GPU pricing of exotic cross-currency interest rate derivatives with a foreign exchange volatility skew model

Duy Minh Dang[a,*], Christina C. Christara[a], Kenneth R. Jackson[a]

[a]*Department of Computer Science, University of Toronto, Toronto, ON, M5S 3G4, Canada*

Abstract

We present a Graphics Processing Unit (GPU) parallelization of the computation of the price of exotic cross-currency interest rate derivatives via a Partial Differential Equation (PDE) approach. In particular, we focus on the GPU-based parallel pricing of long-dated foreign exchange (FX) interest rate hybrids, namely Power Reverse Dual Currency (PRDC) swaps with Bermudan cancelable features. We consider a three-factor pricing model with FX volatility skew which results in a time-dependent parabolic PDE in three spatial dimensions. Finite difference methods on uniform grids are used for the spatial discretization of the PDE, and the Alternating Direction Implicit (ADI) technique is employed for the time discretization. We then exploit the parallel architectural features of GPUs together with the Compute Unified Device Architecture (CUDA) framework to design and implement an efficient parallel algorithm for pricing PRDC swaps. Over each period of the tenor structure, we divide the pricing of a Bermudan cancelable PRDC swap into two independent pricing subproblems, each of which can efficiently be solved on a GPU via a parallelization of the ADI timestepping technique. Numerical results indicate that GPUs can provide significant increase in performance over CPUs when pricing PRDC swaps. An analysis of the impact of the FX skew on such derivatives is provided.

Keywords: Power Reverse Dual Currency (PRDC) swaps, Bermudan cancelable, Partial Differential Equation (PDE), Alternating Direction Implicit (ADI), finite differences, Graphics Processing Units (GPUs)

1. Introduction

Cross-currency interest rate derivatives in general, and foreign exchange (FX) interest rate hybrids in particular, are of enormous current practical importance. In particular, Power Reverse Dual Currency (PRDC) swaps are one of the most widely traded and liquid cross-currency exotics [14]. The pricing of PRDC swaps, especially those with Bermudan cancelable features, is the subject of great interest in practice, especially among financial institutions. As these products are exposed to moves in both the FX rate and the interest rates in both currencies, multi-factor models must be used. The current standard practice is to use two one-factor Gaussian models for the term structures and either a one-factor log-normal model (e.g., [10, 14]) or a one-factor skew model via a local volatility [13] for the FX rate. It is important to emphasize that long-dated cross-currency interest rate derivatives with exotic features, such as Bermudan cancelable PRDC swaps, are particularly sensitive to the FX volatility skew [13]. Hence, the log-normality assumption of the FX rate is questionable for the pricing of PRDC swaps. Using a local volatility model provides better modeling for the skewness of the FX rate, and at the same time avoids introducing more stochastic factors into the model.

The ever growing interest in cross-currency interest rate derivatives in general, and PRDC swaps in particular, has created a need for efficient pricing and hedging strategies for them. The popular choice for pricing PRDC swaps is Monte-Carlo (MC) simulation, but this approach has several major disadvantages, including slow convergence, and the limitation that the price is obtained at a single point only in the domain, as opposed to the global character of the Partial Differential Equation (PDE) approach. In addition, accurate hedging parameters, such as delta and

*Corresponding author

Email addresses: dmdang@cs.toronto.edu (Duy Minh Dang), ccc@cs.toronto.edu (Christina C. Christara), krj@cs.toronto.edu (Kenneth R. Jackson)

gamma, are generally harder to compute via a MC approach than via a PDE approach. On the other hand, when pricing PRDC swaps by the PDE approach, each stochastic factor in the pricing model gives rise to a spatial variable in the PDE. Due to the "curse of dimensionality" associated with high-dimensional PDEs - not to mention additional complexity due to multiple cash flows and exotic features, such as Bermudan cancelability - the pricing of such derivatives via the PDE approach is not only mathematically challenging but also very computationally intensive.

Over the last few years, the rapid evolution of Graphics Processing Units (GPUs) into powerful, cost efficient, programmable computing architectures for general purpose computations has provided application potential beyond the primary purpose of graphics processing. In the area of computational finance, although there has been great interest in utilizing GPUs in developing efficient pricing architectures for computationally intensive problems, the applications mostly focus on option pricing and MC simulations (e.g., [1, 11, 15]). The literature on utilizing GPUs in pricing financial derivatives via a PDE approach is rather sparse, with scattered work presented at conferences or workshops, such as [3, 4]. The literature on GPU-based PDE methods for pricing cross-currency interest rate derivatives is even less developed. In [6], PDE methods for pricing Bermudan cancelable PRDC swaps are introduced in the public domain. However, to the best of our knowledge, an efficient GPU-based parallel algorithm based on the PDE approach to pricing such derivatives has not been discussed in the literature. These shortcomings motivated our work.

This paper discusses the application of GPUs to solve the PDEs for pricing exotic cross-currency interest rate derivatives under a FX volatility skew model, with strong emphasis on Bermudan cancelable PRDC swaps. We use the parallel architectural features of GPUs together with the Compute Unified Device Architecture (CUDA) framework to design and implement an efficient GPU-based parallel algorithm for pricing such derivatives. Our algorithm is based on the observation that, over each period of the swap's tenor structure, the pricing of a Bermudan cancelable PRDC swap can be divided into two independent pricing subproblems: (i) the pricing of the underlying PRDC swap and (ii) the pricing of an associated Bermudan swaption. Each of these subproblems can be solved efficiently on a GPU via an efficient parallelization of the Alternating Direction Implicit (ADI) timestepping technique that is employed for the time discretization. The results of this paper demonstrate the efficiency of the parallel numerical methods and show that GPUs can provide a significant increase in performance over CPUs when pricing exotic cross-currency interest rate derivatives. Although we primarily focus on a three-factor model, many of the ideas and results in this paper can be naturally extended to higher-dimensional applications with constraints.

The remainder of this paper is organized as follows. Section 2 describes the pricing PDE for cross-currency interest rate derivatives. A discretization scheme for the pricing PDE is presented in Section 3. A Finite Difference (FD) method on a uniform grid is employed for the spatial discretization of the PDE and the ADI method is used for its time discretization. The pricing of Bermudan cancelable PRDC swaps is discussed in Section 4. A GPU-based parallel pricing algorithm for Bermudan cancelable PRDC swaps is discussed in Section 5. Numerical results and related discussions are presented in Section 6. Section 7 concludes the paper and outlines possible future work.

2. The pricing model and the associated PDE

We consider an economy with two currencies, "domestic" (d) and "foreign" (f). We denote by $s(t)$ the spot FX rate, the number of units of domestic currency per one unit of foreign currency. Let $r_i(t), i = d, f$, denote the domestic and foreign short rates, respectively. Under the domestic risk-neutral measure, the dynamics of $s(t), r_d(t)$ and $r_f(t)$ are described by [13]

$$\begin{aligned} &\frac{ds(t)}{s(t)} = (r_d(t) - r_f(t))dt + \gamma(t, s(t))dW_s(t), \qquad dr_d(t) = (\theta_d(t) - \kappa_d(t)r_d(t))dt + \sigma_d(t)dW_d(t), \\ &dr_f(t) = (\theta_f(t) - \kappa_f(t)r_f(t) - \rho_{fs}(t)\sigma_f(t)\gamma(t, s(t)))dt + \sigma_f(t)dW_f(t), \end{aligned} \tag{1}$$

where $W_d(t), W_f(t)$, and $W_s(t)$ are correlated Brownian motions with $dW_d(t)dW_s(t) = \rho_{ds}dt$, $dW_f(t)dW_s(t) = \rho_{fs}dt$, and $dW_d(t)dW_f(t) = \rho_{df}dt$. The short rates follow the mean-reverting Hull-White model with deterministic mean reversion rate and the volatility functions respectively denoted by $\kappa_i(t)$ and $\sigma_i(t)$, for $i = d, f$, while $\theta_i(t), i =$

d, f, also deterministic, capture the current term structures. The "quanto" drift adjustment, $-\rho_{fs}(t)\sigma_f(t)\gamma(t, s(t))$, for $dr_f(t)$ comes from changing the measure from the foreign risk-neutral measure to the domestic risk neutral one. The functions $\kappa_i(t), \sigma_i(t), \theta_i(t), i = d, f$, are all deterministic. The local volatility function $\gamma(t, s(t))$ for the spot FX rate has the functional form [13]

$$\gamma(t, s(t)) = \xi(t)\Big(s(t)/L(t)\Big)^{\varsigma(t)-1}, \tag{2}$$

where $\xi(t)$ is the relative volatility function, $\varsigma(t)$ is the time-dependent constant elasticity of variance (CEV) parameter and $L(t)$ is a time-dependent scaling constant which is usually set to the forward FX rate with expiry t, denoted by $F(0, t)$, for convenience in calibration. A discussion of the calibration of the aforementioned pricing model can be found in the literature e.g., [6, 13]. It is important to point out that incorporating a proper FX volatility skew into the model is essential for pricing FX-linked hybrids, such as PRDC swaps (see, for example, [13]), due to the specific choice of strikes of the coupons, as well as the cancelable features of the swaps. We return to this point in Subsection 6.2 on the analysis of the FX skew on the prices of cancelable PRDC swaps.

