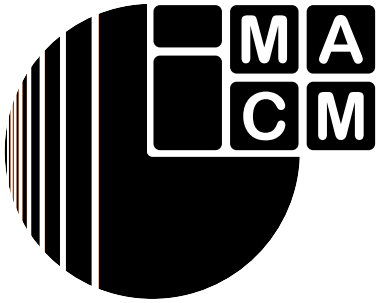


Bergische Universität Wuppertal

Fachbereich Mathematik und Naturwissenschaften

Institute of Mathematical Modelling, Analysis and Computational Mathematics (IMACM)

Preprint BUW-IMACM 19/28

L. Kapllani and L. Teng

Multistep schemes for solving backward stochastic differential equations on GPU

October 9, 2019

http://www.math.uni-wuppertal.de

Preprint – Preprint – Preprint – Preprint – Preprint – Preprint

Multistep schemes for solving backward stochastic differential equations on GPU

Lorenc Kapllani, Long Teng

Lehrstuhl für Angewandte Mathematik und Numerische Analysis,
Fakultät für Mathematik und Naturwissenschaften,
Bergische Universität Wuppertal, Gaußstr. 20,
42119 Wuppertal, Germany

Abstract

The goal of this work is to parallelize the multistep scheme for the numerical approximation of the backward stochastic differential equations (BSDEs) in order to achieve both, a high accuracy and a reduction of the computation time as well. In the multistep scheme the computations at each 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 for reducing the computation time, using a uniform domain. Finally, some examples with financial applications are provided to demonstrate the achieved acceleration on GPUs.

Keywords *backward stochastic differential equations, multistep scheme, GPU computing, CUDA, option pricing*

1 Introduction

In this work we parallelize the multistep scheme developed in [Teng et al., 2018] to approximate numerically the solution of the following (decoupled) *forward backward stochastic differential equation (FBSDE)*:

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

where $X_t, a \in \mathcal{R}^n$, b is a $n \times d$ matrix, W_t is a d-dimensional Brownian motion, $f\left(t, X_t, y_t, z_t\right) : [0,T] \times \mathcal{R}^n \times \mathcal{R}^m \times \mathcal{R}^{m \times d} \to \mathcal{R}^m$ is the driver function and ξ is the terminal condition. The terminal condition y_T depends on the final value of a forward stochastic differential equation (SDE). For $a = 0$ and $b = 1$, namely $X_t = W_t$, one obtains a *backward stochastic differential equation (BSDE)* of the form

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

where $y_t \in \mathcal{R}^m$ and $f\left(t, y_t, z_t\right) : [0,T] \times \mathcal{R}^m \times \mathcal{R}^{m \times d} \to \mathcal{R}^m$. In the sequel of this work, we investigate the acceleration of numerical scheme developed in [Teng et al., 2018] for solving (2).

Preprint – Preprint – Preprint – Preprint – Preprint – Preprint

Note that the developed schemes can be applied also for solving (1), where the general Markovian diffusion X_t can be approximated, e.g., by using the Euler-Scheme.

The existence and uniqueness of the solution of (2) are proven by Pardoux and Peng [Pardoux and Peng, 1990]. Peng [Peng, 1991] obtained a direct relation between FBSDEs and partial differential equations (PDEs). Based on this relationship, many numerical methods are proposed, e.g., probabilistic methods in [Bender and Steiner, 2012, Bouchard and Touzi, 2004, Gobet et al., 2005, Lemor et al., 2006, Zhao et al., 2006], tree-based methods in [Crisan and Manolarakis, 2012, Teng, 2018] etc. El Karoui et al. [El Karoui et al., 1997] showed that the solution of a linear BSDE is in fact the pricing and hedging strategy of an option derivative. This is the first claim of the application of BSDEs in finance.

In the field of financial mathematics, the approach with BSDEs has a couple of advantages compared to the standard approach with FSDEs. Firstly, many market models can be presented in terms of BSDEs (or FBSDEs), e.g., local volatility models [Labart and Lelong, 2011], stochastic volatility models [Fahim et al., 2011], jump-diffusion models [Eyraud-Loisel, 2005], defaultable options [Ankirchner et al., 2010] etc. Secondly, BSDEs can also be used in incomplete markets [El Karoui et al., 1997] to solve the maximization problem of difference between the value of the super-replicating portfolio and the option value. Another advantage of using BSDEs is that one does not need to switch to the so-called risk-neutral measure for pricing financial options in complete markets. Therefore, BSDEs represent a more intuitive and more understandable way for pricing problems.

In general, the solution of BSDEs cannot be established in a closed form. Therefore, a numerical method is mandatory. There are two main classes of numerical methods for approximating the solution of BSDEs. The first class is proposed based on the relation between the BSDE and its related PDE. The other class contains approaches which are developed direchtly based on the BSDE. The θ-discretization method has been most widely used, a second order convergence rate could be achieved with Crank Nicolson type. For a higher convergence rate, the authors in [Zhao et al., 2010] proposed the multistep scheme in which the integrands are approximated by using Lagrange interpolating polynomials. For a better stability and the admission of more time levels, this multistep scheme has been generalised in [Teng et al., 2018] with spline instead of Lagrange interpolating polynomials. This kind of multistep schemes are computationally not efficient, since the values of the integrands at multiple time levels need to be known. Fortunately, computations in the multistep scheme are independent at each grid point. This fact motivates us to use massively parallel GPU computing to make these high-order accurate method more useful in practice.

Many acceleration strategies based on GPU computing have been developed for pricing problems, however, a very little of them are BSDE-based approach. In [Dai et al., 2010] a linear BSDE is solved on the GPU with the θ-scheme method. They analyzed the effects of the thread number per block to increase the speedup. The parallel program with CUDA achieved high speedups and showed that the GPU architecture is well suited for solving the BSDEs in parallel. Peng et al. [Peng et al., 2011] developed acceleration strategies for option pricing with non-linear BSDEs using a binomial lattice based method. To increase the speedup, they reduce the global memory access frequency by avoiding the kernel invocation on each time step. Also, due to the load imbalance produced by the binomial grid, they provided load-balanced strategies and showed that the acceleration algorithms exhibit very high speedup over the sequential CPU implementation and therefore suitable for real-time application. Peng et al. [Peng et al., 2014] considered solving high dimensional BSDEs on GPUs with application in high dimensional American option pricing. A *Least Square Monte-Carlo (LSMC)* method based numerical algorithm is studied, and

Preprint – Preprint – Preprint – Preprint – Preprint – Preprint – Preprint

summarised in four phases. Multiple factors which affect the performance (task allocation, data store/access strategies and the thread synchronisation) are considered. Results showed much better performance than the CPU version. Gobet et al. [Gobet et al., 2016] designed a new algorithm for solving BSDEs based on LSMC. Due to stratification, the algorithm is very efficient especially for large scale simulations. They showed big speedups even in high dimensions.

Next we introduce some preliminaries needed to understand the multistep scheme. Let $(\Omega, \mathcal{F}, \mathbb{P}, \{\mathcal{F}_t\}_{0\leq t\leq T})$ be a complete, filtered probability space. In this space a standard d-dimensional Brownian motion W_t is defined, such that the filtration $\{\mathcal{F}_t\}_{0\leq t\leq T}$ is the natural filtration of W_t. We define $\|\cdot\|$ as the standard Euclidean norm in the Euclidean space $\mathcal{R}^m$ or $\mathcal{R}^{m\times d}$ and $L^2 = L^2_{\mathcal{F}}\left(0, T; \mathcal{R}^d\right)$ the set of all $\mathcal{F}_t$-adapted and square integrable processes valued in $\mathcal{R}^d$. A pair of processes $(y_t, z_t) : [0, T] \times \Omega \to \mathcal{R}^m \times \mathcal{R}^{m\times d}$ is the solution of BSDE (2) if it is $\mathcal{F}_t$-adapted, square integrable, and satisfies (2) in the sense of

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

where $f\left(t, y_t, z_t\right) : [0, T] \times \mathcal{R}^m \times \mathcal{R}^{m\times d} \to \mathcal{R}^m$ 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 [Pardoux and Peng, 1990]. Let us consider the following:

$$y_t = u\left(t, W_t\right), \quad z_t = \nabla u\left(t, W_t\right) \quad \forall t \in [0, T), \tag{4}$$

where ∇u denotes the derivative of $u\left(t, x\right)$ with respect to the spatial variable x and $u\left(t, x\right)$ is the solution of the following (backward in time) parabolic PDE:

$$\frac{\partial u}{\partial t} + \frac{1}{2}\sum_{i=1}^{d} \frac{\partial^2 u}{\partial x_i^2} + f\left(t, u, \nabla u\right) = 0, \tag{5}$$

with the terminal condition $u\left(T, x\right) = \phi(x)$. Under regularity conditions, the PDE (5) possess a unique solution $u\left(t, x\right)$. Therefore, for $\xi = \phi\left(W_T\right)$, the pair (y_t, z_t) is the unique solution of BSDE (3).

