Journal of Mathematics in Industry
a SpringerOpen Journal

RESEARCH Open Access

Multistep schemes for solving backward stochastic differential equations on GPU

Lorenc Kapllani[1*] and Long Teng[1]

*Correspondence:
kapllani@math.uni-wuppertal.de
[1]Bergische Universität Wuppertal, Gaußstr. 20, 42119 Wuppertal, Germany

Abstract

The Backward Stochastic Differential Equation (BSDE) is an important tool for pricing and hedging. Highly accurate pricing for low computation time becomes interesting for minimizing monetary loss. Therefore, we explore the opportunity of parallelizing high-order multistep schemes in option pricing. In the multistep scheme the computations at each space grid point are independent and this fact motivates us to select massively parallel GPU computing using CUDA. In our investigations we identify performance bottlenecks and apply appropriate optimization techniques to reduce the computation time in a uniform space domain. Runtime experiments manifest optimistic speedups for the parallel implementation on a single GPU, NVIDIA GeForce 1070 Ti.

Keywords: Backward stochastic differential equations; Multistep scheme; GPU computing; CUDA; Option pricing

1 Introduction

The backward stochastic differential equations (BSDEs) have been widely used in various areas such as physics and finance due to one of their key features, namely they provide a probabilistic representation of solutions of nonlinear parabolic partial differential equations (PDEs). We consider a (decoupled) forward backward stochastic differential equation (FBSDE) which has the form:

$$\begin{cases} dX_t = a(t, X_t)\,dt + b(t, X_t)\,dW_t, \quad X_0 = x_0, \\ -dy_t = f(t, X_t, y_t, z_t)\,dt - z_t\,dW_t, \\ y_T = \xi = g(X_T), \end{cases} \tag{1}$$

where X_t, $a \in \mathbb{R}^n$, b is a $n \times d$ matrix, W_t is a d-dimensional Brownian motion, $f(t, X_t, y_t, z_t) : [0, T] \times \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^{m\times d} \to \mathbb{R}^m$ is the driver function and ξ is the terminal condition. For $a = 0$ and $b = 1$, namely $X_t = W_t$, one obtains the standard BSDE of the form

$$\begin{cases} -dy_t = f(t, y_t, z_t)\,dt - z_t\,dW_t, \\ y_T = \xi = g(W_T), \end{cases} \tag{2}$$

where $y_t \in \mathbb{R}^m$ and $f(t, y_t, z_t) : [0, T] \times \mathbb{R}^m \times \mathbb{R}^{m\times d} \to \mathbb{R}^m$. The existence and uniqueness of the solution of (1) are proven in [20]. If dX_t in (1) is defined as the geometric brownian motion for modelling the asset price, and g is the payoff function of an European option, then (1) is related to the Black-Scholes PDE, see [14]. This is to say that y_0 gives the option price and z_t/b represents the hegding portfolio, which presents the sensitivity of the option price y_t with respect to the asset price X_t. This is called also Δ-hedging in the pricing theory.

Generally, it is rarely possible to find an analytical solution to a BSDE. Recently, many numerical methods have been thus proposed. For the numerical methods on spatial grids we refer to, e.g., [29–32]. The approaches based on the Fourier method for BSDEs are developed in [24, 25]. For probabilistic methods, Monte-Carlo approaches are investigated, e.g., in [10, 16], and tree-based methods in [6, 26]. And many others, see [1, 2, 4, 8, 9, 17].

Higher order numerical schemes demand usually more computing efforts, effective parallel implementations are thus in great demand. Some acceleration strategies based on Graphics Processing Unit (GPU) computing have been developed for the pricing problems in finance, however, a very little of them are BSDE-based approach. These works can be found in [7, 11, 22], where the acceleration strategies are applied on numerical methods of convergence order not higher than 2.

For higher order of convergence rate, the first multistep scheme on time-space grids is proposed in [32], where the resulting integrands by discretizing BSDE in time are approximated by using Lagrange interpolating polynomials. In [13], we have successfully parallelized that multistep scheme on GPUs, and showed the gain in computational time for option pricing via the Black-Scholes BSDE. However, the multistep scheme is only stable up to 3 multiple time levels due to Runge's phenomenon. For a better stability and the admission of more time levels, a new multistep scheme is proposed in [27] by using spline instead of Lagrange interpolating polynomials. In principle, arbitrarily many multilevel time levels can be chosen in that multistep scheme, and in general the more time levels the higher accuracy. However, using more time levels also requires more computational cost. For this reason, in this work we investigate the massively parallel GPU computing in the multistep scheme [27], to make the scheme be more useful in practice. For example, a high accuracy for low computation time can minimize monetary loss in financial problems. An application that shows the importance of efficient approaches for pricing and the risk management of financial models can be found in [3].

The reminder of this paper is organized as following. In Sect. 2, we introduce the multistep method [27] for the numerical solution of BSDEs. Section 3 presents the algorithmic framework of the numerical method and the potential for preliminary reduction of computing time due to its special structure for uniform domains. In Sect. 4, we describe the GPU acceleration strategies for further reduction of computing time. Section 5 shows the numerical results including financial applications. Finally, we give the conclusions in Sect. 6.

2 The multistep scheme

In this section we introduce the multistep scheme [27] orientated to be parallelized on GPUs.

2.1 Preliminaries

Let $(\Omega, \mathcal{F}, \mathbb{P}, \{\mathcal{F}_t\}_{0\le t\le T})$ be a complete, filtered probability space. In this space a standard d-dimensional Brownian motion W_t is defined, such that the filtration $\{\mathcal{F}_t\}_{0\le t\le T}$ is the natural filtration of W_t. We define $|\cdot|$ as the standard Euclidean norm in the Euclidean space $\mathbb{R}^m$ or $\mathbb{R}^{m\times d}$ and $L^2 = L^2_{\mathcal{F}}(0,T;\mathbb{R}^d)$ the set of all $\mathcal{F}_t$-adapted and square integrable processes valued in $\mathbb{R}^d$. Moreover, let $\mathcal{F}_s^{t,x}$ for $t \le s \le T$ be a σ-field generated by the Brownian motion $\{x + W_r - W_t, t \le r \le s\}$ starting from the time-space point (t,x). We define $E_s^{t,x}[X]$ as the conditional expectation of the random variable X under the filtration $\mathcal{F}_s^{t,x}$, i.e. $E_s^{t,x}[X] = E[X|\mathcal{F}_s^{t,x}]$.

A pair of processes $(y_t, z_t) : [0,T]\times\Omega \to \mathbb{R}^m \times \mathbb{R}^{m\times d}$ is the solution of BSDE (1) if it is $\mathcal{F}_t$-adapted, square integrable, and satisfies (1) in the sense of

$$y_t = \xi + \int_t^T f(s, X_s, y_s, z_s)\, ds - \int_t^T z_s\, dW_s, \quad t \in [0,T), \tag{3}$$

where $f(t, X_t, y_t, z_t) : [0,T]\times\mathbb{R}^m\times\mathbb{R}^n\times\mathbb{R}^{m\times d} \to \mathbb{R}^m$ is $\mathcal{F}_t$-adapted and the third term on the right-hand side is an Itô-type integral. This solution exist under regularity conditions [20].

Let us consider the semilinear PDE

$$\frac{\partial u}{\partial t} + \sum_{i=1}^{n} a_i(t,x)\frac{\partial u}{\partial x_i} + \frac{1}{2}\sum_{i,j=1}^{n}\sum_{k=1}^{d} b_{i,k}(t,x)b_{j,k}(t,x)\frac{\partial^2 u}{\partial x_i x_j} + f\big(t,x,u,b(t,x)^\top D_x u\big) = 0 \tag{4}$$

with the terminal condition $u(T,x) = g(x)$. The following theorem can be obtained with a straightforward application of Itô's lemma.

Theorem 1 (Nonlinear Feynman-Kac Theorem) *Let $u \in C^{1,2}$ satisfying (4) and suppose that there exists a constant C such that $|b(t,x)^\top D_x u(t,x)| \le C(1+|x|)$ for each $(t,x) \in [0,T]\times\mathbb{R}^m$, then*

$$y_t = u(t, X_t), \qquad z_t = b(t, X_t)^\top D_x u(t, X_t) \tag{5}$$

is the unique solution of (1).

We note that the authors in [19] show the existence and uniqueness of a solution for BSDEs driven by a Lévy process with moments of all orders, which can be used for pricing in a Lévy market. The nonlinear Feynman-Kac formula for a general non-Markovian BSDE has been established in [21], where the path-dependent quasi-linear parabolic PDEs are considered. Furthermore, Feynman-Kac representation of fully nonlinear PDEs has been investigated, e.g., in [23].

2.2 The stable semidiscrete scheme

Let N be a positive integer and $\Delta t = T/N$ the step size that partitions uniformly the time interval $[0,T]$: $0 = t_0 < t_1 < \cdots < t_{N-1} < t_N = T$, where $t_n = t_0 + n\Delta t$, $n = 0, 1, \ldots, N$.