We now give a PDE that the price of any security whose payoff is a function of $s(t), r_d(t)$ and $r_f(t)$ must satisfy. A derivation of the PDE is presented in [6]. Let $u \equiv u(s, r_d, r_f, t)$ denote the domestic value function of a security, such as a PRDC swap, with a terminal payoff measurable with respect to the σ-algebra at maturity time T_{end} and without intermediate payments. Furthermore, assume that u is sufficiently smooth on $\mathbb{R}^3_+ \times [T_{start}, T_{end})$. Then on $\mathbb{R}^3_+ \times [T_{start}, T_{end})$, u satisfies the PDE

$$\begin{aligned}\frac{\partial u}{\partial t}+\mathcal{L}u \equiv &\frac{\partial u}{\partial t}+\frac{1}{2}\gamma^2(t, s(t))s^2\frac{\partial^2 u}{\partial s^2}+\frac{1}{2}\sigma_d^2(t)\frac{\partial^2 u}{\partial r_d^2}+\frac{1}{2}\sigma_f^2(t)\frac{\partial^2 u}{\partial r_f^2}+\rho_{ds}\sigma_d(t)\gamma(t, s(t))s\frac{\partial^2 u}{\partial s\partial r_d}+\rho_{fs}\sigma_f(t)\gamma(t, s(t))s\frac{\partial^2 u}{\partial s\partial r_f}\\ &+\rho_{df}\sigma_d(t)\sigma_f(t)\frac{\partial^2 u}{\partial r_d\partial r_f}+(r_d-r_f)s\frac{\partial u}{\partial s}+\Big(\theta_d(t)-\kappa_d(t)r_d\Big)\frac{\partial u}{\partial r_d}+\Big(\theta_f(t)-\kappa_f(t)r_f-\rho_{fs}\sigma_f(t)\gamma(t, s(t))\Big)\frac{\partial u}{\partial r_f}-r_d u=0.\end{aligned} \tag{3}$$

Since payoffs and fund flows are deal-specific, we defer specifying the terminal conditions until Section 4. The difficulty with choosing boundary conditions is that, for an arbitrary payoff, they are not known. A detailed analysis on the boundary conditions is certainly beyond the scope of this short paper, and is a topic of future research. For this project, we impose Dirichlet-type "stopped process" boundary conditions where we stop the processes $s(t), r_f(t), r_d(t)$ when any of the three hits the boundary. Thus, the value on the boundary is simply the discounted payoff for the current values of the state variables [7]. Note that, since we solve the PDE backward in time, the change of variable $\tau = T_{end} - t$ is used. Under this change of variable, the PDE (3) becomes $\frac{\partial u}{\partial \tau} = \mathcal{L}u$ and is solved forward in τ. The pricing of cross-currency interest rate derivatives is defined in an unbounded domain $\{(s, r_d, r_f, \tau)|s \geq 0, r_d \geq 0, r_f \geq 0, \tau \in [0, T]\}$, where $T = T_{end} - T_{start}$. To solve the PDE (3) numerically by finite difference methods, we must truncate the unbounded domain into a finite-sized computational one $\{(s, r_d, r_f, \tau) \in [0, S] \times [0, R_d] \times [0, R_f] \times [0, T]\} \equiv \Omega \times [0, T]$, where S, R_f, R_d are sufficiently large [16].

3. Discretization

Let the number of subintervals be $n+1$, $p+1$, $q+1$ and $l+1$ in the s-, r_d-, r_f-, and τ-directions, respectively. The uniform grid mesh widths in the respective directions are denoted by $\Delta s = \frac{S}{n+1}$, $\Delta r_d = \frac{R_d}{p+1}$, $\Delta r_f = \frac{R_f}{q+1}$, and $\Delta\tau = \frac{T}{l+1}$. The grid point values of a FD approximation are denoted by

$$u^m_{i,j,k} \approx u(s_i, r_{dj}, r_{fk}, \tau_m) = u(i\Delta s, j\Delta r_d, k\Delta r_f, m\Delta\tau),$$

where $i = 1, \ldots, n$, $j = 1, \ldots, p$, $k = 1, \ldots, q$, $m = 1, \ldots, l+1$. Second-order FD approximations to the first and second partial derivatives of the space variables in (3) are obtained by two- and three-point standard *central* schemes, respectively, while the cross derivatives are approximated by a second-order four-point FD stencil.

Regarding the time discretization, we employ a splitting technique of ADI type. Let $\mathbf{u}^m$ denote the vector of values of the unknown price at time $\tau_m = m\Delta\tau$ on the mesh Ω that approximates the exact solution $u^m =$

$u(s, r_d, r_f, \tau_m)$. We denote by $\mathbf{A}^m$ the matrix arising from the FD discretization of the differential operator $\mathcal{L}$ at τ_m. For brevity, we omit the standard explicit formula for $\mathbf{A}^m$. We decompose the matrix $\mathbf{A}^m$ into four submatrices: $\mathbf{A}^m = \sum_{i=0}^{3} \mathbf{A}_i^m$. The matrix $\mathbf{A}_0^m$ is the part of $\mathbf{A}$ that comes from the FD discretization of the mixed derivative terms in (3), while the matrices $\mathbf{A}_1^m$, $\mathbf{A}_2^m$ and $\mathbf{A}_3^m$ are the three parts of $\mathbf{A}^m$ that correspond to the spatial derivatives in the s-, r_d-, and r_f-directions, respectively. The term $r_d u$ in (3) is distributed evenly over $\mathbf{A}_1^m$, $\mathbf{A}_2^m$ and $\mathbf{A}_3^m$.

We consider the splitting scheme based on the Hundsdorfer and Verwer (HV) approach (scheme (1.4) in [9] with $\theta = \mu = \frac{1}{2}$). Starting from $\mathbf{u}^{m-1}$, the HV scheme generates an approximation $\mathbf{u}^m$ to the exact solution u^m, $m = 1, \ldots, l+1$, by

$$\begin{cases} \mathbf{v}_0 = \mathbf{u}^{m-1} + \Delta\tau(\mathbf{A}^{m-1}\mathbf{u}^{m-1} + \mathbf{g}^{m-1}), & (4.1) \\ (\mathbf{I} - \frac{1}{2}\Delta\tau\mathbf{A}_i^m)\mathbf{v}_i = \mathbf{v}_{i-1} - \frac{1}{2}\Delta\tau\mathbf{A}_i^{m-1}\mathbf{u}^{m-1} + \frac{1}{2}\Delta\tau(\mathbf{g}_i^m - \mathbf{g}_i^{m-1}), \quad i = 1,2,3, & (4.2) \\ \widetilde{\mathbf{v}}_0 = \mathbf{v}_0 + \frac{1}{2}\Delta\tau(\mathbf{A}^m\mathbf{v}_3 - \mathbf{A}^{m-1}\mathbf{u}^{m-1}) + \frac{1}{2}\Delta\tau(\mathbf{g}^m - \mathbf{g}^{m-1}), & (4.3) \\ (\mathbf{I} - \frac{1}{2}\Delta\tau\mathbf{A}_i^m)\widetilde{\mathbf{v}}_i = \widetilde{\mathbf{v}}_{i-1} - \frac{1}{2}\Delta\tau\mathbf{A}_i^m\mathbf{v}_3, \quad i = 1,2,3, & (4.4) \\ \mathbf{u}^m = \widetilde{\mathbf{v}}_3. & (4.5) \end{cases} \tag{4}$$

In (4), the vector $\mathbf{g}^m$ is given by $\mathbf{g}^m = \sum_{i=0}^{3} \mathbf{g}_i^m$, where $\mathbf{g}_i^m$ are obtained from the boundary conditions corresponding to the respective spatial derivative terms. Note that the HV scheme has been proved to be unconditionally stable for arbitrary spatial dimension [9].

REMARK 1. The HV splitting scheme treats the mixed derivative part ($\mathbf{A}_0^m$) in a fully-explicit way, while the $\mathbf{A}_i^m$ parts, $i = 1, 2, 3$, are treated implicitly. The FD discretization for the spatial variables by two- and three-point central schemes implies that, if the grid points are ordered appropriately, the matrices $\mathbf{A}_1^m$, $\mathbf{A}_2^m$ and $\mathbf{A}_3^m$ are block-diagonal with tridiagonal blocks. (There is a different ordering for each of $\mathbf{A}_1^m$, $\mathbf{A}_2^m$ and $\mathbf{A}_3^m$.) As a result, the number of floating-point operations per time step is directly proportional to npq, which yields a big reduction in computational cost compared to the application of a direct method, such as the LU factorization, to solve the problem arising from a FD time discretization, such as Crank-Nicolson. For more details about our discretization scheme, see [6].

