A Highly Efficient Implementation on GPU Clusters of PDE-Based Pricing Methods for Path-Dependent Foreign Exchange Interest Rate Derivatives

Duy-Minh Dang[1], Christina C. Christara[2], and Kenneth R. Jackson[2]

[1] David R. Cheriton School of Computer Science,
University of Waterloo, Waterloo, ON, N2L 3G1, Canada
dm2dang@uwaterloo.ca
[2] Department of Computer Science,
University of Toronto, Toronto, ON, M5S 3G4, Canada
{ccc,krj}@cs.toronto.edu

Abstract. We present a highly efficient parallelization of the computation of the price of exotic cross-currency interest rate derivatives with path-dependent features via a Partial Differential Equation (PDE) approach. In particular, we focus on the parallel pricing on Graphics Processing Unit (GPU) clusters of long-dated foreign exchange (FX) interest rate derivatives, namely Power-Reverse Dual-Currency (PRDC) swaps with FX Target Redemption (FX-TARN) features under a three-factor model. Challenges in pricing these derivatives via a PDE approach arise from the high-dimensionality of the model PDE, as well as from the path-dependency of the FX-TARN feature. The PDE pricing framework for FX-TARN PRDC swaps is based on partitioning the pricing problem into several independent pricing sub-problems over each time period of the swap's tenor structure, with possible communication at the end of the time period. Finite difference methods on non-uniform grids are used for the spatial discretization of the PDE, and the Alternating Direction Implicit (ADI) technique is employed for the time discretization. Our implementation of the pricing procedure on a GPU cluster involves (i) efficiently solving each independent sub-problem on a GPU via a parallelization of the ADI timestepping technique, and (ii) utilizing MPI for the communication between pricing processes at the end of the time period of the swap's tenor structure. Numerical results showing the efficiency of the parallel methods are provided.

1 Introduction

In the current era of wildly fluctuating exchange rates, cross-currency interest rate derivatives, especially FX interest rate hybrid derivatives, referred to as hybrids, are of enormous practical importance. In particular, long-dated (maturities of 30 years or more) FX interest rate hybrids, such as Power-Reverse Dual-Currency (PRDC) swaps, are among the most liquid cross-currency interest rate derivatives [1]. The pricing of PRDC swaps, especially those with FX Target Redemption (TARN), is a subject of great interest in practice, especially among financial institutions. In a PRDC swap

B. Murgante et al. (Eds.): ICCSA 2013, Part V, LNCS 7975, pp. 107–126, 2013.

with a TARN feature, the sum of all FX-linked PRDC coupon amounts paid to date is recorded, and the underlying swap is terminated pre-maturely on the first date of the tenor structure when the accumulated PRDC coupon amount, including the coupon amount scheduled on that date, has reached or exceeded a pre-determined target cap. Hence, this exotic feature is usually referred to as a FX-TARN.

As FX interest rate derivatives, such as PRDC swaps, are exposed to movements in both the spot FX rate and the interest rates in both currencies, multi-factor pricing models having at least three factors, namely the domestic and foreign interest rates and the spot FX rate, must be used for the valuation of such derivatives. A popular choice for pricing PRDC swaps is Monte-Carlo (MC) simulation. However, this approach has several major disadvantages, such as slow convergence for problems in low-dimensions, i.e. fewer than five dimensions, 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, MC methods usually suffer from difficulty in computing accurate hedging parameters, such as delta and gamma, especially when dealing with the FX-TARN feature [2]. On the other hand, the pricing of these derivatives via the PDE approach is not only mathematically challenging but also very computationally intensive, due to (i) the "curse of dimensionality" associated with high-dimensional PDEs, and (ii) the complexities in handling path-dependent exotic features.

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 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 MC simulations applied to option pricing (e.g. [3, 4, 5]). The literature on utilizing GPUs in pricing financial derivatives via a PDE approach is rather sparse, with scattered work, such as [6, 7, 8, 9, 10]. The literature on GPU-based PDE methods for pricing cross-currency interest rate derivatives is even less developed.

In our paper [11], an efficient PDE pricing framework for pricing FX-TARN PRDC swaps is introduced in the public domain. The approach is to use an auxiliary path-dependent state variable to keep track of the accumulated PRDC coupon amount. This allows us to partition the pricing problem of these derivatives into several independent pricing sub-problems over each period of the swap's tenor structure, each of which corresponds to a discretized value of the auxiliary variable, with possible communication at the end of each time period.