Let k and K_y be two positive integers such that $1 \le k \le K_y \le N$, which represent the number of time layers and interpolation points respectively. The BSDE (2) can be ex-

pressed as

$$y_{t_n} = y_{t_{n+k}} + \int_{t_n}^{t_{n+k}} f(s, y_s, z_s)\, ds - \int_{t_n}^{t_{n+k}} z_s\, dW_s. \tag{6}$$

Taking the conditional expectation $E^x_{t_n}[\cdot]$ in (6) to obtain the adaptability of the solution and using cubic spline polynomial to approximate the integrand, the reference equation for y process reads (see the Appendix)

$$y_{t_n} = E^x_{t_n}[y_{t_{n+k}}] + \sum_{j=0}^{K_y-1}\left[a_j^y \Delta t + \frac{b_j^y \Delta t^2}{2} + \frac{c_j^y \Delta t^3}{3} + \frac{d_j^y \Delta t^4}{4}\right] + R_y^n,$$

where R_y^n is the interpolation error. For the z process, using l and K_z instead of k and K_y, multiplying both sides by $\Delta W_{t_{n+l}}$ in (6) and taking the conditional expectation $E^x_{t_n}[\cdot]$, the reference equation using cubic spline interpolation reads (see the Appendix)

$$\begin{aligned} 0 = {} & l\Delta t E^x_{t_n}[z_{t_{n+l}}] + \sum_{j=0}^{K_z-1}\left[a_j^{z_1} \Delta t + \frac{b_j^{z_1} \Delta t^2}{2} + \frac{c_j^{z_1} \Delta t^3}{3} + \frac{d_j^{z_1} \Delta t^4}{4}\right] \\ & - \sum_{j=0}^{K_z-1}\left[a_j^{z_2} \Delta t + \frac{b_j^{z_2} \Delta t^2}{2} + \frac{c_j^{z_2} \Delta t^3}{3} + \frac{d_j^{z_2} \Delta t^4}{4}\right] + R_z^n, \end{aligned}$$

where $R_z^n = R_{z_1}^n + R_{z_2}^n$ are the interpolation errors. In [27], the authors have shown that the scheme is stable when

$$\begin{aligned} & k = 1, \ldots, K_y, \quad \text{with } K_y = 1, 2, 3, \ldots, N, \\ & l = 1, \quad \text{with } K_z = 1, 2, 3, \ldots, N. \end{aligned}$$

This is to say that the algorithm allows for arbitrary multiple time levels K_y and K_z. Using the conditions of cubic spline interpolation to calculate the unknown coefficients, we have

$$\begin{aligned} y_{t_n} &= E^x_{t_n}[y_{t_{n+K_y}}] + \Delta t K_y \sum_{j=0}^{K_y} \gamma^{K_y}_{K_y,j} E^x_{t_n}\left[f(t_{n+j}, y_{t_{n+j}}, z_{t_{n+j}})\right] + R_y^n, \\ z_{t_n} &= \Bigg(E^x_{t_n}[z_{t_{n+1}}] + \sum_{j=1}^{K_z} \gamma^1_{K_z,j} E^x_{t_n}\left[f(t_{n+j}, y_{t_{n+j}}, z_{t_{n+j}}) \Delta W_{t_{n+j}}\right] \\ &\quad - \sum_{j=1}^{K_z} \gamma^1_{K_z,j} E^x_{t_n}[z_{t_{n+j}}] \Bigg) / \gamma^1_{K_z,0} + \frac{R_z^n}{\Delta t}, \end{aligned} \tag{7}$$

with $\gamma^{K_y}_{K_y,j}$ and $\gamma^1_{K_z,j}$ representing the calculated coefficients of the cubic spline interpolation (Table 1 and Table 2 give the values up to 6 time levels). It is shown in [27] that the local errors in (7) are given by

$$\left|R_y^n\right| = \mathcal{O}\left(\Delta t^5\right), \qquad \left|R_z^n\right| = \mathcal{O}\left(\Delta t^5\right),$$

Table 1 The coefficients $\{\gamma_{K_y,j}^{K_y}\}_{j=0}^{K_y}$ until $K_y = 6$

K_y	$\gamma_{K_y,j}^{K_y}$						
	$j=0$	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$	$j=6$
1	$\frac{1}{2}$	$\frac{1}{2}$					
2	$\frac{1}{6}$	$\frac{2}{3}$	$\frac{1}{6}$				
3	$\frac{1}{8}$	$\frac{3}{8}$	$\frac{3}{8}$	$\frac{1}{8}$			
4	$\frac{1}{12}$	$\frac{1}{3}$	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{12}$		
5	$\frac{41}{600}$	$\frac{19}{75}$	$\frac{107}{600}$	$\frac{107}{600}$	$\frac{19}{75}$	$\frac{41}{600}$	
6	$\frac{19}{336}$	$\frac{3}{14}$	$\frac{15}{112}$	$\frac{4}{21}$	$\frac{15}{112}$	$\frac{3}{14}$	$\frac{19}{336}$

Table 2 The coefficients $\{\gamma_{K_z,j}^{1}\}_{j=0}^{K_z}$ until $K_z = 6$

K_z	$\gamma_{K_z,j}^{1}$						
	$j=0$	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$	$j=6$
1	$\frac{1}{2}$	$\frac{1}{2}$					
2	$\frac{5}{12}$	$\frac{2}{3}$	$-\frac{1}{12}$				
3	$\frac{3}{8}$	$\frac{19}{24}$	$-\frac{5}{24}$	$\frac{1}{24}$			
4	$\frac{35}{96}$	$\frac{5}{6}$	$-\frac{13}{48}$	$\frac{1}{12}$	$-\frac{1}{96}$		
5	$\frac{131}{360}$	$\frac{151}{180}$	$-\frac{103}{360}$	$\frac{37}{360}$	$-\frac{1}{45}$	$\frac{1}{360}$	
6	$\frac{163}{448}$	$\frac{47}{56}$	$-\frac{129}{448}$	$\frac{3}{28}$	$-\frac{37}{1344}$	$\frac{1}{168}$	$-\frac{1}{1344}$

provided that f and g are smooth enough. In (7) we need to divide by Δt to find the value of z process. Therefore, in order to balance time truncation errors, one might set $K_z = K_y + 1$.

The stable semidiscrete scheme for the d-dimensional case is given as follows: we denote (y^n, z^n) as the approximation to (y_{t_n}, z_{t_n}), given random variables (y^{N-i}, z^{N-i}), $i = 0, 1, \ldots, K-1$ with $K = \max\{K_y, K_z\}$. Then (y^n, z^n) can be found for $n = N-K, \ldots, 0$ such that

$$\begin{aligned} y^n &= E_{t_n}^x\left[y^{n+K_y}\right] + \Delta t K_y \sum_{j=0}^{K_y} \gamma_{K_y,j}^{K_y} E_{t_n}^x\left[\left(t_{n+j}, y^{n+j}, z^{n+j}\right)\right], \\ z^n &= \left(E_{t_n}^x\left[z^{n+1}\right] + \sum_{j=1}^{K_z} \gamma_{K_z,j}^{1} E_{t_n}^x\left[f\left(t_{n+j}, y^{n+j}, z^{n+j}\right)\Delta W_{t_{n+j}}^{\top}\right] - \sum_{j=1}^{K_z} \gamma_{K_z,j}^{1} E_{t_n}^x\left[z^{n+j}\right]\right) \Big/ \gamma_{K_z,0}^{1} \end{aligned} \tag{8}$$

where $y^n = (y^{n,\tilde{m}})_{\tilde{m}\times 1}$, $z^n = (z^{n,\tilde{m},\tilde{d}})_{\tilde{m}\times d}$, $\Delta W_{t_{n+j}}^{\top} = (W_{t_{n+j}}^{\tilde{d}})_{\tilde{d}\times 1} - (W_{t_n}^{\tilde{d}})_{\tilde{d}\times 1}$, $\tilde{m} = 1, 2, \ldots, m$ and $\tilde{d} = 1, 2, \ldots, d$. In the following, we only present the results of the error analysis, for their proofs we refer to [27] and [32].

Lemma 1 *The local estimates of the local truncation errors in* (7) *satisfy*

$$\left|R_y^n\right| \le C\Delta t^{\min\{K_y+2,5\}}, \qquad \left|R_z^n\right| \le C\Delta t^{\min\{K_z+2,5\}},$$

where $C > 0$ *is a constant depending on* T, f, g *and the derivatives of* f *and* g.

Theorem 2 *Suppose that the initial values satisfy*

$$\begin{cases} \max_{N-K_y+1\le n\le N} E[|y_{t_n}-y^n|] = \mathcal{O}(\Delta t^{K_y+1}), \\ \max_{N-K_y+1\le n\le N} E[|y_{t_n}-y^n|] = \mathcal{O}(\Delta t^{4}), \end{cases}$$

where $K_y = 1,2,3$ for the first equation and $K_y > 3$ for the second one. For a sufficiently small time step Δt it can be shown that

$$\sup_{0\le n\le N} E\big[\big|y_{t_n}-y^n\big|\big] \le C\Delta t^{\min\{K_y+1,4\}},$$

where $C > 0$ is a constant depending on T, f, g and the derivatives of f and g.

Theorem 3 *Suppose that the initial values satisfy*

$$\begin{cases} \max_{N-K_z+1\le n\le N} E[|z_{t_n}-z^n|] = \mathcal{O}(\Delta t^{K_z}), \\ \max_{N-K_z+1\le n\le N} E[|z_{t_n}-z^n|] = \mathcal{O}(\Delta t^{3}), \end{cases}$$

where $K_z = 1,2,3$ for the first equation and $K_z > 3$ for the second one, and the condition on the initial values in Theorem 2 is fulfilled. For a sufficiently small time step Δt it can be shown that

$$\sup_{0\le n\le N} E\big[\big|z_{t_n}-z^n\big|\big] \le C\Delta t^{\min\{K_y+1,K_z,3\}},$$

where $C > 0$ is a constant depending on T, f, g and the derivatives of f and g.

Remark 1 If f does not depend on process z, the maximum order of convergence for the y process is 4 and 3 for the z process; If f depends on process z, the maximum order of convergence for the y and z processes is 3.