4. Pricing Bermudan cancelable PRDC swaps

Essentially, PRDC swaps are long-dated swaps (maturities of 30 years or more) which pay FX-linked coupons, i.e. PRDC coupons, in exchange for LIBOR floating-rate payments, referred to as the *funding leg*. Both the PRDC coupon and the floating rates are applied on the domestic currency principal N_d. There are two parties involved in the swaps: the *issuer* of PRDC coupons (the receiver of the floating-rate payments – usually a bank) and the *investor* (the receiver of the PRDC coupons). We investigate PRDC swaps from the perspective of the issuer of PRDC coupons. Since a large variety of PRDC swaps are traded, for the sake of simplicity, only the basic structure is presented here. Consider the tenor structure

$$T_0 = 0 < T_1 < \cdots < T_{\beta-1} < T_\beta = T, \quad \nu_\alpha = \nu(T_{\alpha-1}, T_\alpha) = T_\alpha - T_{\alpha-1}, \quad \alpha = 1, 2, \ldots, \beta,$$

where ν_α represents the year fraction between $T_{\alpha-1}$ and T_α using a certain day counting convention, such as the Actual/365 one. Let $P_i(t, \overline{T})$, $i = d, f$, be the prices at time t in their respective currencies, of the "domestic" (d) and "foreign" (f) zero-coupon discount bonds with maturity $\overline{T}$. For use later in this section, define

$$T_{\alpha^+} = T_\alpha + \delta \text{ where } \delta \to 0^+, \quad T_{\alpha^-} = T_\alpha - \delta \text{ where } \delta \to 0^+,$$

i.e. T_{α^+} and T_{α^-} are instants of time just before and just after the date T_α in backward time, respectively. Note that the aforementioned T_{start} and T_{end} typically are the two consecutive dates $T_{\alpha-1}$ and T_α, respectively, of the swap's tenor structure.

For "vanilla" PRDC swaps, at each time $\{T_\alpha\}_{\alpha=1}^{\beta-1}$, there is an exchange of a PRDC coupon for a domestic LIBOR floating-rate payment. The PRDC coupon rate C_α, $\alpha = 1, 2, \ldots, \beta-1$, of the coupon amount $\nu_\alpha C_\alpha N_d$ issued at time T_α for the period $[T_\alpha, T_{\alpha+1}]$, $\alpha = 1, 2, \ldots, \beta-1$, has the structure

$$C_\alpha = \min\big(\max\big(c_f \frac{s(T_\alpha)}{f_\alpha} - c_d, b_f\big), b_c\big), \tag{5}$$

where c_d and c_f are domestic and foreign coupon rates; b_f and b_c are the floor and cap of the payoff. The scaling factor f_α usually is set to the forward FX rate $F(0,T_\alpha)$ defined by $F(0,T_\alpha) = \frac{P_f(0,T_\alpha)}{P_d(0,T_\alpha)} s(0)$, which follows from no-arbitrage arguments. All parameters in (5) can vary from coupon to coupon, i.e. they may depend on $\{T_\alpha\}_{\alpha=1}^{\beta-1}$. In the standard structure, in which $b_f = 0$ and $b_c = \infty$, by letting $h_\alpha = \frac{c_f}{f_\alpha}$ and $k_\alpha = \frac{f_\alpha c_d}{c_f}$, the coupon rate C_α can be viewed as a call option on FX rates, since

$$C_\alpha = h_\alpha \max(s(T_\alpha) - k_\alpha, 0). \tag{6}$$

In (6), the *option notional* h_α determines the overall level of the coupon payment, while the strike k_α determines the likelihood of the positiveness of the coupon. It is important to emphasize that, if the strike k_α is low, the coupon has a relatively high chance of paying a positive amount. However, in this case, the option notional h_α is typically chosen to be low and thus the overall level of a coupon payment is small. This is a *low-leverage* situation (to the investor). If k_α and h_α are high, then we have a *high-leverage* situation.

For $\alpha = 1, \ldots, \beta-1$, the funding leg pays the amount $\nu_\alpha L_d(T_{\alpha-1},T_\alpha) N_d$ for the period $[T_{\alpha-1}, T_\alpha]$, where $L_d(T_{\alpha-1},T_\alpha)$ denotes the domestic LIBOR rates for the period $[T_{\alpha-1}, T_\alpha]$, as observed at time $T_{\alpha-1}$. This rate is simply-compounded and is defined by $L_d(T_{\alpha-1},T_\alpha) = \frac{1-P_d(T_{\alpha-1}T_\alpha)}{\nu(T_{\alpha-1}T_\alpha)P_d(T_{\alpha-1},T_\alpha)}$. Note that $L_d(T_{\alpha-1}, T_\alpha)$ is set at time $T_{\alpha-1}$, but the actual floating leg payment for the period $[T_{\alpha-1}, T_\alpha]$ does not occur until time T_α, i.e. "in arrears".

If the interest rate for the domestic currency (e.g. Japanese Yen (JPY)) is low relative to the interest rate for the foreign currency (e.g. USD), the forward FX rate curve $F(0,t), t > 0$, decreases steeply as t increases, predicting a significant strengthening of the domestic currency. However, historical data suggests that the future spot FX rate will remain near its current level. This is reflected in the coupon rate formula (5): the investor receives a positive coupon at time T_α if $s(T_\alpha)$ is sufficiently large compared to $f_\alpha \equiv F(0,T_\alpha)$. Thus, we can view the investor as betting that the domestic currency is not going to strengthen as much as predicted by the forward FX rate curve.

We now consider the pricing of a "vanilla" PRDC swap. Let $u_\alpha^c(t)$ and $u_\alpha^f(t)$ be the values at time t of all PRDC coupons and floating payments, respectively, of a "vanilla" PRDC swap scheduled on or after $T_{\alpha+1}$. We are interested in the quantity $u_0^f(T_0) - u_0^c(T_0)$ as the value of the vanilla PRDC swap, taking into account the fact that the PRDC coupons being paid and the LIBOR payments being received. The funding leg can be priced via the "fixed notional" method, and not by solving the PDE, i.e. by equating each domestic LIBOR inflow $\nu_\alpha L_d(T_{\alpha-1}, T_\alpha) N_d$ at time $\{T_\alpha\}_{\alpha=1}^{\beta-1}$ to receiving and paying the amount N_d at times $T_{\alpha-1}$ and T_α, respectively. The PRDC coupon part of a "vanilla" PRDC swap can be viewed as a collection of simple FX options with different maturities $\{T_\alpha\}_{\alpha=1}^{\beta-1}$, and hence, the quantity $u_0^c(T_0)$ can be obtained by progressing backward in time from time $T_{\beta-1}$ to time T_0, as specified in Algorithm 1.

Algorithm 1 Algorithm for computing the coupon part of "vanilla" PRDC swaps.

1: set $u_{\beta-1}^c(T_{(\beta-1)^+}) = 0$;
2: **for** $\alpha = \beta-1, \ldots, 1$ **do**
3: set
$$u_{\alpha-1}^c(T_{\alpha^-}) = u_\alpha^c(T_{\alpha^+}) + \nu_\alpha C_\alpha N_d; \tag{7}$$
4: solve the PDE (3) backward in time with the terminal condition (7) from T_{α^-} to $T_{(\alpha-1)^+}$ using the ADI scheme (4) for each time τ_m, $m = 1, \ldots, l+1$, to obtain $u_{\alpha-1}^c(T_{(\alpha-1)^+})$.
5: **end for**
6: set $u_0^c(T_0) = u_0^c(T_0^+)$;

The most popular PRDC swaps are those with exotic features, such as *Bermudan cancelable*. In a Bermudan cancelable PRDC swap, the issuer of the PRDC coupons has the right to cancel the swap at any of the dates

$\{T_\alpha\}_{\alpha=1}^{\beta-1}$ *after* the occurrence of any exchange of fund flows scheduled on that date. Such exotic features are often included in PRDC swaps, since they appeal to both the investors, who want to receive a rate of return as high as possible, and to the issuers, who want to have protection against excessive movements in the FX rate. There usually is a high fixed coupon paid by the issuer to the investor at time T_0 that is not included in the definition above. This explains why exotic features, such as Bermudan cancelable, are attractive to investors. The size of the initial coupon is determined by the value to the issuer of the option to cancel the PRDC swap. We discuss this in more detail in the Subsection 6.2

The crucial observation in valuing Bermudan cancelable PRDC swaps is that terminating a swap at time T_α is the same as (i) continuing the underlying swap which is a "vanilla" PRDC swap, and (ii) entering into the offsetting (opposite) swap at time T_α, i.e. the swap with the reversed fund flows at each of the remaining dates $\{T_{\alpha+1}, \dots T_{\beta-1}\}$ of the tenor structure. Following this approach, the pricing of a Bermudan cancelable swap can be divided into the pricing of the underlying swap and the pricing of an option that gives the holder of the swap the right, but not an obligation, to cancel the swap, i.e. an option to enter the offsetting swap, at any of the dates $\{T_\alpha\}_{\alpha=1}^{\beta-1}$. This option is essentially a Bermudan swaption.

Following the above argument, we can regard a Bermudan cancelable PRDC swap as a "vanilla" PRDC swap that has the same tenor structure, referred to as the *underlying PRDC swap*, plus a long position in a Bermudan swaption, the underlying of which is a "vanilla" swap with the same tenor structure, but involves PRDC coupons being received and domestic floating payments being paid. We refer to this Bermudan swaption as the *offsetting Bermudan swaption* and its underlying swap as the *offsetting swap*. As a result, the pricing of a Bermudan cancelable PRDC swap can be divided into two subproblems: (i) the pricing of the underlying PRDC swap and (ii) the pricing of the associated offsetting Bermudan swaption described above. The subproblem (i) can be solved via Algorithm 1. Below, we discuss the pricing of the associated offsetting Bermudan swaption.