In this paper, we describe a highly efficient parallelization of the PDE-based computation developed in [11] for the price of FX interest rate swaps with the FX-TARN feature. We adopt the three-factor pricing model proposed in [12]. Our implementation involves two levels of parallelism. The first is to use a cluster of GPUs together with the Compute Unified Device Architecture (CUDA) Application Programming Interface (API) to solve the afore-mentioned independent sub-problems simultaneously, each on a separate GPU. Since the main computational task associated with each sub-problem is the solution of the model three-dimensional PDE, the second level of parallelism

is exploited via a highly efficient GPU-based parallelization of the ADI timestepping technique developed in our paper [7] for the solution of the model PDE. In addition, we utilize the Message Passing Interface (MPI) [13], a widely used message passing library standard, for efficient communication between the pricing processes at the end of each time period. The results of this paper show that GPU clusters can provide a significant increase in performance over GPUs when pricing exotic cross-currency interest rate derivatives with path-dependence features. 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. In Section 2, we briefly describe PRDC swaps with FX-TARN features, then introduce a three-factor pricing model and the associated PDE. Discretization methods and a PDE-based pricing algorithm for FX-TARN PRDC swaps are discussed in Section 3. A parallelization of the pricing algorithm on GPU clusters for FX-TARN PRDC swaps is described in detail in Section 4. Numerical results are presented and discussed in Section 5. Section 6 concludes the paper and outlines possible future work.

2 Power-Reverse Dual-Currency Swaps

2.1 Introduction

Essentially, PRDC swaps are long-dated swaps (maturities of 30 years or more) which pay FX-linked coupons, i.e. PRDC coupons, referred to as the *coupon leg*, in exchange for London Interbank Offered Rate (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 swap: 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.

To be more specific, we consider the tenor structure

$$T_0 = 0 < T_1 < \cdots < T_\beta < T_{\beta+1} = T, \nu_\alpha = T_\alpha - T_{\alpha-1}, \alpha = 1, 2, \ldots, \beta + 1, \quad (2.1)$$

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 day counting one [14]. Unless otherwise stated, in this paper, the sub-scripts "d" and "f" are used to indicate domestic and foreign, respectively. Let $P_d(t, \overline{T})$ be the price at time $t \leq \overline{T}$ in domestic currency of a domestic zero-coupon discount bond with maturity $\overline{T}$, and face value one unit of domestic currency. Note that, $P_d(t, \overline{T}) \leq 1$ and $P_d(\overline{T}, \overline{T}) = 1$. For use later in the paper, define

$$T_{\alpha^+} = T_\alpha + \delta \text{ where } \delta \to 0^+, \quad T_{\alpha^-} = T_\alpha - \delta \text{ where } \delta \to 0^+, \quad (2.2)$$

i.e. T_{α^-} and T_{α^+} are instants of time just before and just after the date T_α, respectively.

Given the tenor structure (2.1), for a "vanilla" PRDC swap, at each time $\{T_\alpha\}_{\alpha=1}^{\beta}$, there is an exchange of a PRDC coupon amount for a domestic LIBOR floating-rate payment. More specifically, the funding leg pays the amount $\nu_\alpha L_d(T_{\alpha-1}, T_\alpha) N_d$ at

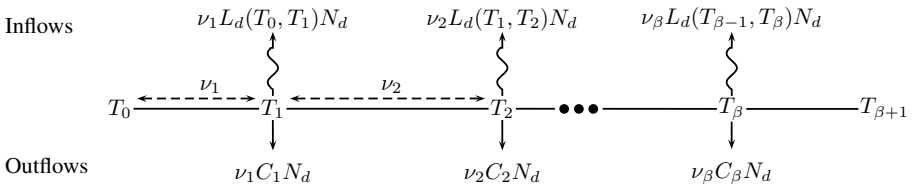

Fig. 1. Fund flows in a "vanilla" PRDC swap. Inflows and outflows are from the perspective of the PRDC coupon issuer, usually a bank.

time T_α for the period $[T_{\alpha-1}, T_\alpha]$. Here, $L_d(T_{\alpha-1}, T_\alpha)$ denotes the domestic LIBOR rate for the period $[T_{\alpha-1}, T_\alpha]$, as observed at time $T_{\alpha-1}$. This rate is simply-compounded and is defined by [14]

$$L_d(T_{\alpha-1}, T_\alpha) = \frac{1 - P_d(T_{\alpha-1}, T_\alpha)}{\nu_\alpha P_d(T_{\alpha-1}, T_\alpha)}. \tag{2.3}$$

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.