2.3 The fully discrete scheme

Let Δx denote the step size in the partition of the uniform d-dimensional real axis, i.e.

$$\mathbb{R}^{\tilde d} = \Big\{ x_i^{\tilde d} \big| x_i^{\tilde d} \in \mathbb{R}, i \in \mathbb{Z}, x_i^{\tilde d} < x_{i+1}^{\tilde d}, \Delta x = x_{i+1}^{\tilde d} - x_i^{\tilde d}, \lim_{i\to\pm\infty} x_i^{\tilde d} = \pm\infty \Big\},$$

where

$$\mathbb{R}^{\tilde d} = \mathbb{R}^1 \times \mathbb{R}^2 \times \cdots \times \mathbb{R}^d \quad \text{and} \quad \tilde d = 1,2,\dots,d.$$

Let $x_{\mathbf{i}} = (x_{i_1}^1, x_{i_2}^2, \dots, x_{i_d}^d)$ for $\mathbf{i} = (i_1, i_2, \dots, i_d) \in \mathbb{Z}^d$. We denote $(y_{\mathbf{i}}^n, z_{\mathbf{i}}^n)$ as the approximation to $(y_{t_n,x_{\mathbf{i}}}, z_{t_n,x_{\mathbf{i}}})$, given the random variables $(y_{\mathbf{i}}^{N-l}, z_{\mathbf{i}}^{N-l})$, $l = 0,1,\dots,K-1$ with $K = \max\{K_y, K_z\}$. Then $(y_{\mathbf{i}}^n, z_{\mathbf{i}}^n)$ can be found for $n = N-K,\dots,0$ such that

$$\begin{aligned} y_{\mathbf{i}}^n &= \hat E_{t_n}^{x_{\mathbf{i}}}\big[\hat y^{n+K_y}\big] + \Delta t K_y \sum_{j=1}^{K_y} \gamma_{K_y,j}^{K_y} \hat E_{t_n}^{x_{\mathbf{i}}}\big[f\big(t_{n+j}, \hat y^{n+j}, \hat z^{n+j}\big)\big] + \Delta t K_y \gamma_{K_y,0}^{K_y} f\big(t_n, y_{\mathbf{i}}^n, z_{\mathbf{i}}^n\big), \\ z_{\mathbf{i}}^n &= \left(\hat E_{t_n}^{x_{\mathbf{i}}}\big[\hat z^{n+1}\big] + \sum_{j=1}^{K_z} \gamma_{K_z,j}^{1} \hat E_{t_n}^{x_{\mathbf{i}}}\big[f\big(t_{n+j}, \hat y^{n+j}, \hat z^{n+j}\big)\Delta W_{t_{n+j}}^{\top}\big] - \sum_{j=1}^{K_z} \gamma_{K_z,j}^{1} \hat E_{t_n}^{x_{\mathbf{i}}}\big[\hat z^{n+j}\big] \right) \Big/ \gamma_{K_z,0}^{1}, \end{aligned} \tag{9}$$

where $\hat{E}_{t_n}^{x_i}[\cdot]$ is used to denote the approximation of the conditional expectation. The calculations at each space grid point $x_{\mathbf{i}}$ in (9) are independent, for each time layer t_n. Therefore, the parallelization strategy is fully related with the space discretization, which will be discussed in the following Sections.

The functions in the conditional expectations involve the d-dimensional probability density function of the Brownian Motions, one can choose e.g., the Gauss-Hermite quadrature rule to achieve a high accuracy only with a few space points. The conditional expectation can be can be sufficiently accurately approximated by a clever table interpolation

$$\hat{E}_{t_n}^{x_{\mathbf{i}}}\left[\hat{y}^{n+k}\right] = \frac{1}{\pi^{\frac{d}{2}}} \sum_{\Lambda=1}^{L} \omega_\Lambda \hat{y}^{n+k}(x_{\mathbf{i}} + \sqrt{2k\Delta t} a_\Lambda), \tag{10}$$

where $\hat{y}^{n+k}$ are interpolating values at the space points $(x_{\mathbf{i}} + \sqrt{2k\Delta t} a_\Lambda)$ based on y^{n+k} values, $(\omega_\Lambda, a_\Lambda)$ for $\Lambda = (\lambda_1, \lambda_2, \ldots, \lambda_d)$ are the weights and roots of the Hermite polynomial of degree L (see [12]), $\omega_\Lambda = \prod_{\tilde{d}=1}^{d} \omega_{\lambda_{\tilde{d}}}$, $a_\Lambda = (a_{\lambda_1}, a_{\lambda_2}, \ldots, a_{\lambda_d})$ and $\sum_{\Lambda=1}^{L} = \sum_{\lambda_1=1,\ldots,\lambda_d=1}^{L,\ldots,L}$. In a similar way, one can express the other conditional expectations in (9).

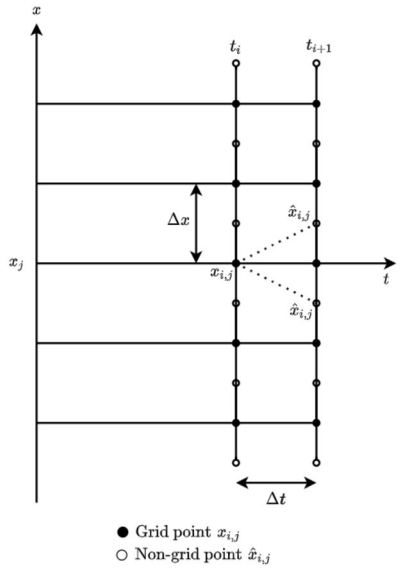
3 The algorithmic framework

According to the numerical algorithm represented in Sect. 2, the whole process for numerically solving BSDEs can be divided into three steps.

1. **Construct the time-space discrete domain.**
   We divide the time period $[0, T]$ into N time steps using $\Delta t = T/N$, i.e., $N + 1$ time layers, and the space domain $\mathbb{R}^d$ using step size Δx (see also Sect. 2.3). We use the truncated domains $[-16, 16]$ and $[-8, 8]$ for the grid space, where the former is used for 1-dimensional examples and the latter for the 2-dimensional ones. Note that larger domain can be also used, however, the approximation will be not improved. This is to say that those truncated domains are sufficient for our numerical experiments. Furthermore, in order to balance the errors in time and space directions, we adjust Δx and Δt such that they satisfy the equality $(\Delta x)^r = (\Delta t)^{q+1}$, where $q = \min\{K_y + 1, K_z\}$ and r denotes the global error from the interpolation method used to generate the non-grid points when calculating the conditional expectations. For a better illustration, we show a visualisation of the domain for $K_y = K_z = 1$, $d = 1$ in Fig. 1, including the non-grid points.
2. **Calculate K initial solutions with $K = \max\{K_y, K_z\}$.**
   Generally, only the terminal values are given and one needs to compute the other $K - 1$ initial values. To obtain these initial values, we start with $K = 1$ and choose an extremely small time step size Δt.
3. **Calculate the numerical solution (y_0^0, z_0^0) backward using equation (9).**
   Note that the calculation for the y process is done implicitly with Picard iteration.

In step 1, some of the generated non-grid points could be outside of the truncated domain (see Fig. 1). For these points, we take the values on the boundaries as approximations (constant extrapolation). Note that one could also do extrapolation for those points outside, however, a longer computation time will be needed. As mentioned in (10), we use interpolation to approximate the values for y and z at non-grid points. The computation time drawback when interpolating is finding the position of the new points in the interpolating process. A natural search algorithm is to loop over all the grid points, and find in

Figure 1 Time-space domain



which interval the point belongs to. In the worst case, an $\mathcal{O}(M^d)$ work is needed. Fortunately, the structure of the Gauss-Hermite quadrature creates the symmetry for the non-grid points. Recall that each new point is generated as $X_{\lambda_{\tilde{d}}} = x_{i_{\tilde{d}}} + \sqrt{2k\Delta t}a_{\lambda_{\tilde{d}}}$. This means that taking $\text{int}(\frac{X_{\lambda_{\tilde{d}}} - x_{\min}}{\Delta x})$ for $x_{i_{\tilde{d}}} \in [x_{\min}, x_{\max}]$ and $M - \text{int}(\frac{X_{\lambda_{\tilde{d}}} - x_{\min}}{\Delta x})$ for $x_{i_{\tilde{d}}} \in [x_{\max}, x_{\min}]$ gives the left boundary of the grid interval that $X_{\lambda_{\tilde{d}}}$ belongs to, with $\text{int}(x)$ giving the integer part of x. As a result, this step in the algorithm can be done in $\mathcal{O}(d)$, i.e., without a for-loop. This benefit comes from the uniformity of the space domain. This substantially reduces the total computation time, as it will be demonstrated in the numerical experiments.

In step 2, we do not consider $2K$ (K for y and K for z) interpolations for each new calculation, but only 2. Suppose that we are at time layer t_{n-K}. To calculate y and z values on this time layer, one needs the calculation of conditional expectations for K time layers. In our numerical experiments, we consider 1 and 2 dimensional examples, the higher dimensional problem can be parallelized for a device (GPU) with large memory. The cubic spline interpolation is used to find the non-grid values for 1-dimensional cases and bicubic interpolation for 2-dimensional cases. For instance, the coefficients for the y process are $A_y \in \mathbb{R}^{K\times(4^d\times M^d)}$, all the coefficients are stored. When we are at time layer t_{n-K-1}, only the spline interpolation corresponding to the previous calculated values is considered. Then, the columns of matrix A_y are shifted +1 to the right in order to delete the last column and enter the current calculated coefficients in the first column. The new A_y is used for the current step. The same procedure is followed until t_0. This reduces as well the amount of work for the algorithm.

In the final step (step 3), we consider to merge calculation of conditional expectations in (9), which is important for the reduction of computation time in the parallel implementation. This will be mentioned in more details in the next Section.

4 Parallel algorithm

In this Section we present the parallelization strategy of the multistep scheme presented in the previous sections. Basically, we parallelize the problem straightforwardly, while keeping attention on the optimal Compute Unified Device Architecture (CUDA) execution

model, i.e., creating arrays such that the access will be aligned and coalesced, reducing the redundant access to global memory, using registers when needed etc..