Now we introduce some notation which is used in the Sections that follow. Let $\mathcal{F}_s^{t,x}$ for $t \leq s \leq T$ be a σ-field generated by the Brownian motion $\{x + W_r - W_t, t \leq r \leq s\}$ starting from the time-space point (t, x). We define $E_s^{t,x}\left[X\right]$ as the conditional expectation of the random variable X under the filtration $\mathcal{F}_s^{t,x}$, i.e. $E_s^{t,x}\left[X\right] = E\left[X|\mathcal{F}_s^{t,x}\right]$.

In the next Section, we introduce the multistep scheme. In Section 3, we present our algorithmic framework. Section 4 is devoted to strategies of parallel GPU computing using CUDA. In Section 5, we illustrate our findings with some examples including financial applications. Finally, Section 6 concludes this work.

2 The multistep scheme

In this Section we present the multistep scheme [Teng et al., 2018].

2.1 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

Preprint – Preprint – Preprint – Preprint – Preprint – Preprint

and K_y be two positive integers such that $1 \leq k \leq K_y \leq N$. The BSDE (3) can be expressed as

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

In order to approximate y_{t_n} based on the later information $[t_n, t_{n+k}]$, we need to obtain the adaptability. Therefore, we take the conditional expectation $E_{t_n}^x[\cdot]$ in (6) and obtain

$$y_{t_n} = E_{t_n}^x\left[y_{t_{n+k}}\right] + \int_{t_n}^{t_{n+k}} E_{t_n}^x\left[f\left(s, y_s, z_s\right)\right] ds. \tag{7}$$

In order to approximate the integral in (7), Teng et al. [Teng et al., 2018] used the cubic spline polynomial to approximate that integrand. Based on the support points $\left(t_{n+j}, E_{t_n}^x\left[f\left(t_{n+j}, y_{t_{n+j}}, z_{t_{n+j}}\right)\right]\right)$, $j = 0, \cdots, K_y$, we have

$$\int_{t_n}^{t_{n+k}} E_{t_n}^x\left[f\left(s, y_s, z_s\right)\right] ds = \int_{t_n}^{t_{n+k}} \tilde{S}_{K_y}^{t_n,x}\left(s\right)\, ds + R_y^n, \tag{8}$$

where the cubic spline interpolant is given as

$$\tilde{S}_{K_y}^{t_n,x}\left(s\right) = \sum_{j=0}^{K_y-1} \tilde{s}_{K_y}^{t_n,x,j}\left(s\right), \tag{9}$$

where

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

with

$$s \in \left[t_{n+j}, t_{n+j+1}\right], j = 0, \cdots, K_y - 1.$$

Obviously, the residual reads

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

We calculate

$$\begin{aligned}
\int_{t_n}^{t_{n+k}} \tilde{S}_{K_y}^{t_n,x}\left(s\right)\, ds &= \int_{t_n}^{t_{n+k}} \sum_{j=0}^{K_y-1} \tilde{s}_{K_y}^{t_n,x,j}\left(s\right)\, ds \\
&= \sum_{j=0}^{K_y-1} \int_{t_n}^{t_{n+k}} \tilde{s}_{K_y}^{t_n,x,j}\left(s\right)\, ds \\
&= \sum_{j=0}^{K_y-1} \int_{t_{n+j}}^{t_{n+j+1}} \tilde{s}_{K_y}^{t_n,x,j}\left(s\right)\, ds \\
&= \sum_{j=0}^{K_y-1} \left[a_j^y \Delta t + \frac{b_j^y \Delta t^2}{2} + \frac{c_j^y \Delta t^3}{3} + \frac{d_j^y \Delta t^4}{4}\right].
\end{aligned} \tag{10}$$

and obtain the reference equation for y as

$$y_{t_n} = E_{t_n}^x\left[y_{t_{n+k}}\right] + \sum_{j=0}^{K_y-1} \left[a_j^y \Delta t + \frac{b_j^y \Delta t^2}{2} + \frac{c_j^y \Delta t^3}{3} + \frac{d_j^y \Delta t^4}{4}\right] + R_y^n. \tag{11}$$

Preprint – Preprint – Preprint – Preprint – Preprint – Preprint

In order to obtain the reference equation for the z process, we follow the similar approach. Let $\Delta W_s = W_s - W_{t_n}$ for $s \geq t_n$. Then ΔW_s is a standard Brownian motion with the zero mean and the standard deviation $\sqrt{s-t_n}$. Let l and K_z be two positive integers such that $1 \leq l \leq K_z \leq N$. Using l instead of k in (6), multiplying both sides by $\Delta W_{t_{n+l}}$, taking the conditional expectation $E^x_{t_n}[\cdot]$ and using the Itô isometry we obtain

$$0 = E^x_{t_n}\left[y_{t_{n+l}} \Delta W_{t_{n+l}}\right] + \int_{t_n}^{t_{n+l}} E^x_{t_n}\left[f\left(s, y_s, z_s\right) \Delta W_s\right] ds - \int_{t_n}^{t_{n+l}} E^x_{t_n}\left[z_s\right] ds. \tag{12}$$

Using again the cubic spline interpolation to approximate the two integrals in (12) and the relation

$$E^x_{t_n}\left[y_{t_{n+l}} \Delta W_{t_{n+l}}\right] = l\Delta t E^x_{t_n}\left[z_{t_{n+l}}\right],$$

we obtain the reference equation for z process

$$\begin{aligned} 0 = l\Delta t E^x_{t_n}\left[z_{t_{n+l}}\right] &+ \sum_{j=0}^{K_z-1}\left[a_j^{z_1}\Delta t + \frac{b_j^{z_1}\Delta t^2}{2} + \frac{c_j^{z_1}\Delta t^3}{3} + \frac{d_j^{z_1}\Delta t^4}{4}\right] \\ &- \sum_{j=0}^{K_z-1}\left[a_j^{z_2}\Delta t + \frac{b_j^{z_2}\Delta t^2}{2} + \frac{c_j^{z_2}\Delta t^3}{3} + \frac{d_j^{z_2}\Delta t^4}{4}\right] + R^n_{z_1} + R^n_{z_2}. \end{aligned} \tag{13}$$

The results above can be straightforwardly generalized to the d-dimensional case as

$$\begin{aligned} y^{\tilde{m}}_{t_n} &= E^x_{t_n}\left[y^{\tilde{m}}_{t_{n+k}}\right] + \sum_{j=0}^{K_y-1}\left[a_j^{y,\tilde{m}}\Delta t + \frac{b_j^{y,\tilde{m}}\Delta t^2}{2} + \frac{c_j^{y,\tilde{m}}\Delta t^3}{3} + \frac{d_j^{y,\tilde{m}}\Delta t^4}{4}\right] + R^{n,\tilde{m}}_y, \\ 0 &= l\Delta t E^x_{t_n}\left[z^{\tilde{m},\tilde{d}}_{t_{n+l}}\right] + \sum_{j=0}^{K_z-1}\left[a_j^{z_1,\tilde{m},\tilde{d}}\Delta t + \frac{b_j^{z_1,\tilde{m},\tilde{d}}\Delta t^2}{2} + \frac{c_j^{z_1,\tilde{m},\tilde{d}}\Delta t^3}{3} + \frac{d_j^{z_1,\tilde{m},\tilde{d}}\Delta t^4}{4}\right] \\ &\quad - \sum_{j=0}^{K_z-1}\left[a_j^{z_2,\tilde{m},\tilde{d}}\Delta t + \frac{b_j^{z_2,\tilde{m},\tilde{d}}\Delta t^2}{2} + \frac{c_j^{z_2,\tilde{m},\tilde{d}}\Delta t^3}{3} + \frac{d_j^{z_2,\tilde{m},\tilde{d}}\Delta t^4}{4}\right] + R^{n,\tilde{m},\tilde{d}}_{z_1} + R^{n,\tilde{m},\tilde{d}}_{z_2}, \end{aligned} \tag{14}$$

where $\tilde{m} = 1, 2, \cdots, m$ and $\tilde{d} = 1, 2, \cdots, d$.