Throughout the paper, we denote by $s(t)$ the spot FX rate prevailing at time t. The PRDC coupon rate C_α, $\alpha = 1, 2, \ldots, \beta$, of the coupon amount $\nu_\alpha C_\alpha N_d$ issued at time T_α for the period $[T_\alpha, T_{\alpha+1}]$, $\alpha = 1, 2, \ldots, \beta$, has the structure

$$C_\alpha = \max\left(c_f \frac{s(T_\alpha)}{f_\alpha} - c_d, 0\right), \tag{2.4}$$

where c_d and c_f respectively are constant domestic and foreign coupon rates. The scaling factor f_α is usually set to the forward FX rate $F(0, T_\alpha)$ defined by [14]

$$F(0, T_\alpha) = \frac{P_f(0, T_\alpha)}{P_d(0, T_\alpha)} s(0), \tag{2.5}$$

which follows from no-arbitrage arguments. A diagram of fund flows in a "vanilla" PRDC swap is presented in Figure 1.[1]

By letting $h_\alpha = \dfrac{c_f}{f_\alpha}$, and $k_\alpha = \dfrac{c_d}{c_f} f_\alpha$, the PRDC coupon rate C_α can be viewed as a call option on FX rates, since, in this case, C_α reduces to

$$C_\alpha = h_\alpha \max(s(T_\alpha) - k_\alpha, 0). \tag{2.6}$$

As a result, the PRDC coupon leg in a "vanilla" PRDC swap can be viewed as a portfolio of long-dated options on the spot FX rate, i.e. long-dated FX options.

In a FX-TARN PRDC swap, the PRDC coupon amount, $\nu_\alpha C_\alpha N_d$, $\alpha = 1, 2, \ldots$, is recorded. The PRDC swap is pre-maturely terminated on the first date $T_{\alpha_e} \in \{T_\alpha\}_{\alpha=1}^{\beta}$ when the accumulated PRDC coupon amount, including the coupon amount scheduled on that date, reaches or exceeds a pre-determined target cap, hereinafter denoted by

[1] Note that in the above setting, the last period $[T_\beta, T_{\beta+1}]$ of the swap's tenor structure is redundant, since there is no exchange of fund flows at time $T_{\beta+1}$. However, to be consistent with [12], we follow the same notation used in [12].

a_c. That is, the associated underlying PRDC swap terminates immediately on the first date $T_{\alpha_e} \in \{T_\alpha\}_{\alpha=1}^{\beta}$ when $\sum_{\alpha=1}^{\alpha_e} \nu_\alpha C_\alpha N_d \geq a_c$. In this paper, we discuss the case when the early termination is determined by the equality, i.e. $\sum_{\alpha=1}^{\alpha_e} \nu_\alpha C_\alpha N_d = a_c$. Note that, in this case, the last PRDC coupon amount could possible get truncated, due to the cap a_c. A description of other variations of FX-TARN PRDC swaps, as well as the financial motivation for these derivatives can be found in [11].

We conclude this subsection by noting that, usually, there is a settlement in the form of an initial fixed-rate coupon between the issuer and the investor at time T_0 that is not included in the description above. This signed coupon is typically the value at time T_0 of the swap to the issuer, i.e. the value at time T_0 of all net fund flows in the swap, with a positive value of the fixed-rate coupon indicating a fund outflow for the issuer or a fund inflow for the investor, i.e. the issuer pays the investor. Conversely, a negative value of this coupon indicates a fund inflow for the issuer.