The first and second steps of the algorithm are implemented in the host. The third step, together with the operator for performing $\mathcal{O}(d)$ work in the searching procedure of the interpolation and the idea of shifting coefficients of this procedure, are fully implemented in the GPU device. Recall from (9) that the following steps are needed to calculate the approximated values on each unknown time layer backward:

- **Generation of the non-grid points $\boldsymbol{X_\Lambda = x_i + \sqrt{2k\Delta t}a_\Lambda}$.**

  In the uniform space domain, the non-grid points need to be generated only once. To do this, a kernel is created where each thread generates L^d points (Gauss-Hermite points) for each space direction. This kernel adds insignificant computing time in the total algorithmic time.
- **Calculation of the values $\hat{y}$ and $\hat{z}$ at the non-grid points.**

  This is the most time consuming part of the algorithm, which is related with finding the corresponding interpolating functions for the given y and z points, in order to interpolate their values in the non-grid ones. For the 1-dimensional cases, we have considered the cubic spline interpolation. Since (9) involves the solution of two linear systems, the Biconjugate Gradient Stabilized (*BiCGSTAB*) [28] iterative method is used since the matrix is tridiagonal. For this, we consider the CUDA Basic Linear Algebra Subroutine (*cuBLAS*) and CUDA Sparse (*cuSPARSE*) libraries [5]. For the inner product, second norm and addition of vectors, we use the *cuBLAS* library. For the matrix vector multiplication, we use the *cuSPARSE* library with the compressed sparse row format, due to the structure of the system matrix. Moreover, we created a kernel to calculate the spline coefficients based on the solved systems. Finally, a kernel to apply the operator for the searching procedure of interpolation is created to find the values at non-grid points. Note that each thread is assigned to find $m + m \times d$ values (m for y and $m \times d$ for z). For the 2-dimensional examples, we have considered the bicubic interpolation. We need to calculate 16 coefficients for each point. Based on the bicubic interpolation idea, we need the first and mixed derivatives. These are approximated using finite difference schemes of the fourth order of accuracy (central for the interior points, forward and backward for the boundary points). Therefore, a kernel is created where each thread calculates these values. Moreover, to find the 16 coefficients, a matrix vector multiplication needs to be applied for each point. Therefore, each thread performs a matrix-vector multiplication using another kernel. Finally, a kernel to apply the operator for the searching procedure of interpolation is created to find the values at non-grid points, where each thread calculates $m + m \times d$ values.
- **Calculation of the conditional expectations.**

  As mentioned above, we merge the calculation of conditional expectations. For the first conditional expectations in the right hand side of (9), we create one kernel, where each thread calculates one value by using (10). For the other ones, we merged their calculation (three conditional expectations) in one kernel, namely $\hat{E}_{t_n}^{x_i}[\hat{z}^{n+j}]$, $\hat{E}_{t_n}^{x_i}[f(t_{n+j}, \hat{y}^{n+j}, \hat{z}^{n+j})]$ and $\hat{E}_{t_n}^{x_i}[f(t_{n+j}, \hat{y}^{n+j}, \hat{z}^{n+j})\Delta W_{t_{n+j}}]$, for $j = 1, 2, \ldots, K$. This reduces the accessing of data multiple times from the global memory and gives more work to the thread, such that it does not stay idle. Note that one thread calculates $2 \times m \times d + m$ values.

- **Calculation of the z values.**
  The second equation in (9) is used and each thread calculates $m \times d$ values.
- **Calculation of the y values.**
  The first equation in (9) is used and each thread calculates m values, using the Picard iterative process.

In the next Section, we present some numerical examples to show the accelerated computation on GPU computing with applications in option pricing.

5 Numerical results

We implement the parallel algorithm using CUDA C programming. The parallel computing times (only for one run) are compared with the serial ones on a CPU. Furthermore, the speedups are calculated. The Central Processing Unit (CPU) is Intel(R) Core(TM) i5-4670 3.40Ghz with 4 cores, where only one core is used for the serial calculations. The GPU is a NVIDIA GeForce 1070 Ti with a total 8GB GDDR5 memory.

We choose the degree of the Hermite polynomial $L = 32$ for the 1-dimensional examples and $L = 8$ for the 2-dimensional ones. For the Picard interations, we choose $p = 30$. With these values, the quadrature error and iteration error are so small that they won't affect the convergence rate.

Example 1 Consider the nonlinear BSDE [32].

$$\begin{cases} -dy_t = (-y_t^3 + \frac{5}{2}y_t^2 - \frac{3}{2}y_t)\,dt - z_t\,dW_t, \\ y_T = \frac{\exp(W_T+T)}{\exp(W_T+T)+1}. \end{cases}$$

The analytic solution is

$$\begin{cases} y_t = \frac{\exp(W_t+t)}{\exp(W_t+t)+1}, \\ z_t = \frac{\exp(W_t+t)}{(\exp(W_t+t)+1)^2}. \end{cases}$$

The exact solution with $T = 1$ is $(y_0, z_0) = (\frac{1}{2}, \frac{1}{4})$. In Table 3, we show the importance of the proposed technique when interpolating non-grid values, for a high N. Note that the computation time is given in seconds. We see that without the operator to find the position of the non-grid points in space, the computation serial time is very high.

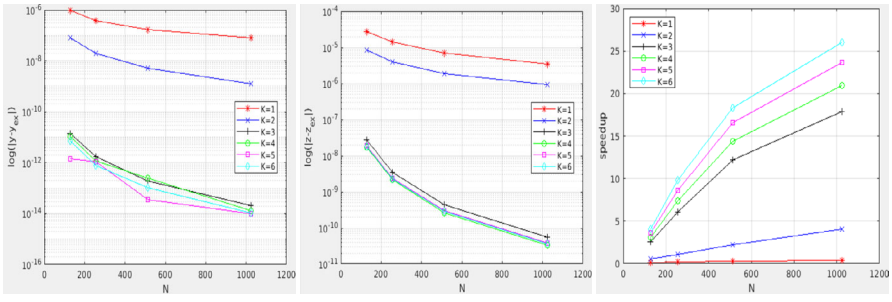
In Table 4, we present the results using 256 threads per block with $K = K_y = K_z$, $t_0 = 0$ and $T = 1$. As mentioned before, the better accuracy can be archived by using a higher-step scheme. For $K > 3$, the errors decrease moderately due to Theorem 2. The higher the value of time layers N the more work can be assigned to the GPU, and the speedup of the application can thus be increased. The speed up also increases on a smaller magnitude for $K > 3$, since the number of space data points given from M is the same. These are visualized in the plots of $\log_{10}(|y_{0,0} - y_0^0|)$, $\log_{10}(|z_{0,0} - z_0^0|)$ and speedup with respect to N

Table 3 Preliminary results with $N = 256$, $K_y = K_z = 3$ for Example 1

M	$t_{\text{CPU}}^{\text{non-optimal}}$	$t_{\text{CPU}}^{\text{optimal}}$	speedup
8192	2041.89	11.02	185.31

Table 4 The results for Example 1

K	N	M	$\lvert y_{0,0} - y_0^0 \rvert$	$\lvert z_{0,0} - z_0^0 \rvert$	t_{CPU}	t_{GPU}	speedup
1	128	364	9.36E−07	2.78E−05	0.14	0.91	0.15
1	256	512	3.89E−07	1.40E−05	0.37	1.73	0.21
1	512	726	1.74E−07	7.04E−06	1.06	3.57	0.30
1	1024	1024	8.22E−08	3.53E−06	2.91	6.96	0.42
2	128	1218	8.01E−08	8.61E−06	0.64	1.05	0.61
2	256	2048	2.03E−08	4.00E−06	2.06	1.88	1.10
2	512	3446	5.02E−09	1.92E−06	7.18	3.21	2.24
2	1024	5794	1.25E−09	9.41E−07	23.93	5.83	4.10
3	128	4096	1.44E−11	2.77E−08	2.71	1.04	2.61
3	256	8192	1.70E−12	3.50E−09	11.02	1.82	6.06
3	512	16,384	1.87E−13	4.41E−10	44.86	3.68	12.19
3	1024	32,768	2.05E−14	5.53E−11	180.30	10.08	17.89
4	128	4096	1.06E−11	1.69E−08	3.28	1.05	3.13
4	256	8192	1.20E−12	2.13E−09	13.57	1.84	7.36
4	512	16,384	2.57E−13	2.68E−10	55.16	3.84	14.35
4	1024	32,768	1.29E−14	3.34E−11	223.28	10.68	20.91
5	128	4096	1.46E−12	1.90E−08	3.86	1.06	3.63
5	256	8192	1.12E−12	2.40E−09	16.23	1.88	8.65
5	512	16,384	3.46E−14	3.02E−10	65.80	3.97	16.57
5	1024	32,768	9.77E−15	3.78E−11	267.79	11.33	23.64
6	128	4096	6.94E−12	1.84E−08	4.53	1.10	4.11
6	256	8192	7.71E−13	2.32E−09	18.97	1.93	9.84
6	512	16,384	1.07E−13	2.92E−10	77.64	4.23	18.35
6	1024	32,768	1.03E−14	3.65E−11	311.87	11.97	26.06



(a) Plot of y error as a function of time layers N for $K = 1, \cdots, 6$.

(b) Plot of z error as a function of time layers N for $K = 1, \cdots, 6$.

(c) Plot of speedup as a function of time layers N for $K = 1, \cdots, 6$.

Figure 2 Plots of the results for Example 1

for $K = 1, \ldots, 6$ in Fig. 2. The highest speedup ($26\times$) is for the multistep scheme with $K = 6$ and $N = 1024$.