The unknown coefficients in (14) are found using cubic spline conditions. For instance, for the y process (in 1-dimension), using support points $\left(t_{n+j}, E^x_{t_n}\left[f\left(t_{n+j}, y_{t_{n+j}}, z_{t_{n+j}}\right)\right]\right)$, $j = 0, \cdots, K_y$, the conditions are

$$\begin{cases} \tilde{S}^{t_n,x}_{K_y}\left(t_{n+j}\right) & = E^x_{t_n}\left[f\left(t_{n+j}, y_{t_{n+j}}, z_{t_{n+j}}\right)\right], \qquad j = 0, \cdots, K_y \\ \tilde{s}^{t_n,x,j}_{K_y}\left(t_{n+j}\right) & = \tilde{s}^{t_n,x,j+1}_{K_y}\left(t_{n+j}\right), \qquad j = 0, \cdots, K_y - 2 \\ \tilde{s}'^{\,t_n,x,j}_{K_y}\left(t_{n+j}\right) & = \tilde{s}'^{\,t_n,x,j+1}_{K_y}\left(t_{n+j}\right), \qquad j = 0, \cdots, K_y - 2 \\ \tilde{s}''^{\,t_n,x,j}_{K_y}\left(t_{n+j}\right) & = \tilde{s}''^{\,t_n,x,j+1}_{K_y}\left(t_{n+j}\right), \qquad j = 0, \cdots, K_y - 2 \end{cases} \tag{15}$$

For $K_y = 3$ and using not-a-knot cubic spline, the coefficients are calculated as follows. Consider the notation

$$g_{t_{n+j}} = E^x_{t_n}\left[f\left(t_{n+j}, y_{t_{n+j}}, z_{t_{n+j}}\right)\right].$$

Then

- For $\tilde{s}^{t_n,x,0}_{K_y}\left(s\right), s \in [t_n, t_{n+1}]$

$$
\begin{aligned}
a_0 &= g_{t_n}, \\
b_0 &= -\left(11g_{t_n} - 18g_{t_{n+1}} + 9g_{t_{n+2}} - 2g_{t_{n+3}}\right)/6\Delta t, \\
c_0 &= \left(2g_{t_n} - 5g_{t_{n+1}} + 4g_{t_{n+2}} - g_{t_{n+3}}\right)/2\Delta t^2, \\
d_0 &= -\left(g_{t_n} - 3g_{t_{n+1}} + 3g_{t_{n+2}} - g_{t_{n+3}}\right)/6\Delta t^3,
\end{aligned}
$$

- For $\tilde{s}_{K_y}^{t_n,x,1}(s), s \in [t_{n+1}, t_{n+2}]$

$$
\begin{aligned}
a_1 &= g_{t_{n+1}}, \\
b_1 &= -\left(2g_{t_n} + 3g_{t_{n+1}} - 6g_{t_{n+2}} + g_{t_{n+3}}\right)/6\Delta t, \\
c_1 &= \left(g_{t_n} - 2g_{t_{n+1}} + g_{t_{n+2}}\right)/2\Delta t^2, \\
d_1 &= -\left(g_{t_n} - 3g_{t_{n+1}} + 3g_{t_{n+2}} - g_{t_{n+3}}\right)/6\Delta t^3,
\end{aligned}
$$

- For $\tilde{s}_{K_y}^{t_n,x,2}(s), s \in [t_{n+2}, t_{n+3}]$

$$
\begin{aligned}
a_2 &= g_{t_{n+2}}, \\
b_2 &= \left(g_{t_n} - 6g_{t_{n+1}} + 3g_{t_{n+2}} + 2g_{t_{n+3}}\right)/6\Delta t, \\
c_2 &= \left(g_{t_n} - 2g_{t_{n+1}} + g_{t_{n+3}}\right)/2\Delta t^2, \\
d_2 &= -\left(g_{t_n} - 3g_{t_{n+1}} + 3g_{t_{n+2}} - g_{t_{n+3}}\right)/6\Delta t^3.
\end{aligned}
$$

Reference equation for y process can therefore be written as

$$
\begin{aligned}
y_{t_n} &= E_{t_n}^x\left[y_{t_{n+k}}\right] + \frac{3\Delta t}{8}g_{t_n} + \frac{9\Delta t}{8}g_{t_{n+1}} + \frac{9\Delta t}{8}g_{t_{n+2}} + \frac{3\Delta t}{8}g_{t_{n+3}} + R_y^n, \\
&= E_{t_n}^x\left[y_{t_{n+k}}\right] + \Delta t K_y \sum_{j=0}^{K_y} \gamma_{K_y,j}^k E_{t_n}^x\left[f\left(t_{n+j}, y_{t_{n+j}}, z_{t_{n+j}}\right)\right] + R_y^n,
\end{aligned}
$$

where

$$
\gamma_{K_y,0}^{K_y} = \gamma_{K_y,3}^{K_y} = \frac{1}{8}, \gamma_{K_y,1}^{K_y} = \gamma_{K_y,2}^{K_y} = \frac{3}{8}.
$$

In a similar way, the corresponding coefficients can be found for other choices of K_y. In [Teng et al., 2018], the authors have shown that the semidiscrete scheme are stable when

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

This is to say that the algorithm allows for arbitrary multiple time levels K_y and K_z. In Table 1 and 2, we present the coefficients up to 6 time levels.

We denote (y^n, z^n) as the approximation to (y_{t_n}, z_{t_n}), given random variables $\left(y^{N-i}, z^{N-i}\right)$, $i = 0, 1, \ldots, K-1$ with $K = \max\{K_y, K_z\}$, (y^n, z^n) can be found for $n = N-K, \ldots, 0$ such that

$$
\begin{aligned}
y^n &= E_{t_n}^x\left[y^{n+K_y}\right] + K_y \Delta t \sum_{j=0}^{K_y} \gamma_{K_y,j}^{K_y} E_{t_n}^x\left[f\left(t_{n+j}, y^{n+j}, z^{n+j}\right)\right] + R_y^n, \\
0 &= E_{t_n}^x\left[z^{n+1}\right] + \sum_{j=1}^{K_z} \gamma_{K_z,j}^1 E_{t_n}^x\left[f\left(t_{n+j}, y^{n+j}, z^{n+j}\right) \Delta W_{t_{n+j}}^\top\right] - \sum_{j=0}^{K_z} \gamma_{K_z,j}^1 E_{t_n}^x\left[z^{n+j}\right] + \frac{R_z^n}{\Delta t},
\end{aligned} \tag{16}
$$

Preprint – Preprint – Preprint – Preprint – Preprint – Preprint

Preprint – Preprint – Preprint – Preprint – Preprint – Preprint

Table 1: The coefficients $\left\{\gamma_{K_y,j}^{K_y}\right\}_{j=0}^{K_y}$ 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 $\left\{\gamma_{K_z,j}^{1}\right\}_{j=0}^{K_z}$ 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	$\frac{163}{448}$	$\frac{47}{56}$	$-\frac{129}{448}$	$\frac{3}{28}$	$-\frac{37}{1344}$	$\frac{1}{168}$	$-\frac{1}{1344}$

where $y^n = \left(y^{n,1}, y^{n,2}, \cdots, y^{n,m}\right)^T$, $z^n = \left(z^{n,\tilde{m},\tilde{d}}\right)_{m\times d}$ and $\Delta W_{t_{n+j}}^{\top} = \left(W_{t_{n+j}}^1, W_{t_{n+j}}^2, \cdots, W_{t_{n+j}}^d\right)^{\top} - \left(W_{t_n}^1, W_{t_n}^2, \cdots, W_{t_n}^d\right)^{\top}$. One can show that the local errors in (16) are given by

$$|R_y^n| = \mathcal{O}\left(\Delta t^5\right), \qquad |R_z^n| = \mathcal{O}\left(\Delta t^5\right) \tag{17}$$

provided that f and g are smooth enough. In (15) we need to divide by Δt to find the value of z process. Therefore, in order to balance time truncation errors, one might set $K_z = K_y + 1$. In the following, we only present the results of error analysis, for their proofs we refer to [Teng et al., 2018] and [Zhao et al., 2010].

Lemma 2.1. *The local estimates of the local truncation errors in* (16) *satisfy*

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

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

Theorem 2.1. *Suppose that the initial values satisfy*

$$\begin{cases} \max_{N-K_y+1\le n\le N} E\left[|y_{t_n} - y^n|\right] & = \mathcal{O}\left(\Delta t^{K_y+1}\right), for\ K_y = 1,2,3 \\ \max_{N-K_y+1\le n\le N} E\left[|y_{t_n} - y^n|\right] & = \mathcal{O}\left(\Delta t^4\right), for\ K_y > 3 \end{cases}$$

for sufficiently small time step Δt it can be shown that

$$\sup_{0\le n\le N} E\left[|y_{t_n} - y^n|\right] \le C\Delta t^{\min\{K_y+1,4\}}, \tag{18}$$

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

Preprint – Preprint – Preprint – Preprint – Preprint – Preprint

Theorem 2.2. *Suppose that the initial values satisfy*

$$\begin{cases} \max_{N-K_z+1\le n\le N} E\big[|z_{t_n}-z^n|\big] & = \mathcal{O}\left(\Delta t^{K_z}\right), for\ K_z = 1,2,3 \\ \max_{N-K_z+1\le n\le N} E\big[|z_{t_n}-z^n|\big] & = \mathcal{O}\left(\Delta t^{3}\right), for\ K_z > 3 \end{cases}$$

and the condition on the initial values in Theorem 2.1 is fulfilled. For sufficiently small time step Δt it can be shown that

$$\sup_{0\le n\le N} E\big[|z_{t_n}-z^n|\big] \le C\Delta t^{\min\{K_y+1,K_z,3\}}, \tag{19}$$

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

Remark 2.3. *If f does not depend on process z, the maximum order of convergence for y process is 4 and 3 for z process; If f depends on process z, the maximum order of convergence for y and z processes is 3*

2.2 The fully discrete scheme

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

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

where

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

Let $x_{\mathbf{i}} = \left(x_{i_1}^1, x_{i_2}^2, \cdots, x_{i_d}^d\right)$ for $\mathbf{i} = (i_1,i_2,\cdots,i_d)\in\mathcal{Z}^d$.