2.2 The Model and the Associated PDE

We consider the multi-currency model proposed in [12]. We denote by $s(t)$ the spot FX rate, and by $r_i(t), i = d, f$, the domestic and foreign short rates, respectively. Under the domestic risk-neutral measure, the dynamics of $s(t), r_d(t), r_f(t)$ can be described by [15]

$$\begin{aligned} \frac{ds(t)}{s(t)} &= (r_d(t) - r_f(t))dt + \gamma(t, s(t))dW_s(t), \\ dr_d(t) &= (\theta_d(t) - \kappa_d(t) r_d(t))dt + \sigma_d(t) dW_d(t), \\ dr_f(t) &= (\theta_f(t) - \kappa_f(t) r_f(t) - \rho_{fs}(t)\sigma_f(t)\gamma(t, s(t)))dt + \sigma_f(t) dW_f(t), \end{aligned} \tag{2.7}$$

where $W_d(t), W_f(t)$, and $W_s(t)$ are correlated Brownian motions with $dW_d(t)dW_s(t) = \rho_{ds}dt$, $dW_f(t)dW_s(t) = \rho_{fs}dt$, $dW_d(t)dW_f(t) = \rho_{df}dt$. The short rates follow the mean-reverting Hull-White model [16] with deterministic mean reversion rates and 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 local volatility function $\gamma(t, s(t))$ for the spot FX rate has the functional form [12]

$$\gamma(t, s(t)) = \xi(t) \Big(\frac{s(t)}{\ell(t)} \Big)^{\varsigma(t)-1}, \tag{2.8}$$

where $\xi(t)$ is the relative volatility function, $\varsigma(t)$ is the time-dependent constant elasticity of variance (CEV) parameter and $\ell(t)$ is a time-dependent scaling constant which is usually set to the forward FX rate $F(0, t)$, for convenience in calibration [12]. Let $u \equiv u(s, r_d, r_f, t)$ denote the domestic value function of a PRDC swap at time t, $T_{\alpha-1} \leq t < T_\alpha$, $\alpha = \beta, \ldots, 1$. Given a terminal payoff at maturity time T_α, then on $\mathbb{R}_+ \times \mathbb{R} \times \mathbb{R} \times [T_{\alpha-1}, T_\alpha)$, u satisfies the PDE [15][2]

[2] Here, we assume that u is sufficiently smooth on the domain $\mathbb{R}_+ \times \mathbb{R} \times \mathbb{R} \times [T_{\alpha-1}, T_\alpha)$.

$$\frac{\partial u}{\partial t}+\mathcal{L}u\equiv\frac{\partial u}{\partial t}+\frac{1}{2}\gamma^2(t,s(t))s^2\frac{\partial^2 u}{\partial s^2}+\frac{1}{2}\sigma_d^2(t)\frac{\partial^2 u}{\partial r_d^2}+\frac{1}{2}\sigma_f^2(t)\frac{\partial^2 u}{\partial r_f^2}$$
$$+\rho_{ds}\sigma_d(t)\gamma(t,s(t))s\frac{\partial^2 u}{\partial s\partial r_d}+\rho_{fs}\sigma_f(t)\gamma(t,s(t))s\frac{\partial^2 u}{\partial s\partial r_f}+\rho_{df}\sigma_d(t)\sigma_f(t)\frac{\partial^2 u}{\partial r_d\partial r_f}$$
$$+(r_d-r_f)s\frac{\partial u}{\partial s}+\Big(\theta_d(t)-\kappa_d(t)r_d\Big)\frac{\partial u}{\partial r_d}+\Big(\theta_f(t)-\kappa_f(t)r_f-\rho_{fs}\sigma_f(t)\gamma(t,s(t))\Big)\frac{\partial u}{\partial r_f}$$
$$-r_d u=0. \tag{2.9}$$

Since we solve the PDE backward in time, the change of variable $\tau = T_\alpha - t$ is used. Under this change of variable, the PDE (2.9) becomes

$$\frac{\partial u}{\partial \tau}=\mathcal{L}u \tag{2.10}$$

and is solved forward in τ. The pricing of cross-currency interest rate derivatives in general, and PRDC swaps in particular, is defined in an unbounded domain

$$\{(s,r_d,r_f,\tau)|s\geq 0,-\infty<r_d<\infty,-\infty<r_f<\infty,\tau\in[0,T]\}, \tag{2.11}$$

where $T = T_\alpha - T_{\alpha-1}$. Here, $-\infty < r_d < \infty$ and $-\infty < r_f < \infty$, since the Hull-White model can yield any positive or negative value for the interest rate. To solve the PDE (2.10) numerically by FD methods, we truncate the unbounded domain into a finite-sized computational one

$$\{(s,r_d,r_f,\tau)\in[0,s_\infty]\times[-r_{d,\infty},r_{d,\infty}]\times[-r_{f,\infty},r_{f,\infty}]\times[0,T]\}\equiv\Omega\times[0,T], \tag{2.12}$$

where s_∞, $r_{d,\infty}$ and $r_{f,\infty}$ are sufficiently large [17].