Denote by $u_\alpha^e(t)$ the value at time t of all fund flows in the offsetting swap scheduled on or after $T_{\alpha+1}$. By respective definition, the quantity $u_\alpha^e(t)$ can be computed by

$$u_\alpha^e(t) = -(u_\alpha^f(t) - u_\alpha^c(t)), \tag{8}$$

which in turn can be obtained via the pricing of the underlying PRDC swap. Let $u_\alpha^h(t)$ be the value at time t of the offsetting Bermudan swaption that has only the dates $\{T_{\alpha+1}, \dots, T_{\beta-1}\}$ as exercise opportunities. Assume optimal exercise at each of $\{T_\alpha\}_{\alpha=1}^{\beta-1}$. That is, the PRDC coupon issuer will exercise the offsetting Bermudan swaption at T_α if and only if the value $u_\alpha^e(T_\alpha)$ (the "exercise value") exceeds the value $u_\alpha^h(T_\alpha)$ (the "hold value") of the option to delay exercise to the future. Thus the payoff of the offsetting Bermudan swaption at each of $\{T_\alpha\}_{\alpha=1}^{\beta-1}$ is $\max(u_\alpha^h(T_\alpha), u_\alpha^e(T_\alpha))$. A backward pricing algorithm for the associated offsetting Bermudan swaption is described

Algorithm 2 Algorithm for computing the offsetting Bermudan swaption.

1: set $u_{\beta-1}^h(T_{(\beta-1)^+}) = 0$ and $u_{\beta-1}^e(T_{(\beta-1)^+}) = 0$;
2: **for** $\alpha = \beta - 1, \dots, 1$ **do**
3: set
$$u_{\alpha-1}^h(T_{\alpha^-}) = \max(u_\alpha^h(T_{\alpha^+}), u_\alpha^e(T_{\alpha^+})); \tag{9}$$
4: solve the PDE (3) backward in time with the terminal condition (9) from T_{α^-} to $T_{(\alpha-1)^+}$ using the ADI scheme (4) for each time τ_m, $m = 1, \dots, l+1$, to obtain $u_{\alpha-1}^h(T_{(\alpha-1)^+})$;
5: compute $u_{\alpha-1}^e(T_{(\alpha-1)^+})$ by (8), where $u_{\alpha-1}^c(T_{(\alpha-1)^+})$ is computed from Line 4 of Algorithm 1, and $u_{\alpha-1}^f(T_{(\alpha-1)^+})$ is computed by the "fixed notional" method;
6: **end for**
7: set $u_0^h(T_0) = u_0^h(T_{0^+})$;

in Algorithm 2. The value of the offsetting Bermudan swaption is $u_0^h(T_0)$, and the value of the Bermudan cancelable PRDC swap is $u_0^h(T_0) + (u_0^f(T_0) - u_0^c(T_0))$.

We conclude this section by noting that, from the point of view of designing a parallel algorithm, by dividing the pricing of Bermudan cancelable PRDC swaps into two independent pricing subproblems during each period of

the tenor structure, we can run the two pricing processes, each for a subproblem, in parallel with communications only at $\{T_\alpha\}_{\alpha=1}^{\beta-1}$. In Section 5, we describe in more detail how this approach is very well-suited for the parallel pricing of Bermudan cancelable PRDC swaps on a multi-GPU platform.

5. GPU-based parallel pricing of Bermudan cancelable PRDC swaps

5.1. GPU device architecture

In this subsection, we summarize some key properties of the GPU device architecture and the CUDA framework necessary to understand our implementation of the parallelization of the ADI schemes. The modern GPU can be viewed as a set of independent streaming multiprocessors (SMs) [12]. One such SM contains, amongst others, several scalar processors which can execute integer and single-precision floating-point arithmetic, a multi-threaded instruction unit (I/U), and shared memory. Typically, the CPU (the *host*) runs the program skeleton and offloads the more computationally demanding code sections to the GPUs (the *device*). Functions that run on the device are called *kernels*. When a program invokes a kernel, many copies of this kernel, which are referred to as *threads*, are distributed to the available multiprocessors, where they are executed. Since all threads of a parallel phase execute the same code, the programming model of CUDA is an instance of the widely used Single Instruction Multiple Data (SIMD) parallel programming style. Within the CUDA framework, operations are performed by threads that are grouped into *threadblocks*, which are in turn arranged on a *grid*. It is important to note that threads within the same threadblock are able to communicate with each other very efficiently via the shared memory. Furthermore, all threads in a threadblock are able to synchronize their executions. On the other hand, threads belonging to different threadblocks are not able to communicate efficiently with each other, nor to synchronize their executions. Threads are executed in groups of 32 threads, referred to as *warps*. A group of 16 threads is a *half-warp*. Threads of one warp are handled on the same multiprocessor. If the threads of a given warp diverge by a data-induced conditional branch, each branch of the warp will be executed serially and the processing time of the given warp consists of the sum of the branches processing times. This must be avoided when possible.

The NVIDIA Tesla series is the first family of GPUs that is dedicated to general purpose computing. The NVIDIA Tesla 10-series (T10) GPUs (Tesla S1060/S1070 - server version), which are used for the experiments in this paper, consists of 30 independent SMs, each containing 8 processors running at 1.44GHz, a total of 16384 registers, and 16 KB of shared memory.

5.2. GPU-based parallel pricing framework

In a typical GPU code, there usually are two levels of parallelism. The first one is between the CPU and the GPU(s) that conforms to the Master-Slave model, in which the CPU and the GPU(s) play the roles of the master and the slave(s), respectively. The second level of parallelism is within a GPU itself; it conforms to the Peer model. At this level, each thread, playing the role of a peer, executes the same program, possibly with different data.

In pricing Bermudan cancelable PRDC swaps, for each period $[T_{(\alpha-1)^+}, T_{\alpha^-}]$, we need to solve backward in time the PDE (3) from T_{α^-} to $T_{(\alpha-1)^+}$ with payoffs (7) and (9) to obtain $u^c_{\alpha-1}(T_{(\alpha-1)^+})$ and $u^h_{\alpha-1}(T_{(\alpha-1)^+})$, respectively (Lines 4 and 5 of Algorithm 2). These two pricing processes are entirely independent within each time period of the tenor structure. Thus, for each period, it is natural to assign the two pricing processes to separate GPUs. More specifically, for each period, the GPU-based pricing of a Bermudan cancelable PRDC swap consists of three phases. During the first phase, the two pricing processes are assigned to separate GPUs. During the second phase, the PDE (3) is solved simultaneously in each GPU from T_{α^-} to $T_{(\alpha-1)^+}$ with terminal condition (7) or (9), respectively, using the Peer model. The parallelism in each GPU for this phase is based on an efficient parallelization of the computation

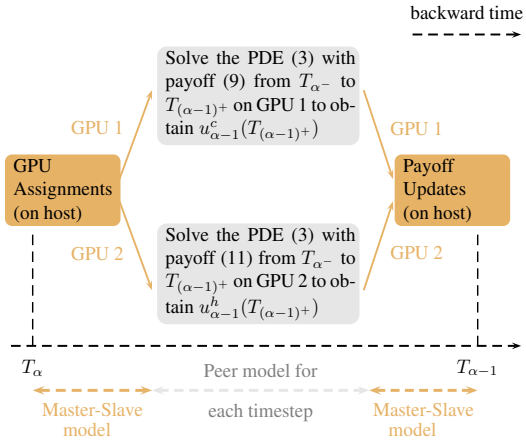

Figure 1: Three phases of the pricing of PRDC swaps with Bermudan cancelable features over each time period $[T_{(\alpha-1)^+}, T_{\alpha^-}]$ of the tenor structure and the associated parallelization paradigm for each phase.

of each timestep of the ADI timestepping technique (4.1)-(4.4). During the last phase, the host (the CPU) collects the pricing results for the past period $[T_{(\alpha-1)^+}, T_{\alpha^-}]$ and updates the payoffs for the next period $[T_{(\alpha-2)^+}, T_{(\alpha-1)^-}]$. Figure 1 provides an illustration of our approach. Note that two pieces of a single-GPU code run on a multi-GPU platform, each for the solution of a PDE. It is desirable to start both kernels at the same time and run them simultaneously. In the CUDA framework, this can be achieved by implementing threads in the host program and dispatching them to different GPUs using the `cudaSetDevice(.)` function.

