glasserman, *Monte Carlo Methods in Financial Engineering*, Springer, 2003.
[35] P. Glasserman and B. Yu, *Simulation for American options: Regression now or regression later*? in *Monte Carlo and Quasi-Monte Carlo Methods 2002*, Springer, 2004, pp. 213–226.
[36] M.B. Haugh and L. Kogan, *Pricing American options: A duality approach*, Oper. Res. 52 (2004), pp. 258–270. http://www.jstor.org/stable/30036577.
[37] S. Jain, Á. Leitao, and C.W. Oosterlee, *Rolling adjoints: Fast greeks along Monte Carlo scenarios for early-exercise options*, J. Comput. Sci. 33 (2019), pp. 95–112. http://www.sciencedirect.com/science/article/pii/S1877750318312547.
[38] S. Jain and C.W. Oosterlee, *Pricing high-dimensional Bermudan options using the stochastic grid method*, Int. J. Comput. Math. 89 (2012), pp. 1186–1211. https://doi.org/10.1080/00207160.2012.690035.
[39] S. Jain and C.W. Oosterlee, *The stochastic grid bundling method: Efficient pricing of Bermudan options and their Greeks*, Appl. Math. Comput. 269 (2015), pp. 412–431.
[40] B. Jan, B. Montrucchio, C. Ragusa, F.G. Khan, and O. Khan, *Fast parallel sorting algorithms on GPUs*, Int. J. Distributed Parallel Syst. 3 (2012), pp. 107–118.
[41] D.B. Kirk and W.W. Hwu, *Programming Massively Parallel Processors: A Hands-On Approach*, 3rd ed., Elsevier, 2017.
[42] P.E. Kloeden and E. Platen, *Numerical Solution of Stochastic Differential Equations*, Springer, 1992.
[43] A. Leitao and C.W. Oosterlee, *GPU acceleration of the stochastic grid bundling method for early-exercise options*, Int. J. Comput. Math. 92 (2015), pp. 2433–2454.
[44] F.A. Longstaff and E.S. Schwartz, *Valuing American options by simulation: A simple least-squares approach*, Rev. Financial Stud. 14 (2001), pp. 113–147.
[45] *Modern GPU webpage*. Available at https://moderngpu.github.io/.
[46] A. Pallavicini, D. Perini, and D. Brigo, *Funding valuation adjustment: A consistent framework including CVA, DVA, collateral, netting rules and re-hypotecation*, SSRN Electronic Journal (2011), pp. 1–23.
[47] A. Pascucci, *PDE and Martingale Methods in Option Pricing*, Springer, 2011.

[48] V. Piterbarg, *Funding beyond discounting: Collateral agreements and derivatives pricing*, Risk 23 (2010), pp. 97–102.
[49] L.C.G. Rogers, *Monte Carlo valuation of American options*, Math. Finance 12 (2002), pp. 271–286.
[50] N. Satish, M. Harris, and M. Garland, *Designing efficient sorting algorithms for manycore GPUs*, in *Proceedings of the 23rd IEEE International Parallel and Distributed Processing Symposium*, 2009.
[51] L. Stentoft, *Convergence of the least squares monte carlo approach to American option valuation*, Manage. Sci. 50 (2004), pp. 1193–1203. https://doi.org/10.1287/mnsc.1030.0155.
[52] *Thrust webpage*. Available at https://developer.nvidia.com/thrust.