Since payoffs and fund flows are deal-specific, we defer specifying the terminal conditions until Section 3. The difficulty with choosing boundary conditions is that, for an arbitrary payoff, they are not known. A detailed analysis of the boundary conditions is not the focus of this paper; we leave it as a topic for future research. For this paper, 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 of the finite-sized computational domain. Thus, the value on the boundary is simply the discounted payoff for the current values of the state variables [11]

3 Numerical Methods

In this section, we briefly discuss a PDE-based pricing method for FX-TARN PRDC swaps. The reader is referred to our paper [11] for more details.

3.1 Discretization of the Model PDE

Let the number of sub-intervals be $n+1$, $p+1$, $q+1$, and l in the s-, r_d-, r_f-, and τ-directions, respectively. We use a fixed, but not necessarily uniform, spatial grid together with dynamically chosen timestep sizes. For the discretization of the space variables in the differential operator $\mathcal{L}$ of (2.10), we employ FD *central* schemes on

non-uniform grids in the interior of the rectangular domain Ω. More specifically, the first and second partial derivatives of the space variables in (2.10) are approximated by the standard three-point stencils *central* FD schemes, while the cross-derivatives in (2.10) are approximated by a nine-point (3×3) FD stencil.[3]

For the time discretization of the PDE (2.10), we employ the ADI timestepping technique based on the Hundsdorfer and Verwer (HV) splitting approach [18], henceforth referred to as the *HV scheme*. Note that the study of the HV scheme for mixed derivatives high-dimensional PDEs is found in [19]. Let $\mathbf{u}^m$ denote the vector of values of the unknown prices at time τ_m 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 of size $npq \times npq$ arising from the FD discretization of the differential operator $\mathcal{L}$ at τ_m.

Following the HV approach, we decompose the matrix $\mathbf{A}^m$ into four sub-matrices: $\mathbf{A}^m = \mathbf{A}_0^m + \mathbf{A}_1^m + \mathbf{A}_2^m + \mathbf{A}_3^m$. The matrix $\mathbf{A}_0^m$ is the part of $\mathbf{A}^m$ that comes from the FD discretization of the cross-derivative terms in (2.10), 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 $\mathcal{L}u$ is distributed evenly over $\mathbf{A}_1^m$, $\mathbf{A}_2^m$ and $\mathbf{A}_3^m$. Starting from $\mathbf{u}^{m-1}$, the HV scheme generates an approximation $\mathbf{u}^m$ to the exact solution u^m, $m = 1, \ldots, l$, by[4]

$$
\begin{cases}
\mathbf{v}_0 = \mathbf{u}^{m-1} + \Delta\tau_m(\mathbf{A}^{m-1}\mathbf{u}^{m-1} + \mathbf{g}^{m-1}), & \text{(3.1a)} \\
(\mathbf{I} - \theta\Delta\tau_m\mathbf{A}_i^m)\mathbf{v}_i = \mathbf{v}_{i-1} - \theta\Delta\tau_m\mathbf{A}_i^{m-1}\mathbf{u}^{m-1} \\
\qquad\qquad + \theta\Delta\tau_m(\mathbf{g}_i^m - \mathbf{g}_i^{m-1}), \quad i = 1, 2, 3, & \text{(3.1b)} \\
\widetilde{\mathbf{v}}_0 = \mathbf{v}_0 + \frac{1}{2}\Delta\tau_m(\mathbf{A}^m\mathbf{v}_3 - \mathbf{A}^{m-1}\mathbf{u}^{m-1}) \\
\qquad\qquad + \frac{1}{2}\Delta\tau_m(\mathbf{g}^m - \mathbf{g}^{m-1}), & \text{(3.1c)} \\
(\mathbf{I} - \theta\Delta\tau_m\mathbf{A}_i^m)\widetilde{\mathbf{v}}_i = \widetilde{\mathbf{v}}_{i-1} - \theta\Delta\tau_m\mathbf{A}_i^m\mathbf{v}_3, \quad i = 1, 2, 3, & \text{(3.1d)} \\
\mathbf{u}^m = \widetilde{\mathbf{v}}_3. & \text{(3.1e)}
\end{cases}
$$

In (3.1), 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.