5.3. GPU implementation

In this subsection, we discuss a GPU-based parallel algorithm for the solution of each PDE problem. We emphasize that we do not aim to parallelize across time, but rather we focus on the parallelism within each timestep only. To step from time τ_{m-1} to time τ_m, we apply the ADI scheme (4.1)-(4.5). Steps (4.1)-(4.4) can be grouped into two phases: (i) the first consists of a forward Euler step (predictor step (4.1)), followed by three implicit, but unidirectional, corrector steps (4.2), the purpose of which is to stabilize the predictor step; and (ii) the second phase (4.3)-(4.4) which aims at restoring the second-order of convergence of the discretization method. Step (4.5) is essentially trivial. With respect to the CUDA implementation, the two phases are essentially the same; they can both be decomposed into matrix-vector multiplications and solving independent tridiagonal systems. Hence, for brevity, we focus on describing a GPU parallelization of the first phase (4.1)-(4.2) only. A more detailed discussion of the GPU-based parallel ADI timestepping algorithm for the solution of multi-dimensional linear parabolic PDEs can be found in an earlier work of ours [5].
For presentation purposes, let

$$\mathbf{w}_i^{m-1} = \Delta\tau \mathbf{A}_i^{m-1}\mathbf{u}^{m-1} + \Delta\tau(\mathbf{g}_i^{m-1} - \mathbf{g}_i^m), \quad i = 0,1,2,3,$$
$$\widehat{\mathbf{A}}_i^m = \mathbf{I} - \frac{1}{2}\Delta\tau\mathbf{A}_i^m, \qquad \widehat{\mathbf{v}}_i^{m-1} = \mathbf{v}_{i-1} - \frac{1}{2}\mathbf{w}_i^{m-1}, \quad i = 1,2,3,$$

and notice that $\mathbf{v}_0 = \mathbf{u}^{m-1} + \sum_{i=0}^{3}\mathbf{w}_i^{m-1} + \Delta\tau\mathbf{g}^m$. The CUDA implementation of the first phase consists of the following steps: (i) Step a.1: Compute the matrices $\mathbf{A}_i^m$, $i = 0,1,2,3$, and $\widehat{\mathbf{A}}_i^m$, $i = 1,2,3$, and the vectors $\mathbf{w}_i^{m-1}$, $i = 0,1,2,3$, and $\mathbf{v}_0$; (ii) Step a.2: Set $\widehat{\mathbf{v}}_1^{m-1} = \mathbf{v}_0 - \frac{1}{2}\mathbf{w}_1^{m-1}$ and solve $\widehat{\mathbf{A}}_1^m\mathbf{v}_1 = \widehat{\mathbf{v}}_1^{m-1}$; (iii) Step a.3: Set $\widehat{\mathbf{v}}_2^{m-1} = \mathbf{v}_1 - \frac{1}{2}\mathbf{w}_2^{m-1}$ and solve $\widehat{\mathbf{A}}_2^m\mathbf{v}_2 = \widehat{\mathbf{v}}_2^{m-1}$; (iv) Step a.4: Set $\widehat{\mathbf{v}}_3^{m-1} = \mathbf{v}_2 - \frac{1}{2}\mathbf{w}_3^{m-1}$ and solve $\widehat{\mathbf{A}}_3^m\mathbf{v}_3 = \widehat{\mathbf{v}}_3^{m-1}$.

5.3.1. First phase - Step a.1

We assume that all data needed for this step, such as the data of the previous timestep, i.e. the vector $\mathbf{u}^{m-1}$, the model constant parameters, are available in the global or constant memory. We emphasize that the data copying from the host memory to the device memory occurs only once on the first timestep of each period of the swap's tenor structure. Data for subsequent timesteps of that tenor structure's period and for the steps of the ADI timestepping scheme (4) are stored on the device memory. The host-to-device copying can be achieved via CUDA functions `cudaMemcpy2D(.)` for data and `cudaMemcpyToSymbol(.)` for constants.

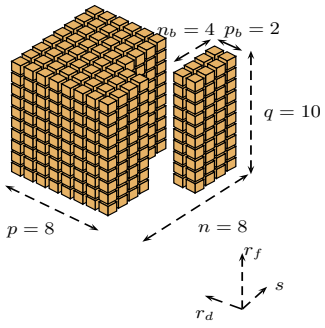

Figure 2: An illustration of the partitioning approach considered for Step a.1. The computational domain is partitioned into 3-D blocks of size $n_b \times p_b \times q \equiv 4 \times 2 \times 10$, each of which can be viewed as consisting of ten 4×2 tiles, or as $8(= 4 \times 2)$ stacks of 10 gridpoints.

We partition the computational grid of size $n \times p \times q$ into 3-D blocks of size $n_b \times p_b \times q$, each of which can be viewed as consisting of q two-dimensional (2-D) blocks, referred to as *tiles*, of size $n_b \times p_b$. For Step a.1, we let the kernel generate a $\texttt{ceil}\left(\frac{n}{n_b}\right) \times \texttt{ceil}\left(\frac{p}{p_b}\right)$ grid of threadblocks, where `ceil` denotes the ceiling function. Each of the threadblocks, in turn, consists of a total of $n_b p_p$ threads arranged in 2-D arrays, each of size $n_b \times p_b$. All gridpoints of a $n_b \times p_b \times q$ 3-D block are assigned to one 2-D threadblock only, with one thread for each "stack" of q gridpoints in the r_f direction. Since each 3-D block has a total of q $n_b \times p_b$ tiles and each 2-D threadblock is of size $n_b \times p_b$, the partitioning

approach that we use here suggests a q-iteration loop in the kernel. During the kth iteration, tiles on the kth s-r_d plan, i.e. tiles corresponding to r_{fk}, are processed. It is important to emphasize that the value $n_b p_b$ is chosen so that the number of 3-D blocks is large enough to take advantage of all available multiprocessors. Figure 2 illustrates an application of the aforementioned partitioning approach on an example computational grid of size $n \times p \times q \equiv 8 \times 8 \times 10$ with $n_b = 4$ and $p_b = 2$.

Regarding the data loading strategy, note that (i) threads within the same threadblock can communicate with each other effectively via the shared memory, while threads in different threadblocks cannot; (ii) accessing the global memory is costly, especially if access is not coalesced; and (iii) each threadblock needs, besides its gridpoints' values, the corresponding *halo* values, i.e. the values of neighboring gridpoints from adjacent tiles in the s- and r_d-directions, or the boundary conditions on all of its sides. Since the halo values of a threadblock belong to different threadblocks, it is desirable to have each threadblock load its halo values from the global memory into its part of the shared memory. Taking into account these considerations, we adopt the following data loading approach: during each iteration of the q-iteration loop, each thread of a threadblock loads the data associated with one gridpoint, plus either one halo, if the gridpoint is at an edge of a tile, or two halos, if the gridpoint is at a corner of a tile. These values may be accessed by neighbouring threads in the same threadblock. Compared to a naive approach which allows each thread, during each iteration, to read in from global memory its gridpoint's old data and all of its neighboring gridpoint's old data required by the FD scheme, our approach has the advantages of (i) reducing repeated loadings of old data from the global memory, and (ii) allowing (partial) memory coalescing.

In Figure 3, an example illustrating the aforementioned data loading approach for 8×8 tiles (marked in ⊠) with halos (marked in □) is presented. Only threads whose gridpoints are close to the boundary of a tile need to load into the shared memory the halo values or the boundary values in addition to their own gridpoint data. As can be seen from Figure 3, the data on the North, South, East, West boundaries of a tile are loaded only twice, except that the data at the corners are accessed four times. Also, threads corresponding to gridpoints at the corners of each tile are responsible for reading in three halo values. This loading approach is partially coalesced, as discussed toward the end of this section.

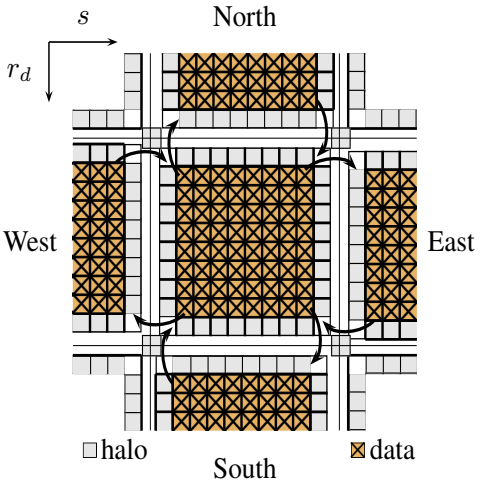

Figure 3: An example of $n_b \times p_b = 8 \times 8$ tiles with halos.

Regarding the construction of the matrices $\mathbf{A}_i^m$, $i = 0, 1, 2, 3$, and $\widehat{\mathbf{A}}_i^m$, $i = 1, 2, 3$, at each iteration, we assign each of the threads to assemble one row of each of the matrices, a total of three entries, since these matrices are tridiagonal. Regarding the computations of the vectors $\mathbf{w}_i^{m-1}$, $i = 0, 1, 2, 3$, and $\mathbf{v}_0$, due to the FD discretization scheme in the r_f-direction, to calculate the values of gridpoints of a tile on the kth s-r_d plane, i.e. gridpoints that correspond to r_{fk}, some data of the corresponding tiles on the $(k-1)$st and $(k+1)$st s-r_d planes are needed as well. Since 16KB of shared memory available per multiprocessor is not sufficient to store a large number of data tiles, each threadblock works with three data tiles of size $n_b \times p_b$ and their halo values, if any, during each iteration of the loop. More specifically, during the kth iteration of the kernel's q-iteration loop, each threadblock (i) first loads into shared memory the corresponding 2-D tile on the $(k+1)$st s-r_d plane, and the associated halos (in the s- and r_d-directions, if any, in the (partially) coalesced manner discussed earlier, (ii) then computes and stores new values for its tile on the kth s-r_d plane using data of the corresponding tiles on the $(k-1)$st, kth, $(k+1)$st s-r_d planes, and of associated halos, if any.