Example 2 Consider the nonlinear BSDE [32]

$$\begin{cases} -dy_t = \frac{1}{2}(\exp(t^2) - 4ty_t - 3\exp(t^2 - y_t\exp(-t^2)) + z_t^2\exp(-t^2))\,dt - z_t\,dW_t, \\ y_T = \ln(\sin(W_T) + 3)\exp(T^2). \end{cases}$$

Table 5 The results for Example 2

K	N	M	$\lvert y_{0,0} - y_0^0 \rvert$	$\lvert z_{0,0} - z_0^0 \rvert$	t_{CPU}	t_{GPU}	speedup
1	128	364	7.85E−04	3.52E−03	0.32	1.17	0.27
1	256	512	3.77E−04	1.76E−03	0.88	2.13	0.41
1	512	726	1.85E−04	8.78E−04	2.52	4.04	0.62
1	1024	1024	9.15E−05	4.39E−04	6.98	7.80	0.89
2	128	1218	1.85E−04	8.37E−04	1.52	1.24	1.23
2	256	2048	9.13E−05	4.29E−04	5.13	2.51	2.04
2	512	3446	4.54E−05	2.17E−04	17.47	5.11	3.42
2	1024	5794	2.26E−05	1.09E−04	58.93	10.65	5.53
3	128	4096	1.92E−07	8.34E−07	6.61	1.53	4.31
3	256	8192	2.41E−08	1.06E−07	26.81	2.97	9.03
3	512	16,384	3.02E−09	1.33E−08	108.92	6.62	16.46
3	1024	32,768	3.77E−10	1.67E−09	435.23	18.35	23.71
4	128	4096	1.10E−07	4.86E−07	8.06	1.53	5.28
4	256	8192	1.42E−08	6.28E−08	32.82	3.02	10.87
4	512	16,384	1.80E−09	7.99E−09	133.26	6.47	20.61
4	1024	32,768	2.27E−10	1.01E−09	538.13	19.33	27.84
5	128	4096	1.20E−07	5.40E−07	9.48	1.54	6.14
5	256	8192	1.58E−08	7.04E−08	38.68	2.97	13.05
5	512	16,384	2.02E−09	8.99E−09	156.63	6.67	23.48
5	1024	32,768	2.55E−10	1.14E−09	635.01	19.55	32.48
6	128	4096	1.11E−07	5.08E−07	10.91	1.54	7.07
6	256	8192	1.49E−08	6.71E−08	44.77	3.09	14.48
6	512	16,384	1.93E−09	8.63E−09	182.74	7.15	25.57
6	1024	32,768	2.45E−10	1.09E−09	735.15	20.97	35.05

The analytic solution is

$$\begin{cases} y_t = \ln(\sin(W_t) + 3)\exp(t^2), \\ z_t = \exp(t^2)\frac{\cos(W_t)}{\sin(W_t)+3}. \end{cases}$$

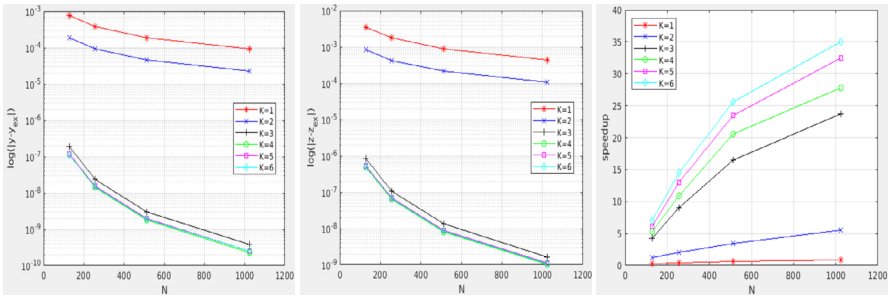
The exact solution with $T = 1$ is $(y_0, z_0) = (\ln(3), \frac{1}{3})$. The results using 256 threads per block with $K = K_y = K_z$, $t_0 = 0$ and $T = 1$ are presented in Table 5. We conclude the same as in the previous example, except the fact that the accuracy is decreased. This is due to the convergence order given in Theorem 3, which reduces to be at most 3, since the driver function depends on the z process. Furthermore, we get higher speedup compared with previous example due to the more complicated driver function (i.e. more data are accessed, more special functional unit is used etc.). The speedup is 35×. We display the plots of $\log_{10}(|y_{0,0} - y_0^0|)$, $\log_{10}(|z_{0,0} - z_0^0|)$ and speedup with respect to N for $K = 1, \ldots, 6$ in Fig. 3.

To optimize the application, we have used the following iterative approach:

1. Apply a profiler to the application to gather information
2. Identify application hotspots
3. Determine performance inhibitors
4. Optimize the code
5. Repeat the previous steps until desired performance is achieved

We considered the case with $N = 1024$ and $K_y = K_z = 6$. To make the optimization process be more clear, we show the detailed information after each optimization iteration as follows.

- In the first iteration of the optimization process, we gathered the application information using *nvprof* (NVIDIA Command-line Profiler). The results are



(a) Plot of y error as a function of time layers N for $K = 1, \cdots, 6$.

(b) Plot of z error as a function of time layers N for $K = 1, \cdots, 6$.

(c) Plot of speedup as a function of time layers N for $K = 1, \cdots, 6$.

Figure 3 Plots of the results for Example 2

Table 6 Results of iterative optimization process for Example 2

Time (%)	Time (s)	Kernel name
(a) Performance of the main kernels		
48.35	8.04	nrm2_kernel
14.94	2.48	sp_inter_non_grid_d_no_for
13.70	2.28	calc_f_and_c_exp_d
6.17	1.03	csrMv_kernel
3.60	0.60	calc_y
3.53	0.89	dot_kernel
1.98	0.33	reduce_1Block_kernel
1.56	0.26	axpby_kernel_val
1.34	0.22	calc_c_exp_d
(b) Performance after first iteration of optimization process		
27.88	2.49	sp_inter_non_grid_d_no_for
25.53	2.28	calc_f_and_c_exp_d
11.35	1.01	csrMv_kernel
9.64	0.86	dot_kernel
6.74	0.60	calc_y
5.22	0.47	reduce_1Block_kernel
2.65	0.24	axpby_kernel_val
2.50	0.22	calc_c_exp_d
1.76	0.16	step_3
(c) Performance after second iteration of optimization process		
22.23	1.46	calc_f_and_c_exp_d
17.67	1.16	sp_inter_non_grid_d_no_for
15.58	1.02	csrMv_kernel
12.86	0.84	dot_kernel
9.05	0.60	calc_y
7.21	0.47	reduce_1Block_kernel
3.41	0.22	axpby_kernel_val
2.38	0.16	step_3
2.12	0.14	copy_d

presented in Table 6(a). The application hotspot is the *nrm2_kernel* kernel, which calculates the second norm in the *BiCGSTAB* algorithm. This is already optimized. Therefore, to overcome this bottleneck, we used the dot kernel *dot_kernel*. The computation time is reduced from 8.04 s to 0.86 s. The new speedup after the first iteration becomes 57× in stead of 35×.

- In the second iteration, the next bottleneck for the application is the kernel that calculates the non-grid values for process y and z (*sp_inter*_non*_grid_d_no_for*) after

each time layer backward. The performance of the kernel is limited by the latency of arithmetic and memory operations. Therefore, we considered loop interchanging and loop unrolling techniques. This reduced the computation time of the corresponding kernel and other kernels related with it, as shown in Table 6(b). By this, we reduced the computation time for *sp_inter_non_grid_d_no_for* from 2.48 s to 1.16 s. By default, we have reduced the computation time from 2.28 s to 1.46 s for *calc_f_and_c_exp* (the kernel in the third item of Sect. 4) because we needed to change the way how the non-grid points are stored and accessed and also reduction for *calc_c_exp_d* (calculates the conditional expectation) from 0.22 s to 0.04 s. The new speedup is 69×. It can be observed from Table 6(c) that again the application bottleneck is the same kernel. Therefore, it is not worth optimizing the application furthermore.

- Finally, we decreased the block dimension from 256 threads to 128 in order to increase parallelism. The final speedup is 70×.

Example 3 Consider the Black-Scholes FBSDE [14]

$$\begin{cases} dS_t = \mu_t S_t\,dt + \sigma_t S_t\,dW_t, \quad S_0 = x, \\ -dy_t = -(r_t y_t + (\mu_t - r_t + \delta(t,S_t))\frac{z_t}{\sigma_t})\,dt + z_t\,dW_t, \\ y_T = (S_T - K)^+. \end{cases}$$

For constant parameters (i.e. $r_t = r$, $\mu_t = \mu$, $\sigma_t = \sigma$, $\delta_t = \delta$), the analytic solution for a call option is

$$\begin{cases} y_t = V(t,S_t) = S_t \exp(-\delta(T-t))N(d_1) - K\exp(-r(T-t))N(d_2), \\ z_t = S_t\frac{\partial V}{\partial S}\sigma = S_t\exp(-\delta(T-t))N(d_1)\sigma, \\ d_{1/2} = \frac{\ln(\frac{S_t}{K}) + (r \pm \frac{\sigma^2}{2})(T-t)}{\sigma\sqrt{T-t}}, \end{cases}$$

where $N(\cdot)$ is the cumulative standard normal distribution function. In this example, we consider $T = 0.33$, $K = S_0 = 100$, $r = 0.03$, $\mu = 0.05$, $\delta = 0.04$ and $\sigma = 0.2$ (taken from [32]) with the solution $(y_0, z_0) \doteq (4.3671, 10.0950)$. Note that the terminal condition has a non-smooth problem for the z process. Therefore, for discrete points near the strike price K (also called at the money region), the initial value for the z process will cause large errors on the next time layers. To overcome this non-smoothness problem, we considered smoothing the initial conditions, cf. the approach of Kreiss [15]. For the forward part of Example 3, we used the analytic solution

$$S_t = S_0 \exp\left(\left(\mu - \frac{\sigma^2}{2}\right)t + \sigma W_t\right).$$

In order to ensure a uniform stock price domain, we switch to the log stock price domain $X_t = \ln(S_t)$. In Table 7 we show the importance of using this transformation.