When solving the PDE (2.10) backward in time over each time period of the swap's tenor structure, for damping purposes, we first apply the HV scheme with $\theta = 1$ for the first few (usually two) initial timesteps, and then switch to $\theta = \frac{1}{2} + \frac{1}{6}\sqrt{3}$ for the remaining timesteps.

3.2 Timestep Size Selector

We use a simple, but effective, timestep size selector, where, given the current stepsize $\Delta\tau_m$, $m \geq 1$, the new stepsize $\Delta\tau_{m+1}$ is given by [11]

$$
\begin{cases}
\Delta\tau_{m+1} = \left(\min_{1 \leq \iota \leq npq} \left[\dfrac{\texttt{dnorm}}{\frac{|\mathbf{u}_\iota^m - \mathbf{u}_\iota^{m-1}|}{\max(N, |\mathbf{u}_\iota^m|, |\mathbf{u}_\iota^{m-1}|)}} \right] \right) \Delta\tau_m, \\
\Delta\tau_{m+1} = \min\left\{ \Delta\tau_{m+1}, T - \tau_m \right\}.
\end{cases} \tag{3.2}
$$

[3] On uniform grids, the nine-point FD stencil reduces to a four-point one.

[4] This is the scheme (1.4) in [19] with $\mu = \frac{1}{2}$.

Here, dnorm is a user-defined target relative change, and the scale N is chosen so that the method does not take an excessively small stepsize where the value of the option is small. Normally, for option values in dollars, $N = 1$ is used. We use $N = 1$ for PRDC swap pricing too. In all our experiments, we used $\Delta\tau_1 = 10^{-2}$ and dnorm $= 0.3$ on the coarsest grids. The value of dnorm is reduced by two at each refinement, while $\Delta\tau_1$ is reduced by four.

3.3 A PDE Pricing Algorithm

Denote by $a(t)$, $0 \leq a(t) < a_c$, the auxiliary path-dependent state variable which represents the accumulated PRDC coupon amount. The value of a FX-TARN PRDC swap depends on four stochastic state variables, namely $s(t)$, $r_d(t)$, $r_f(t)$ and the path-dependent variable $a(t)$. It is important to note that, since $a(t)$ changes only on the dates $\{T_\alpha\}_{\alpha=1}^{\beta}$, the pricing PDE does not depend on $a(t)$ (see (2.9)). For presentation purposes, we further adopt the following notation: $a_{\alpha^+} \equiv a(T_{\alpha^+}), a_{\alpha^-} \equiv a(T_{\alpha^-})$.

Pricing FX-TARN PRDC swaps via a PDE approach is highly challenging due to the path-dependency of the TARN feature and the backward nature of a PDE approach. We observe that, over each period $[T_{(\alpha-1)^+}, T_{\alpha^-}]$ of the swap's tenor structure, the backward procedure, which computes the solution backward in time from T_{α^-} to $T_{(\alpha-1)^+}$, needs to be invoked only if the swap is still alive at time $T_{(\alpha-1)^+}$, i.e. if $a_{(\alpha-1)^+}$ satisfies $0 \leq a_{(\alpha-1)^+} < a_c$. Since we progress backward in time and the variable $a(t)$ is path-dependent, we do not know the exact value of $a_{(\alpha-1)^+}$. However, since $0 \leq a_{(\alpha-1)^+} < a_c$, we can discretize the variable a, as we do with other spatial variables. To this end, we partition the interval $[0, a_c]$ into $w + 1$ sub-intervals having non-uniform gridpoints,

$$0 = a_0 < a_1 < \ldots < a_w < a_{w+1} = a_c, \tag{3.3}$$

where the gridpoints are denser toward a_c. The PDE pricing framework for a FX-TARN PRDC swap involves