Barrier synchronization among threads in the same threadblock, achieved by the function `__syncthreads()`, is used in both the loading and computing phases of each iteration of the q-iteration loop in the kernel. Barrier synchronization ensures that all threads in the same threadblock have completed a phase (e.g., loading the data) before any of them move to the next phase (e.g., accessing the data for the computations).

Regarding coalesced data loading from the global memory, certain necessary conditions, such as (i) the threads in the half-warp must access consecutive global memory locations and (ii) the thread numbering matters only along the first dimension of the threadblock, must be satisfied for the hardware to perform coalesced data loads. We refer

interested readers to [12] for a more complete description of all the requirements. Regarding the loading phase for $\mathbf{v}_0$, to ensure the data transfer coalescing, it is necessary to have the tile size in the s-direction, i.e. n_b, a multiple of 16, since each half-warp is of size 16 and that gridpoints at this step are ordered in the s-direction first. Using our loading strategy, the interior and the halos along the s-direction (i.e. North and South halos in Figure 3) of the data tiles can be loaded in a coalesced way. However, halos along the r_d-direction (i.e East and West halos in Figure 3) cannot be accessed via a coalesced pattern, since they belong to different vectors. As a result, the data loading approach for Step a.1 is not fully coalesced. However, as timing results indicate, it turns out to be effective.

REMARK 2. Due to the limited shared memory, in the current implementation of Step a.1, we work with only three 2-D planes at a time. Having a considerably larger amount of shared memory could (i) yield performance improvements or (ii) lead to a better design of the parallel algorithm for this step. Regarding (i), in the current implementation, a larger amount of shared memory could enable more threadblocks to be scheduled to run concurrently on the same multiprocessor, which might improve the efficiency of the parallel algorithm. With respect to (ii), one could extend the algorithm presented in this paper to work with more than three 2-D planes, i.e. larger blocks of data points. Such an approach allows grouping together several references to the global memory, which may result in lower total cost of accessing the global memory. However, this partitioning approach may allow a smaller number of threadblocks to simultaneously run on the same multiprocessor than that of the three-plane strategy. As a result, it is not obvious whether a parallel algorithm based on this approach would be more efficient than the three-plane strategy presented in this paper on a GPU with larger shared memories. We plan to investigate the efficiency of the aforementioned two approaches in a future paper.

In the current implementation, each thread first loads in its data and associated halos, if any, from graphics to shared memory for use by other threads in the same threadblock, then computes its new values and writes them back to graphics memory, because L1/L2 cache is not available on the NVIDIA Tesla T10. On new generation of GPUs, such as the one based on the NVIDIA "Fermi" architecture, the availability of L1/L2 cache is expected to not only simplify the programming of Step a.1, as references to the global memory will be automatically cached without the need of copying to shared memory, but also improve the performance of the parallel algorithm, since access to cache is usually faster than access to shared memory.

5.3.2. First phase - Steps a.2, a.3, a.4

For each of the Steps a.2, a.3 and a.4, the vectors $\widehat{\mathbf{v}}_i^{m-1}$, $i = 1, 2, 3$, can be computed using the vectors $\mathbf{w}_i^{m-1}$, $i = 1, 2, 3$, already computed in Step a.1, and the vectors $\mathbf{v}_{i-1}$, $i = 1, 2, 3$, computed in the previous ADI step, i.e. in Steps a.1, a.2 and a.3, respectively. Data between Steps a.1, a.2, a.3 and a.4 are held in the device memory, hence the parallel solution of the tridiagonal systems $\widehat{\mathbf{A}}_i^m \mathbf{v}_i = \widehat{\mathbf{v}}_i^{m-1}$, $i = 1, 2, 3$, can be achieved via different data partitioning as follows. As previously noted, each of the matrices $\widehat{\mathbf{A}}_i^m$ is block-diagonal with tridiagonal blocks. For example, the $\widehat{\mathbf{A}}_1^m$ matrix has pq diagonal blocks, and each block is tridiagonal of size $n \times n$, while the $\widehat{\mathbf{A}}_2^m$ matrix has nq diagonal blocks, each block is tridiagonal of size $p \times p$. Our approach for the solution of $\widehat{\mathbf{A}}_i^m \mathbf{v}_i = \widehat{\mathbf{v}}_i^{m-1}$, $i = 1, 2, 3$, is based on the parallelism of independent tridiagonal solutions, rather than the parallelism within each one. When we solve in one direction, the data are partitioned with respect to the other two, which results in several independent tridiagonal systems. The solution of each of the independent tridiagonal systems is assigned to a single thread. For example, the solution of $\widehat{\mathbf{A}}_1^m \mathbf{v}_1 = \widehat{\mathbf{v}}_1^{m-1}$ is executed by first partitioning $\widehat{\mathbf{A}}_1^m$ and $\widehat{\mathbf{v}}_1^{m-1}$ into pq independent $n \times n$ tridiagonal systems, and then assigning them to pq threads.

Since the number of threads in a threadblock is limited to 512, we use multiple 2-D threadblocks for the solution of the independent tridiagonal systems. In our implementation, each of the 2-D threadblocks used in these steps has identical size $r_t \times c_t$, where the values of r_t and c_t are determined by numerical experiments to maximize the performance. The size of the grid of threadblocks is determined accordingly. For example, for the parallel solution of $\widehat{\mathbf{A}}_1^m \mathbf{v}_1 = \widehat{\mathbf{v}}_1^{m-1}$, a 2-D grid of threadblocks of size $\texttt{ceil}(\frac{p}{r_t}) \times \texttt{ceil}(\frac{q}{c_t})$ is invoked.

Regarding the memory coalescing for these steps, the data between Steps a.1, a.2, a.3 and a.4 are ordered in the s-, r_d-, then r_f-directions. As a result, only the data partitionings for the tridiagonal solves of the systems $\widehat{\mathbf{A}}_i^m \mathbf{v}_d = \widehat{\mathbf{v}}_i^{m-1}$, $i = 2, 3$, i.e. solves in the r_d- and r_f-directions, allow full memory coalescence.

REMARK 3. There are at least two possible approaches to improve the efficiency of Steps a.2, a.3 and a.4. First, under the current framework, full memory coalescence for the tridiagonal solves in all three directions could be achieved by renumbering the gridpoints between Steps a.1, a.2, a.3 and a.4. However, such a renumbering will involve some overhead. In a future paper, we plan to investigate the trade-off between this overhead and achieving memory coalescence in two of the three directions only. Second, one can employ GPU-based parallel cyclic reduction methods for the solution of the tridiagonal systems arising in these steps [17]. These techniques have been shown to be more efficient and scalable than the technique adopted in this paper. However, a GPU-based implementation of these techniques is much more involved than that of the approach presented above.

6. Numerical results

We consider the same interest rate, correlation parameters, and local volatility function as given in [13]. The domestic (JPY) and foreign (USD) interest rate curves are given by $P_d(0,t) = \exp(-0.02 \times t)$ and $P_f(0,t) = \exp(-0.05 \times t)$. The volatility parameters for the short rates and correlations are given by $\sigma_d(t)$=0.7%, $\kappa_d(t)$=0.0%, $\sigma_f(t)$=1.2%, $\kappa_f(t)$=5.0%, ρ_{df}=25%, ρ_{ds}=−15%, ρ_{fs}=−15%. The initial spot FX rate is set to $s(0) = 105.00$. The parameters $\xi(t)$ and $\varsigma(t)$ to the local volatility function are assumed to be piecewise constant and given in the following table (Table C in [13]):

period(year)	(0, 0.5]	(0.5, 1]	(1, 3]	(3, 5]	(5, 7]	(7, 10]	(10, 15]	(15, 20]	(20, 25]	(25, 30]
$\xi(t)$	9.03%	8.87%	8.42%	8.99%	10.18%	13.30%	18.18%	16.73%	13.51%	13.51%
$\varsigma(t)$	-200%	-172%	-115%	-65%	-50%	-24%	10%	38%	38%	38%

Note that the forward FX rate, the initial term structures $\theta_i(t)$, $i = d, f$, and the domestic LIBOR rates are fully determined by the above information. We consider a PRDC swap with the following features:
- Tenor structure: $\nu_\alpha = T_\alpha - T_{\alpha-1} = 1$ (year), $\alpha = 1, \ldots, \beta - 1$ and $\beta = 30$ (year).
- Pay annual PRDC coupons and receive annual domestic LIBOR payments.
- Standard structure, i.e. $b_f = 0$, $b_c = +\infty$. The scaling factor f_α is set to $F(0, T_\alpha)$, $\alpha = 1, \ldots \beta - 1$.
- Bermudan cancelable feature, which allows the issuer to cancel the swap on each of T_α, $\alpha = 1, \ldots \beta - 1$.
- The domestic and foreign coupons are chosen to provide different levels of leverage: low ($c_d = 2.25\%, c_f = 4.50\%$), medium ($c_d = 4.36\%, c_f = 6.25\%$), high ($c_d = 8.1\%, c_f = 9.00\%$).

The truncated computational domain Ω is defined by setting $S = 3s(0) = 315$, $R_d = 3r_d(0) = 0.06$, and $R_f = 3r_f(0) = 0.15$. Grid sizes indicated are for each period $[T_{(\alpha-1)^+}, T_{\alpha^-}]$. The values of swaps are expressed as a percentage of the notional N_d.