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 $(y_{\mathbf{i}}^{N-l}, z_{\mathbf{i}}^{N-l})$, $l=0,1,\ldots,K-1$ with $K=\max\{K_y,K_z\}$, $(y_{\mathbf{i}}^n, z_{\mathbf{i}}^n)$ can be found for $n = N-K,\ldots,0$ such that

$$\begin{aligned} y_{\mathbf{i}}^n &= \hat{E}_{t_n}^{x_{\mathbf{i}}}\big[\hat{y}^{n+K_y}\big] + K_y\Delta t\sum_{j=1}^{K_y} b_{K_y,j}^{K_y}\,\hat{E}_{t_n}^{x_{\mathbf{i}}}\big[f(t_{n+j},\hat{y}^{n+j},\hat{z}^{n+j})\big] \\ &\quad + K_y\Delta t b_{K_y,0}^{K_y} f(t_n, y_{\mathbf{i}}^n, z_{\mathbf{i}}^n), \\ 0 &= \hat{E}_{t_n}^{x_{\mathbf{i}}}\big[\hat{z}^{n+1}\big] + \sum_{j=1}^{K_z} b_{K_z,j}^1\,\hat{E}_{t_n}^{x_{\mathbf{i}}}\big[f(t_{n+j},\hat{y}^{n+j},\hat{z}^{n+j})\,\Delta W_{t_{n+j}}^{\top}\big] \\ &\quad - \sum_{j=1}^{K_z} b_{K_z,j}^1 \hat{E}_{t_n}^{x_{\mathbf{i}}}\big[\hat{z}^{n+j}\big] - b_{K_z,0}^1 z_{\mathbf{i}}^n. \end{aligned} \tag{20}$$

where $\hat{E}_{t_n}^{x_i}\big[\cdot\big]$ is used to denote the approximation of the conditional expectation. 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 points. The conditional expectation can be approximated as

$$\hat{E}_{t_n}^{x_i}\big[\hat{y}^{n+K_y}\big] = \frac{1}{\pi^{\frac{d}{2}}}\sum_{\Lambda=1}^{L}\omega_\Lambda\hat{y}^{n+K_y}\big(x_{\mathbf{i}} + \sqrt{2k\Delta t}\,a_\Lambda\big), \tag{21}$$

where $\hat{y}^{n+K_y}$ are interpolating values at the space points $\big(x_{\mathbf{i}} + \sqrt{2k\Delta t}\,a_\Lambda\big)$ based on y^{n+K_y} values, $(\omega_\Lambda, a_\Lambda)$ for $\Lambda = (\lambda_1,\lambda_2,\cdots,\lambda_d)$ are the weights and roots of the Hermite polynomial of degree L ([Abramowitz and Stegun, 1972]), $\omega_\Lambda = \prod_{\tilde{d}=1}^{d}\omega_{\lambda_{\tilde{d}}}$, $a_\Lambda = (a_{\lambda_1}, a_{\lambda_2},\cdots,a_{\lambda_d})$ and $\sum_{\Lambda=1}^{L} = \sum_{\lambda_1=1,\cdots,\lambda_d=1}^{L,\cdots,L}$. In the same way, one can express the other conditional expectations in (20).

Preprint – Preprint – Preprint – Preprint – Preprint – Preprint

3 The algorithmic framework

In this Section we present the alogithmic framework of the proposed numerical method.

3.1 The Algorithm

According to (20), we will consider the following three steps.

1. **Construct the time-space discrete domain.**
   We divide the time period $[0, T]$ into N time steps using $\Delta t = T/N$ and get $N + 1$ time layers and the space domain $\mathcal{R}^d$ as explained in Subsection 2.2 using step size Δx. We will use the truncated domains $[-8, 8]$ or $[-16, 16]$. 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.

2. **Calculate K initial solutions with $K = \max\{K_y, K_z\}$.**
   Since only the terminal value is given, one needs to generate the other $K - 1$ values. This can be done by running a 1-step scheme for $[t_{N-K+1}, t_{N-1}]$ with a smaller Δt such that the $K - 1$ produced initial values will have neglectable error.

3. **Calculate the numerical solution (y_0^0, z_0^0) backward using equation (20).**
   Note that the calculation for the y process is done implicitly by Picard iteration.

3.2 Preliminary considerations

For our numerical experiments we give the following remarks:

- When generating the non-grid points for the calculation of conditional expectations, some of them will be outside of the truncated domain. For these points, we take the values on the boundaries.

- Due to uniformity of the grid, one does not need to 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. 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 $A_y \in \mathcal{R}^{K \times \left(4^d \times M^d\right)}$, 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 the amount of work for the algorithm.

- There is a very important benefit from the uniformity of the grid. When we need to find the position of the non-grid points, a naive search algorithm is to loop over the grid points. In the worst case, an $\mathcal{O}\left(M^d\right)$ work is needed. Fortunately, this can be done in $\mathcal{O}(d)$, i.e., without for-loop. Recall that each new point is generated as $X_{\lambda_{\tilde{d}}} = x_{i_{\tilde{d}}} + \sqrt{2\Delta t k}\, a_{\lambda_{\tilde{d}}}$.

Preprint – Preprint – Preprint – Preprint – Preprint – Preprint

This means that taking int $\left(\frac{X_{\lambda_{\tilde{d}}}-x_{min}}{\Delta x}\right)$ for $x_{i_{\tilde{d}}} \in [x_{min}, x_{max}]$ and $M - \text{int}\left(\frac{X_{\lambda_{\tilde{d}}}-x_{min}}{\Delta x}\right)$ 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. This reduces substantially the total computation time, as it will be demonstrated in the numerical experiments.

3.3 The Parallel implementation

In this Section we present the naive parallelization of the multistep scheme. Nevertheless, we have kept into attention the optimal 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 is fully implemented in the device. Recall from (20) that the following steps are needed to calculate the approximated values on each time layer backward:

- **Generation of non-grid points $X_\Lambda = x_{\mathbf{i}} + \sqrt{2\Delta t k}\, a_\Lambda$.**
  In the uniform 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 for each space direction.

- **Calculation of the values $\hat{y}$ and $\hat{z}$ at the non-grid points.**
  This is the most time consuming part of the algorithm. For the 1-dimensional cases, we have considered the cubic spline interpolation. Since (20) involves the solution of two linear systems, the $BiCGSTAB$ iterative method is used since the matrix is tridiagonal. To apply the method, we consider the $cuBLAS$ and $cuSPARSE$ libraries. 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 last point in Subsection 3.2 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, forward and backward). 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 last point in Subsection 3.2 is created to find the values at non-grid points, where each thread calculates $m + m \times d$ values.

- **Calculation of the conditional expectations.**
  For the first conditional expectations in the right hand side of (20), we creat one kernel, where each thread calculates one value by using (21). Furthermore, we merged the calculation of three conditional expectation in one kernel, namely

$$\hat{E}^{x_{\mathbf{i}}}_{t_n}\big[\hat{z}^{n+j}\big], \quad \hat{E}^{x_{\mathbf{i}}}_{t_n}\big[f(t_{n+j}, \hat{y}^{n+j}, \hat{z}^{n+j})\big], \quad \hat{E}^{x_{\mathbf{i}}}_{t_n}\big[f(t_{n+j}, \hat{y}^{n+j}, \hat{z}^{n+j})\,\Delta W_{t_{n+j}}\big],$$

  for $j = 1, 2, \ldots, K$. This reduces the accessing of data multiple times from the global memory. Note that one thread calculates three values.

- **Calculation of the z values.**
  The second equation in (20) is used and each thread calculates $m \times d$ values.

- **Calculation of the y values.**
  The first equation in (20) is used and each thread calculates m values, using the Picard iterative process.

4 GPU computing and CUDA

In this Section, we discuss about GPU computing using CUDA. We start with CUDA programming and execution model and present an iterative process to optimize CUDA application.

4.1 CUDA programming and execution model

CUDA (Compute Unified Device Architecture) provides a framework for developing parallel general purpose applications on a GPU. At its core, there are three key abstractions: a hierarchy of thread groups, a hierarchy of memory and multiple thread level communication. These abstractions provide granular and coarse parallelism. Therefore, the application domain can be divided into sub-domains based on the data independence. Threads are organised into blocks of threads (threads within the block can communicate), grid of blocks and are executed in the SIMD fashion (a group of 32 threads called warps). Since the execution is based on warps (and scheduled from the warp schedulers; each GPU has a number of warp schedulers), the dimension of thread blocks gives different performances.

In a GPU, the largest and slowest memory is the global memory, which allows us to transfer data between the host (CPU) and the device (GPU) and is accessible for all threads. The shared memory is exclusive to thread blocks. The access is significantly faster than that of the global memory, and using this memory is profitable for optimisation tuning. In order to have an optimal application, each kernel created should be checked for the possible performance limiters. After the performance inhibitor is found, different techniques are considered to overcome the problem. Note that there can be a trade-off between different techniques. In the next Subsection, we present an iterative process to optimize the performance of the application in the GPU.