(a) across each date $\{T_\alpha\}_{\alpha=\beta}^{1}$ and for each discretized value a_y of the variable a, applying certain updating rules to (i) take into account the fund flows scheduled on that date; (ii) reflect changes in the accumulated PRDC coupon amount, and the possibility of early termination; and (iii) obtain terminal conditions for the solution of the PDE from time T_{α^-} to $T_{(\alpha-1)^+}$.
(b) over each period $[T_{(\alpha-1)^+}, T_{\alpha^-}]$, $\alpha = \beta, \ldots, 1$, of the swap's tenor structure, for each discretized value a_y of the variable a, solving the model PDE (2.9) backward in time from T_{α^-} to $T_{(\alpha-1)^+}$, with the corresponding terminal condition obtained from the above step.

Remark 1. To improve the efficiency of the numerical methods, for the solution of the model PDE, we use non-uniform grids. We denote by Δ_α^y, $y = 0, \ldots, w$, the non-uniform three-dimensional grids used for the solution of the PDE corresponding to a_y over the time period $[T_{(\alpha-1)^+}, T_{\alpha^-}]$ in (b) above. The non-uniform grids Δ_α^y are more refined around $r_d(0)$ and $r_f(0)$ in the r_d- and the r_f-directions, respectively. In the s-direction, the grids Δ_α^y, are more refined around the strike k_α and around the

value of s at which the early termination occurs, hereinafter denoted by b_α^y. Note that, within $[T_{(\alpha-1)^+}, T_{\alpha^-}]$, k_α is the same for all sub-problems, but b_α^y, $y = 0, \ldots, w$, are not. Both k_α and b_α^y, $y = 0, \ldots, w$, change from one time period to the next. In our implementation, we apply linear interpolation along the s- and a-directions to switch between spatial grids (see Lines 5 and 10 of Algorithm 3.1).

Let $u_\alpha(t; a)$ represent the value at time t of a FX-TARN PRDC swap that has (i) $\{T_{\alpha+1}, \ldots, T_\beta\}$ as pre-mature termination opportunities, i.e. the swap is still alive at time T_α; and (ii) the total accumulated PRDC coupon amount, including the coupon amount scheduled on T_α, is equal to $a < a_c$. In particular, the quantity $u_0(T_0; 0)$ is the value of the FX-TARN PRDC swap we are interested in at time T_0. Also let $u_\alpha^{y,\widetilde{\alpha}}(t; a)$, $y = 0, \ldots, w$, $\widetilde{\alpha} = \beta, \ldots, 1$, represent an approximation to $u_\alpha(t; a)$ at gridpoints of the computational grid $\Delta_{\widetilde{\alpha}}^y$. In general, the indices $(y, \widetilde{\alpha})$ denote the associated computational grid $\Delta_{\widetilde{\alpha}}^y$, $y = 0, \ldots, w$, $\widetilde{\alpha} = \beta, \ldots, 1$. A backward pricing algorithm for FX-TARN PRDC swaps is presented in Algorithm 3.1.

4 Efficient Implementation on Clusters of GPUs

4.1 GPU Device Architecture

A GPU is a hierarchically arranged multiprocessor unit, in which several scalar processors are grouped into a smaller number of streaming multiprocessors (SMs). Each SM has shared memory accessed by all its scalar processors. In addition, the GPU has global (device) memory (slower than shared memory) accessed by all scalar processors on the chip, as well as a small amount of cache for storing constants. According to the programming model of CUDA, which we adopt, the host (CPU/master) uploads the intensive work to the GPU as a single program, called the *kernel*. Multiple copies of the kernel, referred to as *threads*, are then distributed to the available processors, where they are executed in parallel. Within the CUDA framework, threads are grouped into *threadblocks*, which are in turn arranged on a *grid*. Threads in a threadblock run on at most one multiprocessor, and can communicate with each other efficiently via the shared memory, as well as synchronize their executions. For a more detailed description of the GPU, interested readers are referred to [20].