We used the CUDA 3.2 driver and toolkit, and all the experiments with the GPU code were conducted on two NVIDIA Tesla T10 GPUs connected to a two quad-core Intel "Harpertown" host system with Intel Xeon E5430 CPUs running at 2.66GHz with 8GB of FB-DIMM PC 5300 RAM. Note that only one CPU core was employed for the experiments with the (non-multithreaded) CPU code written by us. All computations are carried out in single-precision. The size of each tile used in Step a.1 is chosen to be $n_b \times p_b \equiv 32 \times 4$, and the size of each threadblock used in the parallel solution of the independent tridiagonal systems in Steps a.2, a.3 and a.4 is $r_t \times c_t \equiv 32 \times 4$, which appears to be optimal on a Tesla T10.

6.1. Performance comparison

In this subsection, we focus on the GPU versus CPU performance comparison. A detailed analysis of the pricing results is provided in the next subsection. Table 1 presents some selected numerical and timing results for the low-leverage case under the FX skew model presented in this paper. The timing results for the medium- and high-leverage cases are approximately the same. The CPU and GPU computation times, respectively denoted by "CPU time" and "GPU time", measure the total computational times in seconds (s.) required for solving the PDE over 29 periods of the tenor structure. They were obtained using the CUDA functions `cutStartTimer(.)` and `cutStopTimer(.)` for each period. The GPU times include the overhead for memory transfers from the CPU to the device memory. The relative speedup factor ("speed up") is defined as the ratio of the CPU times

				underlying PRDC swap				cancelable PRDC swap			
l	n	p	q	value	CPU	GPU	speed	value	CPU	GPU	speed
(t)	(s)	(r_d)	(r_f)	(%)	time (s.)	time(s.)	up	(%)	time (s.)	time (s.)	up
4	24	12	12	-11.1510	1.2	0.6	2.4	11.2936	2.2	0.6	4.9
8	48	24	24	-11.1205	19.8	1.6	12.4	11.2829	40.1	1.6	25.1
16	96	48	48	-11.1118	324.3	11.9	27.2	11.2806	648.5	11.9	54.5
32	192	96	96	-11.1094	5301.1	180.6	29.3	11.2801	10601.2	180.6	58.7

Table 1: Values and performance for pricing of the underlying PRDC swap and cancelable PRDC swap for the low-leverage case.

over the corresponding GPU times. The CPU times for pricing a cancelable PRDC swap include the times needed for computing the underlying PRDC swap. The CPU times in this case are the sum of timing results over all periods obtained by first running a CPU-based solver for the price of the underlying swap, and then another CPU-based solver for the price of the offsetting Bermudan swaption. The GPU-based pricing of a Bermudan cancelable PRDC swap was conducted on two NVIDIA Tesla T10 GPUs simultaneously, one of which was used for pricing the underlying PRDC swap. The GPU times for pricing cancelable swaps are taken to be the larger of the two GPU times needed for computing the underlying swap and the associated offsetting Bermudan swaption. Note that we started both GPU-based PDE solvers for the underlying PRDC swap and for the offsetting Bermudan swaption at the same time, and as we expected, they finished just as quickly as running a single GPU-based solver. As is evident from Table 1, when pricing the underlying, the GPU is significantly faster than the CPU for every size considered of the discretized problem; the asymptotic speedup is about 30 for the largest grid we considered. When pricing cancelable swaps, we obtained speedups that are approximately double those obtained when pricing the underlying, with an asymptotic speedup about 60 for the largest grid we considered, as expected since we used two GPU cards for the two independent pricing processes.

6.2. Analysis of pricing results and effects of FX volatility skew

In Table 2, we present pricing results for the underlying and cancelable swaps. Since we do not have reference values for the prices of these swaps, to show convergence, we compute the "change" as the difference in values from the coarser grid and the "ratio" as the ratio of changes between successive grids. The numerical results indicate second-order convergence is achieved for the ADI scheme, as expected. Note that negative values for the underlying swap indicate the price that the investor has to pay to the coupon issuer to enter into a "vanilla" PRDC swap, while positive values for the cancelable swap indicate the level of the initial fixed coupon that the issuer is willing to pay to the investor. For instance, for the low-leverage case under the FX skew model, the issuer of a cancelable PRDC swap should pay a net coupon of about 11.2801% of the notional to the investor, while in the high-leverage case, this amount is about 19.6402% of the notional. (Of course, the issuer would prefer to pay less and keep the rest as profit.) Among the three leverage cases, the high-leverage case is more attractive to the JPY investor, due to the high initial coupon.

To investigate the effects of the FX skew, we compare prices of the underlying under the FX skew model with those obtained using the log-normal model, as suggested in [13]. In the log-normal model, the local volatility function is a deterministic function of only the time variable, and not of the FX rate. To this end, we used the parametrization as in (2) but independent of $s(t)$ for the log-normal local volatility function, and calibrated it to the same at-the-money FX option data (Table A of [13]) used for the calibration of the FX skew model. The numerical results for the log-normal and the FX skew models using the finest mesh in Table 2 are presented in Table 3.

First, let us study the effect of the FX skew on the underlying swaps. All the underlying swaps have roughly the same value under the log-normal model across leverage levels. It is important to emphasize that due to the rate differential between JPY and USD, the forward FX curve is strongly downward sloping, hence the forward FX rate f_α is considerably smaller than the spot FX rate $s(T_\alpha)$. Thus, the coupon issuer essentially shorts a collection of FX call options with low strikes. (For the low-, medium-, and high-leverage cases, the strike k_α is set to 50%, 70% and 90% of f_α, respectively, hence is significantly less than $s(T_\alpha)$.) The numerical results indicate that the

leverage	l	n	p	q	underlying PRDC swap			cancelable PRDC swap		
	(t)	(s)	(r_d)	(r_f)	value (%)	change	ratio	value(%)	change	ratio
low	4	24	12	12	-11.1510			11.2936		
	8	48	24	24	-11.1205	3.0e-4		11.2829	1.1e-4	
$(c_d/c_f = 50\%)$	16	96	48	48	-11.1118	8.6e-5	3.6	11.2806	2.3e-5	4.4
	32	192	96	96	-11.1094	2.4e-5	3.7	11.2801	5.8e-6	4.0
medium	4	24	12	12	-12.9418			13.6638		
	8	48	24	24	-12.7495	1.9e-3		13.8012	1.3e-3	
$(c_d/c_f = 70\%)$	16	96	48	48	-12.7033	4.6e-4	4.1	13.8399	3.9e-4	3.5
	32	192	96	96	-12.6916	1.2e-4	3.9	13.8507	1.1e-4	3.6
high	4	24	12	12	-11.2723			19.3138		
	8	48	24	24	-11.2097	6.2e-4		19.5689	2.5e-3	
$(c_d/c_f = 90\%)$	16	96	48	48	-11.1932	1.4e-4	3.8	19.6256	5.6e-4	4.4
	32	192	96	96	-11.1889	4.3e-5	3.8	19.6402	1.4e-4	3.8

Table 2: Values of the underlying PRDC swap and cancelable PRDC swap for various leverage levels with FX skew model.

leverage	FX skew		Log-normal		FX skew - log-normal	
	underlying	cancelable	underlying	cancelable	underlying	cancelable
low	-11.1094	11.2801	-9.0128	13.3128	-2.0966	-2.0327
medium	-12.6916	13.8507	-9.6773	16.8985	-3.0143	-3.0478
high	-11.1889	19.6402	-9.8538	22.9523	-1.3351	-3.3121

Table 3: Values of the underlying PRDC swap and cancelable PRDC swap for various leverage levels with FX skew model and log-normal model using the finest mesh in Table 2.

prices of the underlying swap under the FX skew model are more negative than the prices under the log-normal model, i.e. under a FX skew model, the investor has to pay more to the issuer to enter a vanilla PRDC swap. These results are expected, since, in a skew model, the implied volatility increases for low-strike options, resulting in higher prices for the options and hence pushes down the value of the underlying swap for the issuer. However, the effect of the skew is not uniform across the leverage levels. The effect seems most pronounced for the medium-leverage swaps (a difference of -3.0143 as opposed to -2.0966 and -1.3351). An explanation for this observation is that the total effect is a combination of the change in implied volatility and the sensitivities (the Vega) of the options to that change. Due to the skew, the lower the strikes are, the higher the implied volatility changes are. Thus, among the three leverage levels, the volatilities change the most for the low-leverage swaps, since the strikes of the coupon rates are the lowest in this case. However, the Vega of an option is an increasing function of the strike [8]. Thus, the Vega for low-leverage options is the smallest, since the strikes for coupon rates are the lowest. As a result, the combined effect is limited. The situation is reversed for high-leverage swaps, while the combined effect is the most pronounced for medium-leverage swaps.

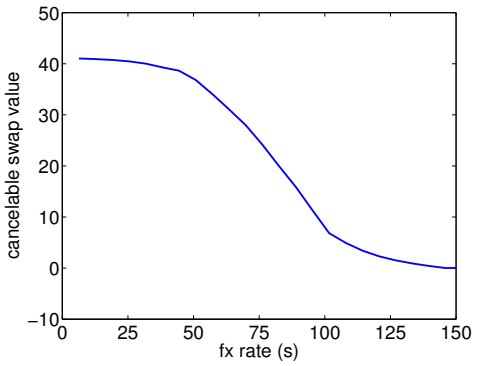

Figure 4: Values of the Bermudan cancelable PRDC swap, in percentage of N_d, as function of spot FX at time $T_\alpha = 5$ with high-leverage coupons.