4.2 Iterative optimization process

The first version of a CUDA program is mostly not the optimal one. Therefore, we should access and identify the bottlenecks. There are three main limiting factors, memory bound, compute bound and latency bound. Therefore, we need to focus on efficient use of GPU memory bandwidth, compute resources and hiding of instruction and memory latency. To identify these factors, one can use CUDA profiling tools (NVIDIA Command-line Profiler or *nvprof* and Visual NVIDIA profiler or *nvvp*). Profile-driven optimization is an iterative process to optimize the program based on profile information. We have used the following iterative approach:

1. Apply 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

Preprint – Preprint – Preprint – Preprint – Preprint – Preprint

Preprint – Preprint – Preprint – Preprint – Preprint – Preprint – Preprint

The *nvprof* profiling tool enables the collection of a timeline of CUDA-related activities on both CPU and GPU, including kernel execution, memory transfers, memory set, CUDA API calls and events or metrics for CUDA kernels. Profiling options are provided through command-line. It is used for the first and second step of profile optimization process. The *nvvp* is a graphical tool with two main features, a timeline to display CPU and GPU activity and automatic performance analysis to help in identifying optimization opportunities. It provides a guided analysis, and guides one step-by-step through analysis in the entire application. In this mode, it helps on understanding of the likely performance limiters and optimization opportunities, including CUDA application analysis, performance-critical kernels, compute, bandwidth or latency bound and compute resources. It is used for the third profile optimization process.

After that the performance inhibitor is found, we consider different techniques to overcome the problem (step 4). The techniques usually are related with exposing sufficient parallelizm, optimizing memory access and optimizing instruction execution. There are two ways to increase parallelism: keeping more concurrent warps active within an Streaming Multiprocessor (SM) and assigning more independent work to each thread or warp. To keep more concurrent warps, we change the grid configuration (e.g. by decreasing the block dimension, one can have more blocks per SM etc.). To assign more independent work, we use unrolling techniques (an operation is split into multiple operations). Note that an increase of parallelism can be limited by compute resources such as shared memory and registers. A 100% occupancy can't be reached in such case. Therefore, a trade of must be found. The goal of memory access optimization is to maximize memory bandwidth utilization, with the focus on memory access patterns (maximize the use of bytes that travel on the bus) and sufficient concurrent memory accesses (hide memory latency). The best access pattern to global memory is aligned and coalesced access. There are several ways to optimize instruction execution, including hiding latency by keeping sufficient active warps or assigning more independent work to a thread and avoiding divergent execution paths within a warp. For the first two, we can use the same techniques as in exposing sufficient parallelism. For diverges branches, CUDA has compiler optimization features that replaces branch instructions (which cause actual control flow to diverge) with predicated instructions. However, for a long code path, the warp divergence will happen. We should use the branches as less as possible and recall the SIMT execution type used by GPU.

In the next Section, we present some numerical examples to show the high effectiveness and accuracy of the numerical scheme using GPU computing.

5 Numerical results

The first example is a linear BSDE with the driver function f that does not depending on the process z. The second one is a non-linear example. Furthermore, we consider the Black Scholes BSDE as an application of BSDEs in finance. Finally, we test our algorithm with a 2-dimensional example. We implement the parallel algorithm using CUDA C programming. The parallel computing times are compared with the serial ones on a CPU. Furthermore, the speedups are calculated. The CPU is Intel(R) Core(TM) i5-4670 3.40Ghz with 4 cores. The GPU is a NVIDIA GeForce 1070 Ti with a total 8GB GDDR5 memory.

Example 5.1. Consider the linear BSDE[1]

$$\begin{cases} -dy_t = \left(-y_t^3 + \frac{5}{2}y_t^2 - \frac{3}{2}y_t\right) dt - z_t\, dW_t, \\ y_T = \frac{\exp(W_T+T)}{\exp(W_T+T)+1}. \end{cases} \tag{22}$$

[1]Taken from [Zhao et al., 2010].

Preprint – Preprint – Preprint – Preprint – Preprint – Preprint

The analytic solution is

$$\begin{cases} y_t = \frac{\exp(W_t+t)}{\exp(W_t+t)+1}, \\ z_t = \frac{\exp(W_t+t)}{\left(\exp(W_t+t)+1\right)^2}. \end{cases} \tag{23}$$

The exact solution with $T = 1$ is $(y_0, z_0) = \left(\frac{1}{2}, \frac{1}{4}\right)$. In Table 3, we show the importance of working in a uniform domain. Note that the computation time is in seconds.

Table 3: Preliminary results for $N = 256, K_y = K_z = 3$.

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

In Table 4, we present the naive results using 256 threads per block with $K = K_y = K_z$, $t_0 = 0$, $T = 1$, $x \in [-16, 16]$, $L = 32$ and $p^2 = 30$. For an easier understanding, the same results are plotted and presented in Figure 1. It can be easily observed the increase of accuracy when

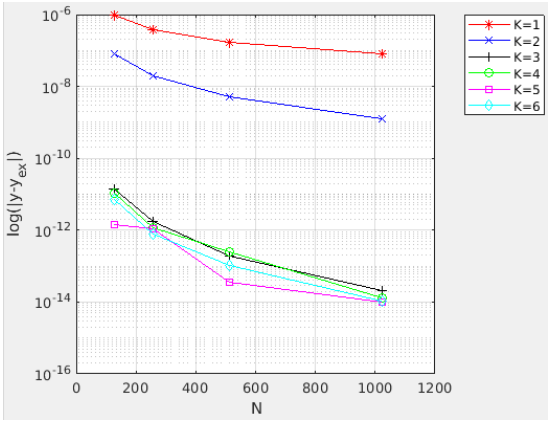
Table 4: Naive results for Example 5.1.

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	16384	1.87E-13	4.41E-10	44.86	3.68	12.19
3	1024	32768	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	16384	2.57E-13	2.68E-10	55.16	3.84	14.35
4	1024	32768	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	16384	3.46E-14	3.02E-10	65.80	3.97	16.57
5	1024	32768	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	16384	1.07E-13	2.92E-10	77.64	4.23	18.35
6	1024	32768	1.03E-14	3.65E-11	311.87	11.97	26.06

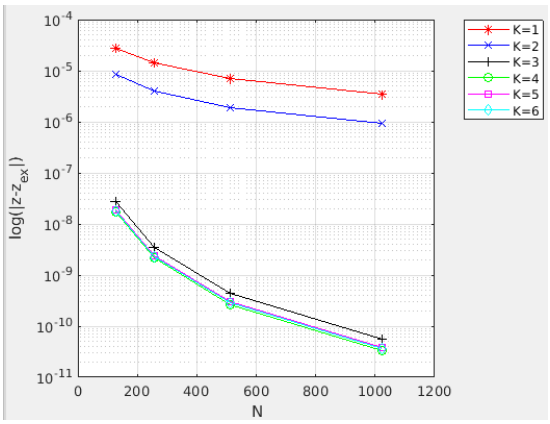
considering a higher-step scheme. Since we have more time layers to consider, more work can be assigned to the GPU and therefore the speedup of the application is increased. That is why the

[2]Number of Picard iterations.

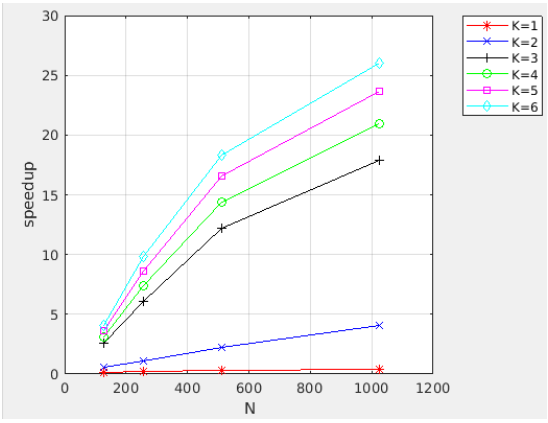
Preprint – Preprint – Preprint – Preprint – Preprint – Preprint



(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 1: Plots of naive results for Example 5.1.

highest speedup (26×) is for a 6-step scheme. Also the highest accuracy is of $\mathcal{O}\left(10^{-15}\right)$ for the y process, since it has 4-th order of convergence.

Example 5.2. Consider the non-linear BSDE[3]

$$\begin{cases} -dy_t = \frac{1}{2}\left(\exp\left(t^2\right) - 4ty_t - 3\exp\left(t^2 - y_t\exp\left(-t^2\right)\right)\right)dt - z_t\,dW_t, \\ y_T = \ln\left(\sin\left(W_T\right) + 3\right)\exp\left(T^2\right). \end{cases} \tag{24}$$

The analytic solution is

$$\begin{cases} y_t = \ln\left(\sin\left(W_t\right) + 3\right)\exp\left(t^2\right), \\ z_t = \exp\left(t^2\right)\frac{\cos(W_t)}{\sin(W_t)+3}. \end{cases} \tag{25}$$

The exact solution with $T = 1$ is $(y_0, z_0) = \left(\ln(3), \frac{1}{3}\right)$. The naive results using 256 threads per block with $K = K_y = K_z$, $t_0 = 0$, $T = 1$, $x \in [-16, 16]$, $L = 32$ and $p = 30$ are presented in Table 5 and plotted in Figure 2. We can observe that the accuracy for this example is smaller than the previous one. This is due to the convergence order of maximum 3, since the driver function depends on the z process. Furthermore, we get higher speedup compared with previous

[3]Taken from [Teng et al., 2018].