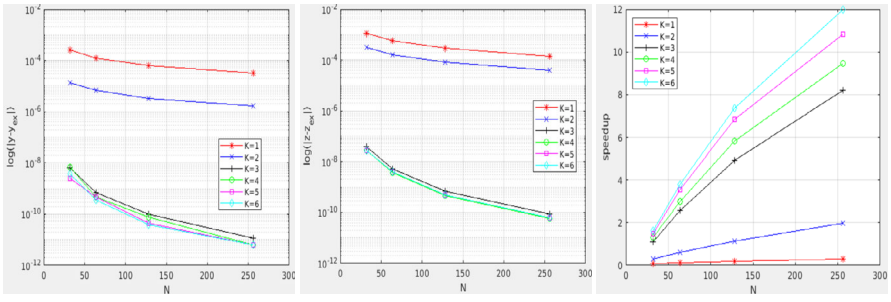
The results using 256 threads per block with $K = K_y = K_z$, $t_0 = 0$ and $T = 0.33$ are presented in Table 8. As in the previous BSDE examples, the highest accuracy is achieved for the maximal considered number of steps K and the number of time layers N, namely a 6-step scheme with $N = 256$, where we also have the highest speedup of 12×. We draw

Table 7 Preliminary results with $N = 256$, $K_y = K_z = 3$ for Example 3

M	$t_{\text{CPU}}^{\text{non-optimal}}$	$t_{\text{CPU}}^{\text{optimal}}$	speedup
24,826	18,531.23	47.54	389.80

Table 8 The results for Example 3

K	N	M	$\lvert y_{0,0} - y_0^0 \rvert$	$\lvert z_{0,0} - z_0^0 \rvert$	t_{CPU}	t_{GPU}	speedup
1	32	316	2.55E−04	1.11E−03	0.04	0.60	0.07
1	64	446	1.24E−04	5.70E−04	0.12	1.03	0.11
1	128	632	6.21E−05	2.89E−04	0.33	1.79	0.18
1	256	892	3.12E−05	1.45E−04	0.93	3.38	0.28
2	32	990	1.34E−05	3.12E−04	0.18	0.61	0.29
2	64	1664	6.88E−06	1.59E−04	0.64	1.04	0.61
2	128	2798	3.38E−06	8.04E−05	2.16	1.92	1.13
2	256	4704	1.69E−06	4.04E−05	7.34	3.73	1.97
3	32	3104	6.45E−09	3.98E−08	0.70	0.63	1.11
3	64	6208	6.88E−10	5.35E−09	2.93	1.14	2.58
3	128	12,414	9.72E−11	6.85E−10	11.81	2.39	4.93
3	256	24,826	1.15E−11	8.50E−11	47.54	5.79	8.21
4	32	3104	6.86E−09	2.73E−08	0.85	0.64	1.32
4	64	6208	4.78E−10	3.81E−09	3.47	1.15	3.00
4	128	12,414	7.55E−11	4.71E−10	14.26	2.45	5.82
4	256	24,826	6.36E−12	5.93E−11	57.54	6.07	9.48
5	32	3104	2.55E−09	2.85E−08	0.94	0.64	1.48
5	64	6208	4.73E−10	4.05E−09	4.04	1.14	3.56
5	128	12,414	4.40E−11	5.04E−10	16.30	2.39	6.83
5	256	24,826	6.22E−12	6.41E−11	67.33	6.22	10.83
6	32	3104	3.77E−09	2.71E−08	1.06	0.65	1.64
6	64	6208	3.56E−10	3.90E−09	4.50	1.18	3.80
6	128	12,414	3.82E−11	4.89E−10	18.69	2.54	7.35
6	256	24,826	6.16E−12	6.24E−11	77.53	6.47	11.99



(a) Plot of y error as a function of time layers N for $K = 1, \cdots, 6$. (b) Plot of z error as a function of time layers N for $K = 1, \cdots, 6$. (c) Plot of speedup as a function of time layers N for $K = 1, \cdots, 6$.

Figure 4 Plots of results for Example 3

the plots of $\log_{10}(|y_{0,0} - y_0^0|)$, $\log_{10}(|z_{0,0} - z_0^0|)$ and speedup with respect to N for $K = 1, \ldots, 6$ in Fig. 4.

Compared to the parallelized multistep scheme [32] in [13], the numerical results are more accurate, since we can considered $K > 3$. Moreover, we optimized the kernels created for the Black-Scholes BSDE for $N = 256$ and a high K, namely $K = K_y = K_z = 6$. The optimization iteration process is the same as in Example 2. The final speedup is 31×. Note

that this speedup is for 256 time layers. In Example 2, we optimized for 1024 time layers. The high accuracy $\mathcal{O}(10^{-12})$ with a high value of K can be achieved for more modest computation time then 6.47 seconds.

Obviously, it is more interesting to achieve the parallelization of higher dimensional BSDEs on the GPUs.

Example 4 We start with the 2-dimensional BSDE [30]

$$\begin{cases} -dy_t = (y_t - z_t A)\,dt - z_t\,dW_t, \\ y_T = \sin(MW_T + T), \end{cases}$$

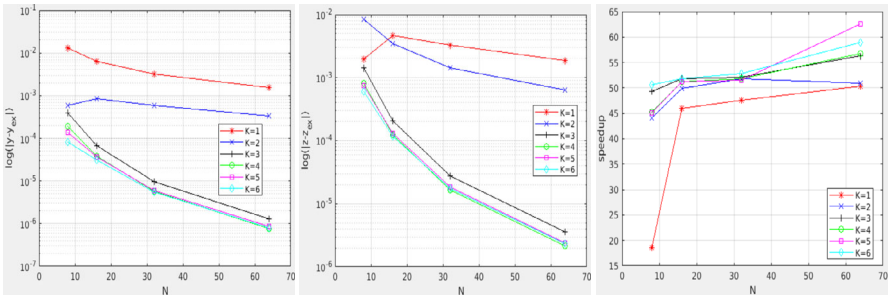
where $W_t = (W_t^1, W_t^2)^\top$, $z_t = (z_t^1, z_t^2)$, $A = (\frac{1}{2}, \frac{1}{2})^\top$ and $M = (1, 1)$.

The analytic solution is

$$\begin{cases} y_t = \sin(MW_t + t), \\ z_t = (\cos(MW_t + t), \cos(MW_t + t)). \end{cases}$$

The exact solution with $T = 1$ is $(y_0, (z_0^1, z_0^2)) = (0, (1, 1))$. The results using 256 threads per block with $K = K_y = K_z$, $t_0 = 0$ and $T = 1$ are presented in Table 9. Note that $|z_{0,0} - z_0^0|$ in 2-dimensional case is given by $\frac{1}{2}(|z_{0,0}^1 - z_0^{0,1}| + |z_{0,0}^2 - z_0^{0,2}|)$. The plots of $\log_{10}(|y_{0,0} - y_0^0|)$, $\log_{10}(|z_{0,0} - z_0^0|)$ and speedup with respect to N for $K = 1, \ldots, 6$ are displayed in Fig. 5. We conclude the same as before in terms of the error reduction, except for the speedup. It is not clear in 2-dimension the increase in magnitude of the later when growing K, since we have considered maximal number of time layers $N = 64$. The highest speedup is ca. 59×, which requires around 3 GB of memory.

Due to GPU memory limitation (8 GB), we optimized the case where $N = 64$ and $K_y = K_z = 6$. In the 1-dimensional case, we used different kernels for the calculation of non-grid points, conditional expectations and the values for processes y and z. Here we merged these kernels due to memory constraint and named it *main_body* kernel. In the first iteration, we gathered the application information using *nvprof*. The results are presented in Table 10. The application hotspot is *main_body* kernel. The performance of the



(a) Plot of y error as a function of time layers N for $K = 1, \cdots, 6$. (b) Plot of z error as a function of time layers N for $K = 1, \cdots, 6$. (c) Plot of speedup as a function of time layers N for $K = 1, \cdots, 6$.

Figure 5 Plots of the results for Example 4

Table 9 The results for Example 4

K	N	M	$\lvert y_{0,0} - y_0^0 \rvert$	$\lvert z_{0,0} - z_0^0 \rvert$	t_{CPU}	t_{GPU}	speedup	Memory
1	8	46	1.32E−02	1.95E−03	0.14	0.01	18.60	0.30
1	16	64	6.46E−03	4.64E−03	0.53	0.01	45.96	0.30
1	32	92	3.18E−03	3.24E−03	2.12	0.04	47.55	0.30
1	64	128	1.57E−03	1.85E−03	8.73	0.17	50.33	0.30
2	8	78	5.90E−04	8.49E−03	0.62	0.01	44.08	0.30
2	16	128	8.31E−04	3.46E−03	3.56	0.07	49.93	0.31
2	32	216	5.84E−04	1.44E−03	21.96	0.42	51.75	0.34
2	64	364	3.39E−04	6.39E−04	131.09	2.57	50.95	0.42
3	8	128	3.94E−04	1.43E−03	2.12	0.04	49.40	0.32
3	16	256	6.75E−05	2.08E−04	21.05	0.41	51.83	0.38
3	32	512	9.72E−06	2.79E−05	187.79	3.61	52.04	0.65
3	64	1024	1.30E−06	3.60E−06	1719.55	30.54	56.30	1.66
4	8	128	1.90E−04	8.09E−04	2.41	0.05	45.19	0.33
4	16	256	3.78E−05	1.24E−04	25.99	0.51	51.18	0.40
4	32	512	5.69E−06	1.67E−05	239.73	4.63	51.80	0.75
4	64	1024	7.76E−07	2.17E−06	2279.06	40.10	56.83	2.09
5	8	128	1.39E−04	7.49E−04	2.39	0.05	45.06	0.35
5	16	256	3.65E−05	1.30E−04	30.16	0.59	51.27	0.43
5	32	512	6.00E−06	1.83E−05	292.10	5.67	51.54	0.86
5	64	1024	8.48E−07	2.42E−06	3049.33	48.75	62.55	2.52
6	8	128	8.13E−05	5.97E−04	2.21	0.04	50.61	0.34
6	16	256	3.07E−05	1.18E−04	32.93	0.64	51.76	0.46
6	32	512	5.49E−06	1.73E−05	341.15	6.46	52.81	0.97
6	64	1024	8.00E−07	2.32E−06	3394.13	57.56	58.96	2.94

Table 10 Performance of the main kernels for Example 4

Time (%)	Time (s)	Kernel name
83.27	47.83	main_body
13.18	7.57	swap_coeff
3.23	1.86	find_coeff
0.16	0.91	swap_val

kernel is limited by the memory operations, because the access of the bicubic spline coefficients is not optimal. However, we can't optimize this part, otherwise requesting aligned and coalesced memory operations shifts the spine coefficients and the threads access the wrong coefficients. Moreover, we can't increase the block dimension because there are not enough resources. Therefore, the final speedup is ca. 59×.