Next, we investigate the effect of the FX skew on the cancelable swaps. To understand the importance of the FX skew for cancelable PRDC swaps, let us look at the value of the cancelable swap at an intermediate date of the tenor structure as a function of the spot FX rate on that date. In Figure 4, a sample plot of such a function for the high-leverage case at $T_\alpha = 5$ is given. The forward FX rate $F(0, T_\alpha)$ is about 90.3 on that date. When the spot FX rate is less than the forward FX rate ($s < 90.3$), it is evident that the value function is concave down

(negative gamma), which agrees with the interpretation that the issuer has a short position in (low-strike) FX call options, since the swap is not canceled. However, if the spot FX rate is high enough, it would be optimal to cancel, an observation that is reflected by the convexity of the value function (positive gamma), due to the long position in (high-strike) FX call options (options to cancel). Thus, the profile of a cancelable PRDC swap is similar to a bear spread created by calls, the payoff of which is known to be very sensitive to the volatility skew. As evident from Table 3, by accounting for the FX skew, the values of cancelable PRDC swaps under the FX skew model are less positive than the values under the log-normal model, a fact indicating that the initial net coupons paid to the investor under the skew model are smaller than those paid under the log-normal model. These changes in values are viewed as profits booked by the issuer. Regarding the effects of the different leverage levels on the values of cancelable PRDC swap, under both the FX skew model and the log-normal model, the values of cancelable PRDC swaps increase with leverage levels: from 11.2801% to 19.6402% under the skew model and from 13.3128% to 22.9523% under the log-normal model. This behavior is expected, due to a positive correlation between the leverage and volatility levels. In addition, due to the sensitivity of the prices of cancelable PRDC swaps to the volatility skew, as explained above, the difference between the values of the FX skew and log-normal models is uniformly increasing in absolute terms across the leverage levels (-2.0327%, -3.0478%, and -3.3121% for low-, medium-, and high-leverage levels, respectively).

7. Conclusions and future work

This paper presents a GPU-based algorithm for pricing exotic cross-currency interest rate derivatives under a FX local volatility skew model via a PDE approach, with emphasis on Bermudan cancelable PRDC swaps. The algorithm is based on an efficient parallelization of the ADI scheme at each timestep and has shown to be effective for pricing such derivatives. By partitioning the pricing of Bermudan cancelable PRDC swaps into two entirely independent pricing subproblems in each period of the tenor structure, we can efficiently utilize two GPUs to linearly scale the speedup when pricing the underlying PRDC swap. The analysis shows that the impact of the skew on cancelable PRDC swaps is significant.

At the time of writing this paper, more powerful GPUs, with more processors, such as the NVIDIA Tesla 20-series, based on the "Fermi" architecture, became available on the market. The increase in the number of parallel processors (448 processors in the Tesla C2050), as well as substantial increase of the shared memory and the availability of L1/L2 caches (64KB of on-chip memory that can be configured between the shared memory (16KB/48KB) and the L1 cache (48KB/16KB), and a 768KB L2 cache) should improve the performance and simplify the programming of the parallel GPU-based pricing algorithm presented in this paper (see Remark 2). In addition, since the algorithm presented in this paper is memory bound, higher memory bandwidths (144GB/s on the Tesla C2050 as opposed to 100GB/s on the Tesla T10), should also increase the performance of the parallel methods. Also, several CUDA libraries, such as the library for sparse linear algebra and graph computations on CUDA (CUPS) [2] which contains routines for sparse matrix-vector multiplications, became available. One could consider tailoring and incorporating these routines into the current implementation, to possibly improve the efficiency of the parallel pricing methods. However, in this case, substantial restructuring of the code may be required, due to possible different sparse storage format for the matrices and vectors used in the library and in the current implementation of the algorithm.

We conclude the paper by mentioning some possible extensions of this work. Several features, such as support for non-uniform grids, could be added to the current implementation to further increase the efficiency of the methods. However, the stability of the ADI method on a non-uniform mesh needs to be studied. From a parallelization perspective, it would also be interesting to investigate other possible extensions of the current algorithm, such as those discussed in Remarks 2 and 3. Extending the current project to a multi-GPU platform should increase the performance of the GPU algorithm presented here. From a modeling perspective, it is desirable to impose stochastic volatility on the FX rate so that the market-observed FX volatility smiles are more accurately approximated. This enrichment to the current model leads to a time-dependent PDE in four state variables – the spot FX rate, domestic and foreign interest rates, and volatility. In such an application, a GPU-based pricing method is expected to deliver even larger speedups and better performance.

Acknowledgment

The authors thank the anonymous referees for their constructive comments and suggestions. The authors especially thank Dr. Asif Lakhany of Algorithmics Inc., Toronto, Canada, for suggesting the problem of pricing PRDC swaps, and Dr. Vladimir Piterbarg of Barclays Capital, London, United Kingdom, for useful discussions regarding PRDC swaps. This research was supported in part by the Natural Sciences and Engineering Research Council (NSERC) of Canada. Access to a GPU cluster was provided by the Shared Hierarchical Academic Research Computing Network (SHARCNET: www.sharcnet.ca).

[1] L. A. ABBAS-TURKI, S. VIALLE, B. LAPEYRE, AND P. MERCIER, *High dimensional pricing of exotic European contracts on a GPU cluster, and comparison to a CPU cluster*, in Proceedings of the 2nd International Workshop on Parallel and Distributed Computing in Finance, IEEE Computer Society, 2009, pp. 1–8.

[2] N. BELL AND M. GARLAND, *CUSP: Generic parallel algorithms for sparse matrix and graph computations*, 2010. http://cusp-library.googlecode.com, Version 0.1.0.

[3] D. M. DANG, *Pricing of cross-currency interest rate derivatives on Graphics Processing Units*, in Proceedings of the International Symposium on Parallel & Distributed Processing (IPDPS), IEEE Computer Society, 2010, pp. 1–8.

[4] D. M. DANG, C. CHRISTARA, AND K. JACKSON, *Pricing multi-asset American options on Graphics Processing Units using a PDE approach*, in Proceedings of the ACM/IEEE International Conference for High Performance Computing, Networking, Storage, and Analysis, IEEE Computer Society, 2010, pp. 1–8.

[5] ——, *A parallel implementation on GPUs of ADI finite difference methods for parabolic PDEs with applications in finance*, To appear in the Canadian Applied Mathematics Quarterly (CAMQ), (2011).

[6] D. M. DANG, C. C. CHRISTARA, K. JACKSON, AND A. LAKHANY, *A PDE pricing framework for cross-currency interest rate derivatives*, in Proceedings of the 10th International Conference In Computational Science (ICCS), Procedia Computer Sciences, Elsevier, 2010, pp. 1–10.

[7] M. DEMPSTER AND J. HUTTON, *Numerical valuation of cross-currency swaps and swaptions*, Cambridge University Press, 1997.

[8] J. C. HULL, *Options, Futures, and Other Derivatives*, Prentice Hall, seventh ed., 2008.

[9] K. IN'T HOUT AND B. WELFERT, *Unconditional stability of second-order ADI schemes applied to multi-dimensional diffusion equations with mixed derivative terms*, Appl. Numer. Math, 59 (2009), pp. 677–692.

[10] R. JARROW AND Y. YILDIRIM, *Pricing treasury inflation protected securities and related derivatives using an HJM model*, Journal of Financial and Quantitative Analysis, 38 (2003), pp. 409–430.

[11] D. MURAKOWSKI, W. BROUWER, AND V. NATOLI, *CUDA implementation of barrier option valuation with jump-diffusion process and Brownian bridge*, in Proceedings of the ACM/IEEE International Conference for High Performance Computing, Networking, Storage, and Analysis, IEEE Computer Society, 2010, pp. 1–4.

[12] NVIDIA, *NVIDIA Compute Unified Device Architecture: Programming Guide Version 3.2*, NVIDIA Developer Web Site, (2010). http://developer.nvidia.com/object/gpucomputing.html.

[13] V. PITERBARG, *Smiling hybrids*, Risk magazine, 19 (2005), pp. 66–70.

[14] J. SIPPEL AND S. OHKOSHI, *All power to PRDC notes*, Risk magazine, 15 (2002), pp. 1–3.

[15] Y. TIAN, Z. ZHU, F. C. KLEBANER, AND K. HAMZA, *Option pricing with the SABR model on the GPU*, in Proceedings of the ACM/IEEE International Conference for High Performance Computing, Networking, Storage, and Analysis, IEEE Computer Society, 2010, pp. 1–8.

[16] H. WINDCLIFF, P. A. . FORSYTH, AND K. R. VETZAL, *Analysis of the stability of the linear boundary condition for the Black-Scholes equation*, Journal of Computational Finance, 8 (2004), pp. 65–92.

[17] Y. ZHANG, J. COHEN, AND J. D. OWENS, *Fast tridiagonal solvers on the GPU*, in Proceedings of the 15th Symposium on Principles and Practice of Parallel Programming, Association for Computing Machinery, 2010, pp. 127–136.