Preprint – Preprint – Preprint – Preprint – Preprint – Preprint

Table 5: Naive results for Example 5.2.

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	16384	3.02E-09	1.33E-08	108.92	6.62	16.46
3	1024	32768	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	16384	1.80E-09	7.99E-09	133.26	6.47	20.61
4	1024	32768	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	16384	2.02E-09	8.99E-09	156.63	6.67	23.48
5	1024	32768	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	16384	1.93E-09	8.63E-09	182.74	7.15	25.57
6	1024	32768	2.45E-10	1.09E-09	735.15	20.97	35.05

example due to the more complicated driver function (i.e. more data are accessed, more special functional unit is used etc.). The naive speedup is 35×.

Furthermore, we optimized the kernels created for the this Example. We used the iterative optimization process described in Subsection 4.2 for the case with $N = 1024$ and $K_y = K_y = 3$.

In the first iteration, we gathered the application information using *nvprof*. The results are presented in Table 6a. The application hotspot is *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 is 57×.

In the second iteration, the new 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 6b. We reduced the computation time from 2.48 s to 1.16 s for *sp_inter_non_grid_d_no_for*. 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 point of Subsection 3.3) 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 6c that again the application bottleneck is the same kernel. Therefore, it is not worth

Preprint – Preprint – Preprint – Preprint – Preprint – Preprint

Table 6: Results of iterative optimization process for Example 5.2.

(a) Performance of the main naive kernels.

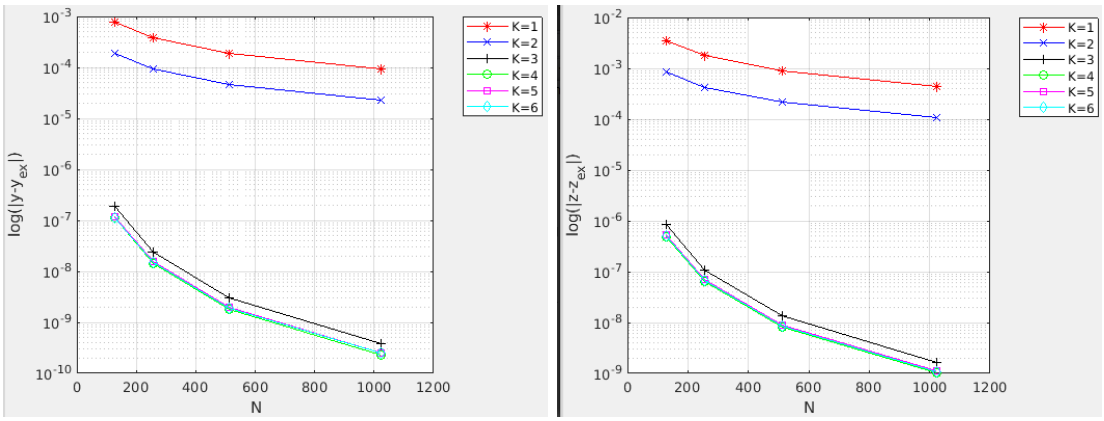
Time(%)	Time(s)	Kernel name
48.35	8.04	nrm2_kernel
14.94	2.48	sp_inter_non_grid_d_no_for
13.70	2.28	calc_f_and_c_exp_d
6.17	1.03	csrMv_kernel
3.60	0.60	calc_y
3.53	0.89	dot_kernel
1.98	0.33	reduce_1Block_kernel
1.56	0.26	axpby_kernel_val
1.34	0.22	calc_c_exp_d

(b) Performance after first iteration of optimization process.

Time(%)	Time(s)	Kernel name
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

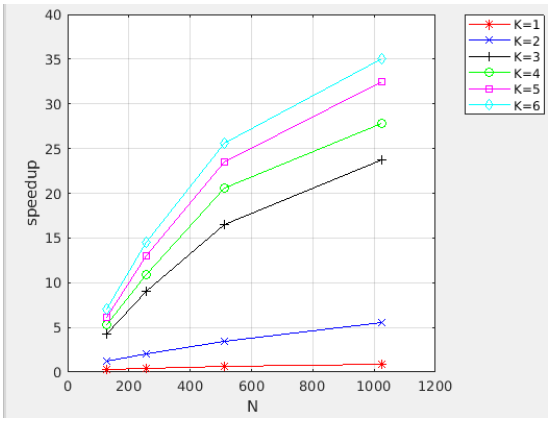
(c) Performance after second iteration of optimization process.

Time(%)	Time(s)	Kernel name
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



(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 naive results for Example 5.2.

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×.

In the following we consider an option pricing example, the Black-Scholes model. Consider a security market that contains one bond with price p_t and one stock with price S_t. Therefore, their dynamics are described by:

$$\begin{cases} dp_t = r_t p_t \, dt, \quad t \geq 0, \\ p_0 = p, \end{cases} \tag{26}$$

$$\begin{cases} dS_t = \mu_t S_t dt + \sigma_t S_t \, dW_t, \quad t \geq 0, \\ S_0 = x, \end{cases} \tag{27}$$

where r_t denotes the interest rate of the bond, p is its current value, μ_t is the expected return on the stock S_t, σ_t is the volatility of the stock, x is its current value and W_t denotes the Brownian motion.

Suppose that an agent sells the option at price y_t and then invests it in the market. Denote his wealth on each time by y_t. Assume that at each time the agent invests a portion of his wealth in an amount given by π_t into the stock, and the rest $(y_t - \pi_t)$ into the bond. Now the agent has

Preprint – Preprint – Preprint – Preprint – Preprint – Preprint

Preprint – Preprint – Preprint – Preprint – Preprint – Preprint – Preprint

a portfolio based on the stock and the bond. Considering a stock that pays a dividend $\delta(t, S_t)$, the dynamics of the wealth process y_t are described by

$$\begin{aligned} dy_t &= \frac{\pi_t}{S_t}\, dS_t + \frac{y_t - \pi_t}{p_t}\, dp_t + \pi_t \delta(t, S_t)\ dt \\ &= \frac{\pi_t}{S_t}\left(\mu_t S_t\, dt + \sigma_t S_t\, dW_t\right) + \frac{y_t - \pi_t}{p_t}\left(r_t p_t\, dt\right) + \pi_t \delta(t, S_t)\ dt \\ &= \left(r_t y_t + \pi_t\left(\mu_t - r_t + \delta(t, S_t)\right)\right) dt + \pi_t \sigma_t\, dW_t. \end{aligned} \tag{28}$$

Let $z_t = \pi_t \sigma_t$, then

$$-\, dy_t = -\left(r_t y_t + \left(\mu_t - r_t + \delta(t, S_t)\right)\frac{z_t}{\sigma_t}\right) dt + z_t\, dW_t. \tag{29}$$

For a call option, one needs to solve a FBSDE, where the forward part is given from the SDE modelling of the stock price dynamics.

Example 5.3. Consider the Black-Scholes FBSDE[4]

$$\begin{cases} dS_t = \mu_t S_t dt + \sigma_t S_t\, dW_t, \quad S_0 = x, \quad t \in [0, T] \\ -dy_t = -\left(r_t y_t + \left(\mu_t - r_t + \delta(t, S_t)\right)\frac{z_t}{\sigma_t}\right) dt + z_t\, dW_t, \quad t \in [0, T) \\ y_T = (S_T - K)^+. \end{cases} \tag{30}$$

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

$$\begin{cases} y_t = V(t, S_t) = S_t \exp\left(-\delta(T - t)\right) N(d_1) - K \exp\left(-r(T - t)\right) N(d_2), \\ z_t = S_t \frac{\partial V}{\partial S}\sigma = S_t \exp\left(-\delta(T - t)\right) N(d_1)\, \sigma, \\ d_{1/2} = \frac{\ln\left(\frac{S_t}{K}\right) + \left(r \pm \frac{\sigma^2}{2}\right)(T - t)}{\sigma\sqrt{T - t}}, \end{cases} \tag{31}$$

where $N(\cdot)$ is the cumulative standard normal distribution function. In this example, we consider $T = 0.33$, $K = S_0 = 100$, $r = 0.03$, $\mu = 0.05$, $\delta = 0.04$, $\sigma = 0.2$, 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 (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 [Kreiss et al., 1970]. For the forward part of (31), we have the analytic solution

$$S_t = S_0 \exp\Big(\big(\mu - \frac{\sigma^2}{2}\big)t + \sigma\, W_t\Big). \tag{32}$$

Discretizing (32), the exponential term will lead to a non-uniform grid. Therefore, instead of working in the stock price domain, we work in the log stock price domain. If we denote $X_t = \ln S_t$, then the analytic solution of X_t reads

$$X_t = X_0 + \big(\mu - \frac{\sigma^2}{2}\big)t + \sigma\, W_t. \tag{33}$$

The backward part is the same as in (30). In Table 7 we show the importance of using the log stock price.