Finally, we consider the zero strike European spread option, which is a 2-dimensional problem.

Example 5 The zero strike European spread option BSDE reads

$$\begin{cases} dS_t = \mu S_t\,dt + \sigma S_t\,dW_t, \quad S_0 = x, \\ E[dW_t] = \rho d_t, \\ -dy_t = -(ry_t + z_t A^{-1} M^{\top})\,dt + z_t\,dW_t, \\ y_T = (S_T^1 - S_T^2)^+. \end{cases}$$

where $S_t = (S_t^1, S_t^2)^{\top}$, $\mu = (\mu_1, \mu_2)$, $\sigma = (\sigma_1, \sigma_2)$, $W_t = (W_t^1, W_t^2)^{\top}$, $z_t = (z_t^1, z_t^2)$, $A = \begin{pmatrix} \sigma_1 & 0 \\ \rho\sigma_2 & \sigma_2\sqrt{1-\rho^2} \end{pmatrix}$ and $M = (\mu_1 - r, \mu_2 - r)$. The analytic solution is given by Margrabe's formula

Table 11 The results for Example 5

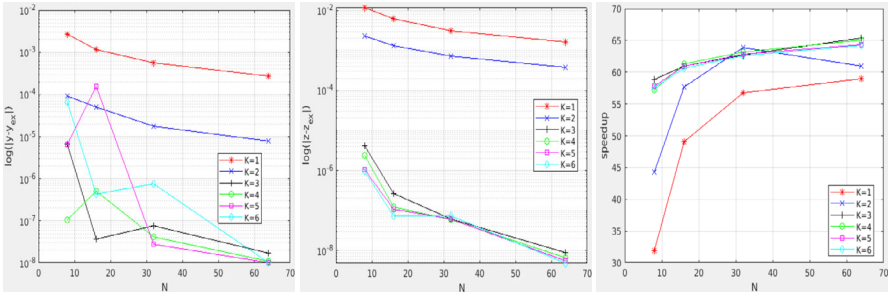
K	N	M	$\lvert y_{0,0} - y_0^0 \rvert$	$\lvert z_{0,0} - z_0^0 \rvert$	t_{CPU}	t_{GPU}	speedup	Memory
1	8	46	2.71E−03	1.21E−02	0.15	0.00	31.96	0.33
1	16	64	1.16E−03	6.22E−03	0.61	0.01	49.03	0.34
1	32	92	5.62E−04	3.18E−03	2.49	0.04	56.82	0.35
1	64	128	2.67E−04	1.61E−03	10.29	0.17	58.93	0.34
2	8	78	9.09E−05	2.31E−03	0.65	0.01	44.33	0.35
2	16	128	5.06E−05	1.34E−03	4.08	0.07	57.72	0.35
2	32	216	1.75E−05	7.22E−04	26.64	0.42	63.82	0.38
2	64	364	8.00E−06	3.73E−04	153.21	2.51	60.96	0.46
3	8	128	6.79E−06	4.23E−06	2.19	0.04	58.92	0.36
3	16	256	3.73E−08	2.67E−07	22.90	0.38	61.02	0.42
3	32	512	7.56E−08	6.22E−08	214.17	3.41	62.74	0.69
3	64	1024	1.69E−08	9.01E−09	1911.74	29.26	65.34	1.70
4	8	128	1.07E−07	2.40E−06	2.29	0.04	57.21	0.36
4	16	256	5.16E−07	1.22E−07	28.10	0.46	61.37	0.45
4	32	512	4.19E−08	5.90E−08	275.75	4.37	63.16	0.79
4	64	1024	1.11E−08	6.46E−09	2509.67	38.55	65.10	2.13
5	8	128	6.50E−06	1.06E−06	2.17	0.04	57.84	0.37
5	16	256	1.53E−04	1.07E−07	32.14	0.53	60.92	0.47
5	32	512	2.84E−08	6.07E−08	333.62	5.31	62.78	0.90
5	64	1024	1.04E−08	5.54E−09	3083.75	47.94	64.32	2.56
6	8	128	6.70E−05	9.72E−07	1.77	0.03	57.63	0.38
6	16	256	4.27E−07	7.13E−08	35.05	0.58	60.59	0.50
6	32	512	7.56E−07	7.14E−08	387.05	6.19	62.56	1.01
6	64	1024	1.02E−08	4.73E−09	3666.74	57.06	64.26	2.99

[18]

$$\begin{cases} y_t = V(t,S_t) = S_t^1 N(d_1) - S_t^2 N(d_2), \\ z_t = A^\top \nabla V S_t, \\ d_{1/2} = \frac{\ln(\frac{S_t^1}{S_t^2}) \pm \frac{\tilde{\sigma}^2}{2}(T-t)}{\tilde{\sigma}\sqrt{T-t}}, \end{cases}$$

where $\tilde{\sigma} = \sqrt{\sigma_1^2 + \sigma_2^2 - 2\sigma_1\sigma_2\rho}$ and $\nabla V S_t = (\frac{\partial V}{\partial S_t^1} S_t^1, \frac{\partial V}{\partial S_t^2} S_t^2)^\top$. In this example, we consider $T = 0.1$, $S_0^1 = S_0^2 = 100$, $r = 0.05$, $\mu_1 = \mu_2 = 0.1$, $\sigma_1 = 0.25$, $\sigma_2 = 0.3$ and $\rho = 0.0$ with the solution $(y_0, (z_0^1, z_0^2)) \doteq (15.48076, (14.3510, -12.6779))$. The results are presented in Table 11 for the same parameters as in Example 4. The highest speedup is ca. 64×, which requires again a large memory of ca. 3 GB. The plots of $\log_{10}(\lvert y_{0,0} - y_0^0 \rvert)$, $\log_{10}(\lvert z_{0,0} - z_0^0 \rvert)$ and speedup with respect to N for $K = 1, \ldots, 6$ in Fig. 6.

Note that the speedup is higher than in Example 4 since we have additionally the forward SDE and more work is conducted from the GPU threads. But we could not optimize further as the main constraint is the GPU memory. In Table 12 we present the performance of the main kernels. Our results show that the multistep scheme [27] with GPU computing performs very well also for the 2-dimensional case. Increasing the dimension becomes difficult when considering a single GPU due to memory constraint. However, this approach can be applied to a cluster of GPUs, where the possibility to achieve even higher speedups is enormous. This highlights the fact that the implementation of cluster GPU computing in the multistep scheme [27] offers very high accurate results in low computation time even for high dimensional problems.



(a) Plot of y error as a function of time layers N for $K = 1, \cdots, 6$.

(b) Plot of z error as a function of time layers N for $K = 1, \cdots, 6$.

(c) Plot of speedup as a function of time layers N for $K = 1, \cdots, 6$.

Figure 6 Plots of the results for Example 5

Table 12 Performance of the main kernels for Example 5

Time (%)	Time (s)	Kernel name
84.77	51.58	main_body
11.91	7.24	swap_coeff
3.00	1.83	find_coeff
0.14	0.88	swap_val

6 Conclusions

In this work we parallelized the multistep method developed in [27] for solving BSDEs on GPU. Firstly, we analyzed the algorithm and presented approaches for reduction of computation time. The most important reduction effort was the optimal operation to find the location of the interpolated values. It was essential for the reduction of the computational time. For a further acceleration, we have investigated how to optimize the application after finding the performance bottlenecks and applying optimization techniques. Our numerical results have shown that the multistep scheme is well suited on massively parallel GPU computing and very efficient for real-time applications such as option pricing and their risk management. Our parallelization strategy should work for other multistep schemes as well, and make those schemes be more useful in practice. Using sparse grids and cluster GPU computing to solve higher dimensional problems is the task for our ongoing work.

Appendix: Detailed derivation of reference equations

We derive the reference equations for y and z in Sect. 2.2. To receive the reference equation for y, we need to obtain the adaptability. Therefore, we take the conditional expectation $E^x_{t_n}[\cdot]$ in (6) and obtain

$$y_{t_n} = E^x_{t_n}[y_{t_{n+k}}] + \int_{t_n}^{t_{n+k}} E^x_{t_n}\big[f(s, y_s, z_s)\big]\, ds, \tag{11}$$

where martingale property of Itô integral is used. To approximate the integral in (11), Teng et al. [27] used the cubic spline polynomial to approximate the integrand. Based on the support points $(t_{n+j}, E^x_{t_n}[f(t_{n+j}, y_{t_{n+j}}, z_{t_{n+j}})])$, $j = 0, \ldots, K_y$, we have

$$\int_{t_n}^{t_{n+k}} E^x_{t_n}\big[f(s, y_s, z_s)\big]\, ds = \int_{t_n}^{t_{n+k}} \tilde{S}^{t_n,x}_{K_y}(s)\, ds + R^n_y,$$

where the cubic spline interpolant is given as

$$\tilde{S}_{K_y}^{t_n,x}(s) = \sum_{j=0}^{K_y-1} \tilde{s}_{K_y}^{t_n,x,j}(s),$$

where

$$\tilde{s}_{K_y}^{t_n,x,j}(s) = a_j^y + b_j^y(s - t_{n+j}) + c_j^y(s - t_{n+j})^2 + d_j^y(s - t_{n+j})^3$$

with

$$s \in [t_{n+j}, t_{n+j+1}], \quad j = 0, \dots, K_y - 1.$$

Obviously, the residual reads

$$R_y^n = \int_{t_n}^{t_{n+k}} \left(E_{t_n}^x\left[f(s, y_s, z_s)\right] - \tilde{S}_{K_y}^{t_n,x}(s)\right) ds.$$

We calculate

$$\begin{aligned}\int_{t_n}^{t_{n+k}} \tilde{S}_{K_y}^{t_n,x}(s)\, ds &= \int_{t_n}^{t_{n+k}} \sum_{j=0}^{K_y-1} \tilde{s}_{K_y}^{t_n,x,j}(s)\, ds \\ &= \sum_{j=0}^{K_y-1} \int_{t_n}^{t_{n+k}} \tilde{s}_{K_y}^{t_n,x,j}(s)\, ds \\ &= \sum_{j=0}^{K_y-1} \int_{t_{n+j}}^{t_{n+j+1}} \tilde{s}_{K_y}^{t_n,x,j}(s)\, ds.\end{aligned}$$

and obtain the reference equation for y as given in Sect. 2.2

$$y_{t_n} = E_{t_n}^x[y_{t_{n+k}}] + \sum_{j=0}^{K_y-1}\left[a_j^y \Delta t + \frac{b_j^y \Delta t^2}{2} + \frac{c_j^y \Delta t^3}{3} + \frac{d_j^y \Delta t^4}{4}\right] + R_y^n.$$

To receive the reference equation for the z process, we use l instead of k in (11), multiply both sides by $\Delta W_{t_{n+l}}$, take the conditional expectation $E_{t_n}^x[\cdot]$ to obtain

$$0 = E_{t_n}^x[y_{t_{n+l}} \Delta W_{t_{n+l}}] + \int_{t_n}^{t_{n+l}} E_{t_n}^x\left[f(s, y_s, z_s)\Delta W_s\right] ds - \int_{t_n}^{t_{n+l}} E_{t_n}^x[z_s]\, ds,$$

where Itô isometry is used. Again, with the cubic spline interpolation and the relation

$$\begin{aligned}E_{t_n}^x[y_{t_{n+l}} \Delta W_{t_{n+l}}] &= \frac{1}{\sqrt{2\pi l \Delta t}} \int_{-\infty}^{\infty} u(t_{n+l}, x+v) \exp\left(-\frac{v^2}{2l\Delta t}\right) dv, \\ &= \frac{l\Delta t}{\sqrt{2\pi l \Delta t}} \int_{-\infty}^{\infty} \frac{\partial u}{\partial x}(t_{n+l}, x+v) \exp\left(-\frac{v^2}{2l\Delta t}\right) dv, \\ &= l\Delta t E_{t_n}^x[z_{t_{n+l}}],\end{aligned}$$

we obtain the reference equation for z process

$$0 = l\Delta t E^x_{t_n}[z_{t_{n+l}}] + \sum_{j=0}^{K_z-1}\left[a_j^{z_1}\Delta t + \frac{b_j^{z_1}\Delta t^2}{2} + \frac{c_j^{z_1}\Delta t^3}{3} + \frac{d_j^{z_1}\Delta t^4}{4}\right]$$
$$- \sum_{j=0}^{K_z-1}\left[a_j^{z_2}\Delta t + \frac{b_j^{z_2}\Delta t^2}{2} + \frac{c_j^{z_2}\Delta t^3}{3} + \frac{d_j^{z_2}\Delta t^4}{4}\right] + R_z^n.$$

Acknowledgements
We thank the anonymous reviewer for the comments, which helped us to improve the manuscript.

Funding
This research received no external funding.

Abbreviations
BSDE, Backward Stochastic Differential Equation; GPU, Graphics Processing Unit; CUDA, Compute Unified Device Architecture; PDE, Partial Differential Equation; FBSDE, Forward Backward Stochastic Differential Equation; BiCGSTAB, Biconjugate Gradient Stabilized; cuBLAS, CUDA Basic Linear Algebra Subroutine; cuSPARSE, CUDA Sparse; CPU, Central Processing Unit; nvprof, NVIDIA Command-line Profiler.

Availability of data and materials
Not applicable.

Declarations

Competing interests
The authors declare that they have no competing interests.

Authors' contributions
LK and LT conceived the presented idea. LK was responsible for the implementation on GPUs. Both authors contributed to the writing and numerical studies. Both authors read and approved the final manuscript.

Publisher's Note

Received: 20 July 2021 Accepted: 29 December 2021 Published online: 28 January 2022

References

1. Ankirchner S, Blanchet-Scalliet C, Eyraud-Loisel A. Credit risk premia and quadratic bsdes with a single jump. Int J Theor Appl Finance. 2010;13(07):1103–29.
2. Bender C, Zhang J. Time discretization and Markovian iteration for coupled fbsdes. Ann Appl Probab. 2008;18(1):143–77.
3. Binder A, Jadhav O, Mehrmann V. Model order reduction for the simulation of parametric interest rate models in financial risk analysis. J Math Ind. 2021;11(1):1.
4. Bouchard B, Touzi N. Discrete-time approximation and Monte-Carlo simulation of backward stochastic differential equations. Stoch Process Appl. 2004;111(2):175–206.
5. Cheng J, Grossman M, McKercher T. Professional CUDA c programming. New York: Wiley; 2014.
6. Crisan D, Manolarakis K. Solving backward stochastic differential equations using the cubature method: application to nonlinear pricing. SIAM J Financ Math. 2012;3(1):534–71.
7. Dai B, Peng Y, Gong B. Parallel option pricing with BSDE method on GPU. In: 2010 ninth international conference on grid and cloud computing. Los Alamitos: IEEE Comput. Soc.; 2010.
8. Fu Y, Zhao W, Zhou T. Efficient spectral sparse grid approximations for solving multi-dimensional forward backward sdes. Discrete Contin Dyn Syst, Ser B. 2017;22(9):3439.
9. Gobet E, Labart C. Solving bsde with adaptive control variate. SIAM J Numer Anal. 2010;48(1):257–77.
10. Gobet E, Lemor JP, Warin X. A regression-based Monte Carlo method to solve backward stochastic differential equations. Ann Appl Probab. 2005;15(3):2172–202.
11. Gobet E, López-Salas JG, Turkedjiev P, Vázquez C. Stratified regression Monte-Carlo scheme for semilinear PDEs and BSDEs with large scale parallelization on GPUs. SIAM J Sci Comput. 2016;38(6):C652–77.
12. Howlett J, Abramowitz M, Stegun IA. Handbook of mathematical functions. Math Gaz. 1966;50(373):358.
13. Kapllani L, Teng L, Ehrhardt M. A multistep scheme to solve backward stochastic differential equations for option pricing on gpus. In: International conference on variability of the sun and sun-like stars: from asteroseismology to space weather. Berlin: Springer; 2019. p. 196–208.

14. Karoui NE, Peng S, Quenez MC. Backward stochastic differential equations in finance. Math Finance. 1997;7(1):1–71.
15. Kreiss HO, Thomée V, Widlund O. Smoothing of initial data and rates of convergence for parabolic difference equations. Commun Pure Appl Math. 1970;23(2):241–59.
16. Lemor JP, Gobet E, Warin X. Rate of convergence of an empirical regression method for solving generalized backward stochastic differential equations. Bernoulli. 2006;12(5):889–916.
17. Ma J, Shen J, Zhao Y. On numerical approximations of forward-backward stochastic differential equations. SIAM J Numer Anal. 2008;46(5):2636–61.
18. Margrabe W. The value of an option to exchange one asset for another. J Finance. 1978;33(1):177.
19. Nudart D, Schoutens W. Backward stochastic differential equations and feynman-kac formula for levy processes, with applications in finance. Bernoulli. 2001;**7**(5).
20. Pardoux E, Peng S. Adapted solution of a backward stochastic differential equation. Syst Control Lett. 1990;14(1):55–61.
21. Peng S, Wang F. Bsde, path-dependent pde and nonlinear Feynman–Kac formula. Sci China Math. 2016;59:19–36.
22. Peng Y, Gong B, Liu H, Dai B. Option pricing on the GPU with backward stochastic differential equation. In: 2011 fourth international symposium on parallel architectures, algorithms and programming. Los Alamitos: IEEE Comput. Soc.; 2011.
23. Pham H. Feynman–Kac representation of fully nonlinear pdes and applications. 2014.
24. Ruijter MJ, Oosterlee CW. A Fourier cosine method for an efficient computation of solutions to bsdes. SIAM J Sci Comput. 2015;37(2):A859–89.
25. Ruijter MJ, Oosterlee CW. Numerical Fourier method and second-order Taylor scheme for backward sdes in finance. Appl Numer Math. 2016;103:1–26.
26. Teng L. A review of tree-based approaches to solve forward-backward stochastic differential equations. J Comput Finance. 2019. In press.
27. Teng L, Lapitckii A, Günther M. A multi-step scheme based on cubic spline for solving backward stochastic differential equations. Appl Numer Math. 2019.
28. van der Vorst HA. Bi-cgstab: a fast and smoothly converging variant of bi-cg for the solution of nonsymmetric linear systems. SIAM J Sci Comput. 1992;13(2):631–44.
29. Zhang J et al. A numerical scheme for bsdes. Ann Appl Probab. 2004;14(1):459–88.
30. Zhao W, Chen L, Peng S. A new kind of accurate numerical method for backward stochastic differential equations. SIAM J Sci Comput. 2006;28(4):1563–81.
31. Zhao W, Fu Y, Zhou T. New kinds of high-order multistep schemes for coupled forward backward stochastic differential equations. SIAM J Sci Comput. 2014;36(4):A1731–51.
32. Zhao W, Zhang G, Ju L. A stable multistep scheme for solving backward stochastic differential equations. SIAM J Numer Anal. 2010;48(4):1369–94.