Preprint – Preprint – Preprint – Preprint – Preprint – Preprint

Table 7: Preliminary results for $N = 256, K_y = K_z = 3$ for Black-Scholes Example.

M	$t_{CPU}^{non-optimal}$	$t_{CPU}^{optimal}$	$speedup$
24826	18531.23	47.54	389.80

Table 8: Naive results for Black-Scholes Example.

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	12414	9.72E-11	6.85E-10	11.81	2.39	4.93
3	256	24826	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	12414	7.55E-11	4.71E-10	14.26	2.45	5.82
4	256	24826	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	12414	4.40E-11	5.04E-10	16.30	2.39	6.83
5	256	24826	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	12414	3.82E-11	4.89E-10	18.69	2.54	7.35
6	256	24826	6.16E-12	6.24E-11	77.53	6.47	11.99

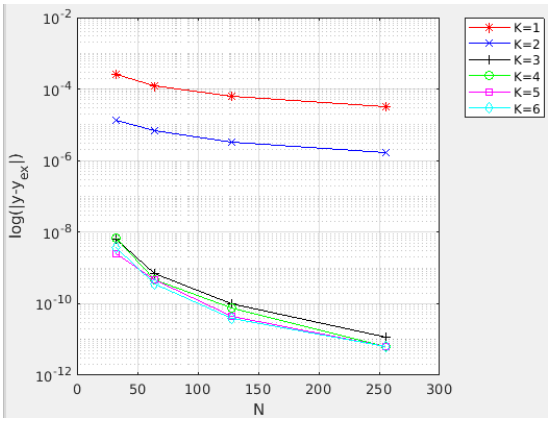
The naive results using 256 threads per block with $K = K_y = K_z$, $t_0 = 0, T = 0.33$, $x \in [-16, 16]$, $L = 32$ and $p = 30$ are presented in Table 8 and plotted in Figure 3. The highest accuracy is achieved when considering a 6-step scheme, and having also the highest speedup of $12\times$.

We optimized the kernels created for the Black-Scholes BSDE for $N = 256$ and $K_y = K_z = 6$. The optimization iteration process is the same as in Example 5.2. The final speedup is $31\times$. Note that this speedup is for 256 time layers. In Example 5.2, we optimized for 1024 time layers. If we consider the same time layers for Black-Scholes Example, we get strange results (errors start to reduce tremendously), due to the non-smooth problem of z process.

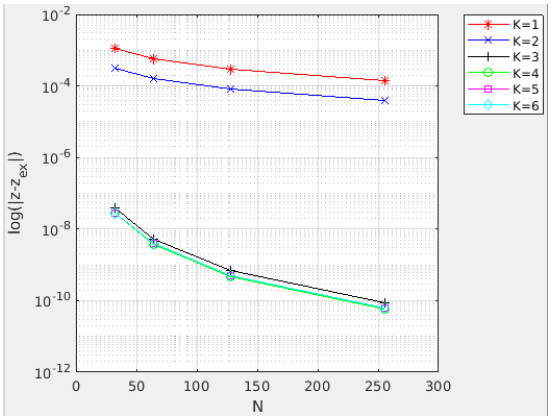
Example 5.4. Consider the 2-dimensional BSDE[5]

$$\begin{cases} -dy_t = (y_t - z_t A)\, dt - z_t\, dW_t, \\ y_T = \sin\left(MW_T + T\right), \end{cases} \tag{34}$$

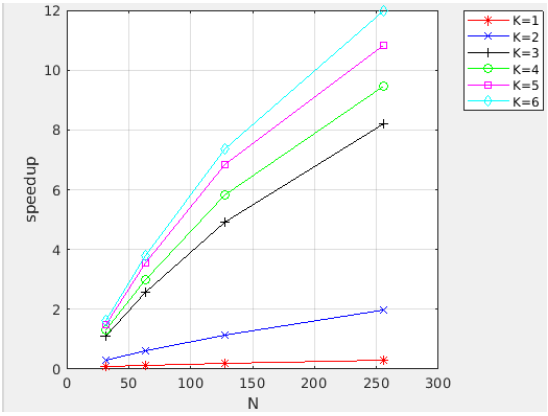
[4]Taken from [Zhao et al., 2010].
[5]Taken from [Teng et al., 2018].



(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 naive results for Example 5.3.

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

The analytic solution is

$$\begin{cases} y_t = \sin\left(MW_t + t\right), \\ y_T = \left(\cos\left(MW_t + t\right), \cos\left(MW_t + t\right)\right. \end{cases} \tag{35}$$

The exact solution with $T = 1$ is $\left(y_0, \left(z_0^1, z_0^2\right)\right) = (0, (1, 1))$. The naive results using 256 threads per block with $K = K_y = K_z$, $t_0 = 0$, $T = 1$, $x \in [-8, 8]$, $L = 8$ and $p = 30$ are presented in Table 9 and plotted in Figure 4. The highest speedup is $21\times$, which requires a 5 GB of memory. Therefore, we did not consider more time layers as the maximum amount of memory for the GPU is 8 GB and with $N = 64$, we could not get the results for $K \geq 3$. We optimized the case where $N = 32$ and $K_y = K_z = 6$.

In the first iteration, we gathered the application information using *nvprof*. The results are presented in Table 10a. The application hotspot is *sp_inter_non_grid* kernel, which calculates the non-grid values for process y and z. The performance of the kernel is limited by the memory operations. Accessing and storing of data is not optimal. Therefore, we used loop interchanging to overcome this problem. We reduced the computation time from 15.28 s to 10.05 s for *sp_inter_non_grid* kernel. By default, we reduced the computation time for the kernels *calc_f_and_c_exp* and *calc_c_exp* from 4.05 s to 0.46 s and 0.16 s to 0.02 s respectively. The new

Preprint – Preprint – Preprint – Preprint – Preprint – Preprint

Preprint – Preprint – Preprint – Preprint – Preprint – Preprint

Table 9: Naive results for Example 5.4.

K	N	M	$\lvert y_{0,0} - y_0^0 \rvert$	$\lvert z_{0,0} - z_0^0 \rvert$	t_{CPU}	t_{GPU}	speedup	Used GPU Memory (GB)
1	8	46	1.32E-02	1.95E-03	0.18	0.01	21.95	0.20
1	16	64	6.46E-03	4.64E-03	0.68	0.02	32.21	0.21
1	32	92	3.18E-03	3.24E-03	2.82	0.11	25.20	0.22
2	8	78	5.90E-04	8.49E-03	0.82	0.03	23.66	0.23
2	16	128	8.31E-04	3.46E-03	4.73	0.22	21.16	0.30
2	32	216	5.84E-04	1.44E-03	27.85	1.42	19.58	0.48
3	8	3128	3.94E-04	1.43E-03	2.89	0.13	21.70	0.35
3	16	3256	6.75E-05	2.08E-04	27.78	1.33	20.92	0.80
3	32	512	9.72E-06	2.79E-05	252.17	11.52	21.88	2.61
4	8	128	1.90E-04	8.09E-04	3.17	0.15	21.53	0.40
4	16	256	3.78E-05	1.24E-04	33.79	1.64	20.63	1.01
4	32	512	5.69E-06	1.67E-05	321.05	14.87	21.60	3.41
5	8	128	1.39E-04	7.49E-04	3.21	0.15	21.71	0.45
5	16	256	3.65E-05	1.30E-04	39.07	1.90	20.56	1.20
5	32	512	6.00E-06	1.83E-05	381.17	17.93	21.26	4.20
6	8	128	8.13E-05	5.97E-04	2.91	0.13	21.77	0.50
6	16	256	3.07E-05	1.18E-04	43.04	2.08	20.68	1.39
6	32	512	5.49E-06	1.73E-05	441.70	20.67	21.37	4.99

Table 10: Results of iterative optimization process for Example 5.4.

(a) Performance of the main naive kernels.

Time(%)	Time(s)	Kernel name
73.95	15.28	sp_inter_non_grid
19.61	4.05	calc_f_and_c_exp
3.65	0.75	swap_coeff
1.09	0.22	find_coeff
0.79	0.16	calc_c_exp

(b) Performance after first iteration of optimization process.

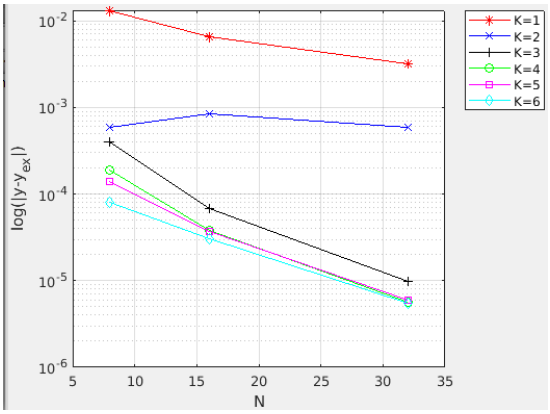
Time(%)	Time(s)	Kernel name
86.68	10.05	sp_inter_non_grid
6.50	0.75	swap_coeff
3.95	0.46	calc_f_and_c_exp
1.93	0.22	find_coeff
0.40	0.05	calc_y

(c) Performance after second iteration of optimization process.

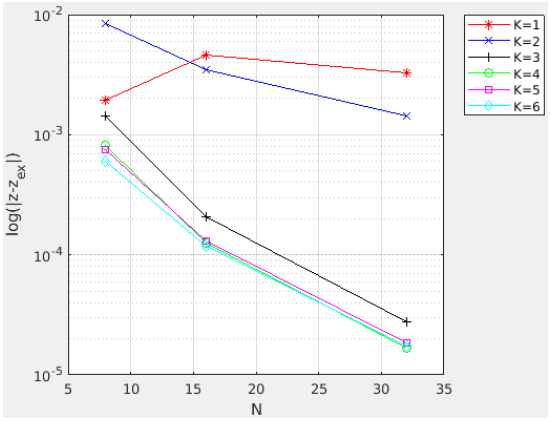
Time(%)	Time(s)	Kernel name
87.62	5.56	sp_inter_non_grid
6.96	0.44	calc_f_and_c_exp
2.04	0.13	find_coeff
2.02	0.13	swap_coeff
0.34	0.02	calc_y

speedup after first iteration is 38×.

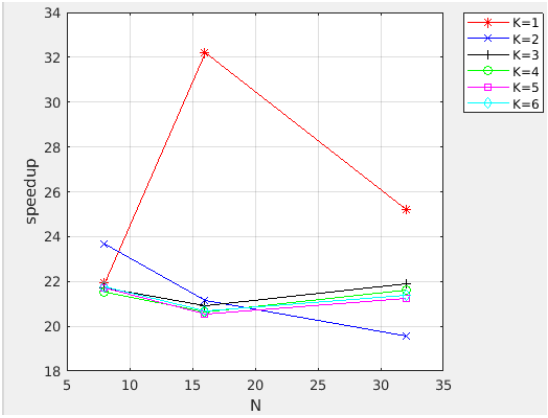
In the second iteration, we checked the new bootleneck for the application and resulted again the same kernel. However, we can't optimize this kernel furthermore, so we considered the next kernel, *swap_coeff*. This kernel swaps the coefficients in order to reduce the unnecessary interpolations (recall second point in Subsection 3.2). The performance of the kernel is limited by the latency of arithmetic and memory operations. Therefore, we considered again loop interchanging. This reduced the computation time of the corresponding kernel and other kernels



(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 naive results for Example 5.3.

related with it, as shown in Table 10b. We reduced the computation time from 0.75 s to 0.13 s for $swap\_coeff$ kernel. By default, we reduced the computation time from 10.05 s to 5.56 s for $sp\_inter\_non\_grid$ because even here the loading of bicubic interpolation coefficients is changed due to loop interchanging and therefore the memory access is optimized and also for $find\_coeff$ (a kernel that find the bicubic coefficients by applying a matrix vector multiplication) from 0.22 s to 0.13 s. The new speedup is 63×. It can be observed from Table 10c that again the application bottleneck is the same kernel. Therefore, it is not worth optimizing the application furthermore. Finally, we increased the block dimension from 256 threads to 1024 in order to increase parallelism. The final speedup is 70×.

6 Conclusions and outlook

In this work we parallelized the multistep method developed in [Teng et al., 2018] for the numerical approximation of BSDEs on GPU. Firstly, we presented an optimal operation to find the location of the interpolated values. This was essential for the reduction of the computational time. Our numerical results have shown that a high accuracy can be achieved with less computation times. For a further acceleration, we have investigated how to optimize the application after finding the performance bottlenecks and applying optimization techniques. The proposed GPU

Preprint – Preprint – Preprint – Preprint – Preprint – Preprint

Preprint – Preprint – Preprint – Preprint – Preprint – Preprint

parallel algorithm make the multistep schemes in [Teng et al., 2018] and [Zhao et al., 2010] be really more useful in practice.

References

[Abramowitz and Stegun, 1972] Abramowitz, M. and Stegun, I. A. (1972). Handbook of mathematical functions dover publications inc. *New York*.

[Ankirchner et al., 2010] Ankirchner, S., Blanchet-Scalliet, C., and Eyraud-Loisel, A. (2010). Credit risk premia and quadratic bsdes with a single jump. *International Journal of Theoretical and Applied Finance*, 13(07):1103–1129.

[Bender and Steiner, 2012] Bender, C. and Steiner, J. (2012). Least-squares monte carlo for backward sdes. In *Numerical methods in finance*, pages 257–289. Springer.

[Bouchard and Touzi, 2004] Bouchard, B. and Touzi, N. (2004). Discrete-time approximation and monte-carlo simulation of backward stochastic differential equations. *Stochastic Processes and their applications*, 111(2):175–206.

[Crisan and Manolarakis, 2012] Crisan, D. and Manolarakis, K. (2012). Solving backward stochastic differential equations using the cubature method: application to nonlinear pricing. *SIAM Journal on Financial Mathematics*, 3(1):534–571.

[Dai et al., 2010] Dai, B., Peng, Y., and Gong, B. (2010). Parallel option pricing with bsde method on gpu. In *2010 Ninth International Conference on Grid and Cloud Computing*, pages 191–195. IEEE.

[El Karoui et al., 1997] El Karoui, N., Peng, S., and Quenez, M. C. (1997). Backward stochastic differential equations in finance. *Mathematical finance*, 7(1):1–71.

[Eyraud-Loisel, 2005] Eyraud-Loisel, A. (2005). Backward stochastic differential equations with enlarged filtration: Option hedging of an insider trader in a financial market with jumps. *Stochastic processes and their Applications*, 115(11):1745–1763.

[Fahim et al., 2011] Fahim, A., Touzi, N., Warin, X., et al. (2011). A probabilistic numerical method for fully nonlinear parabolic pdes. *The Annals of Applied Probability*, 21(4):1322–1364.

[Gobet et al., 2005] Gobet, E., Lemor, J.-P., Warin, X., et al. (2005). A regression-based monte carlo method to solve backward stochastic differential equations. *The Annals of Applied Probability*, 15(3):2172–2202.

[Gobet et al., 2016] Gobet, E., López-Salas, J. G., Turkedjiev, P., and Vazquez, C. (2016). Stratified regression monte-carlo scheme for semilinear pdes and bsdes with large scale parallelization on gpus. *SIAM Journal on Scientific Computing*, 38(6):C652–C677.

[Kreiss et al., 1970] Kreiss, H.-O., Thomée, V., and Widlund, O. (1970). Smoothing of initial data and rates of convergence for parabolic difference equations. *Communications on Pure and Applied Mathematics*, 23(2):241–259.

[Labart and Lelong, 2011] Labart, C. and Lelong, J. (2011). A parallel algorithm for solving bsdes-application to the pricing and hedging of american options. *arXiv preprint arXiv:1102.4666*.

Preprint – Preprint – Preprint – Preprint – Preprint – Preprint

[Lemor et al., 2006] Lemor, J.-P., Gobet, E., Warin, X., et al. (2006). Rate of convergence of an empirical regression method for solving generalized backward stochastic differential equations. *Bernoulli*, 12(5):889–916.

[Pardoux and Peng, 1990] Pardoux, E. and Peng, S. (1990). Adapted solution of a backward stochastic differential equation. *Systems & Control Letters*, 14(1):55–61.

[Peng, 1991] Peng, S. (1991). Probabilistic interpretation for systems of quasilinear parabolic partial differential equations. *Stochastics and stochastics reports*, 37(1-2):61–74.

[Peng et al., 2011] Peng, Y., Gong, B., Liu, H., and Dai, B. (2011). Option pricing on the gpu with backward stochastic differential equation. In *2011 Fourth International Symposium on Parallel Architectures, Algorithms and Programming*, pages 19–23. IEEE.

[Peng et al., 2014] Peng, Y., Liu, H., Yang, S., and Gong, B. (2014). Parallel algorithm for bsdes based high dimensional american option pricing on the gpu. *Journal of Computational Information Systems*, 10(2):763–771.

[Teng, 2018] Teng, L. (2018). A review of tree-based approaches to solve forward-backward stochastic differential equations.

[Teng et al., 2018] Teng, L., Lapitckii, A., and Günther, M. (2018). A multi-step scheme based on cubic spline for solving backward stochastic differential equations. *arXiv preprint arXiv:1809.00324*.

[Zhao et al., 2006] Zhao, W., Chen, L., and Peng, S. (2006). A new kind of accurate numerical method for backward stochastic differential equations. *SIAM Journal on Scientific Computing*, 28(4):1563–1581.

[Zhao et al., 2010] Zhao, W., Zhang, G., and Ju, L. (2010). A stable multistep scheme for solving backward stochastic differential equations. *SIAM J. Numer. Anal.*, 48:1369–1394.