4.2 GPU Cluster

All of the experiments in this paper were carried out on a GPU cluster with the following specifications:
- The cluster has 22 (server) nodes, each of which consists of two quad-core Intel "Harpertown" host systems with Intel Xeon E5430 CPUs running at 2.66GHz, with a total of 8GB of memory shared between the two quad-core Xeon processors. Thus, there are 44 hosts available. All the nodes are interconnected via 4x DDR Infiniband (16 Gigabytes/s).
- The GPU portion of the cluster is composed of 11 NVIDIA S1070 GPU servers, each of which contains two pairs of Tesla 10-series (T10) GPUs. Thus, there are 44 GPUs available. Each pair of the T10 GPUs is attached to a node via a PCI Express 2.0x16

Algorithm 3.1 Backward algorithm for computing FX-TARN PRDC swaps.

construct Δ_β^y; set $u_\beta(T_{\beta^+}; a_y) = 0$, $y = 0, \ldots, w$;
for $\alpha = \beta, \ldots, 1$ **do**
  for each a_y, $y = 0, \ldots, w$, **do**
    set
$$\bar{a}_y = a_y + \min(a_c - a_y, \nu_\alpha C_\alpha N_d); \tag{3.4}$$
    set
$$u_{\alpha-1}^{y,\alpha}(T_{\alpha^+}; \bar{a}_y) = \begin{cases} 0 & \text{if } \bar{a}_y \geq a_c, \\ \frac{\bar{a}_y - a_{\bar{y}}}{a_{\bar{y}+1} - a_{\bar{y}}} u_\alpha^{y,\alpha}(T_{\alpha^+}; a_{\bar{y}+1}) + \frac{a_{\bar{y}+1} - \bar{a}_y}{a_{\bar{y}+1} - a_{\bar{y}}} u_\alpha^{y,\alpha}(T_{\alpha^+}; a_{\bar{y}}) & \\ & \text{if } a_{\bar{y}} \leq \bar{a}_y \leq a_{\bar{y}+1}, \bar{y} \in \{0, \ldots, w\}, \end{cases} \tag{3.5}$$
    where $u_\alpha^{y,\alpha}(T_{\alpha^+}; a_{\bar{y}})$ and $u_\alpha^{y,\alpha}(T_{\alpha^+}; a_{\bar{y}+1})$ are obtained by linear interpolation along the s-direction on $u_\alpha^{\bar{y},\alpha}(T_{\alpha^+}; a_{\bar{y}})$ and $u_\alpha^{\bar{y}+1,\alpha}(T_{\alpha^+}; a_{\bar{y}+1})$, respectively;
    set
$$\hat{u}_{\alpha-1}^{y,\alpha}(T_{\alpha^-}; a_y) = u_{\alpha-1}^{y,\alpha}(T_{\alpha^+}; \bar{a}_y) - \min(a_c - a_y, \nu_\alpha C_\alpha N_d); \tag{3.6}$$
    solve the PDE (2.9) with the terminal condition (3.6) from T_{α^-} to $T_{(\alpha-1)^+}$ using the ADI scheme (3.1) for each time τ_m, $m = 1, \ldots, l$, with the timestep size $\Delta\tau_m$ selected by (3.2), to obtain $\hat{u}_{\alpha-1}^{y,\alpha}(T_{(\alpha-1)^+}; a_y)$;
    if $\alpha \geq 2$ **then**
      construct $\Delta_{\alpha-1}^y$
      linearly interpolate $\hat{u}_{\alpha-1}^{y,\alpha}(T_{(\alpha-1)^+}; a_y)$ along the s-direction to obtain $\hat{\hat{u}}_{\alpha-1}^{y,\alpha-1}(T_{(\alpha-1)^+}; a_y)$;
      set
$$u_{\alpha-1}^{y,\alpha-1} T_{(\alpha-1)^+}; a_y) = \hat{\hat{u}}_{\alpha-1}^{y,\alpha-1}(T_{(\alpha-1)^+}; a_y) + (1 - P_d(T_\alpha))N_d; \tag{3.7}$$
    else
      set
$$u_{\alpha-1}^{y,\alpha} T_{(\alpha-1)^+}; a_y) = \hat{u}_\alpha^{y,\alpha-1}(T_{(\alpha-1)^+}; a_y) + (1 - P_d(T_\alpha))N_d; \tag{3.8}$$
    end if
  end for
end for
set $u_0(T_0; 0) = u_0(T_{0^+}; 0)$;

link. As such, there is a T10 GPU per quad-core Xeon processor, and thus each host has a GPU associated with it, and vice-versa.
Each NVIDIA Tesla T10 GPU consists of 4GB of global memory, 30 independent SMs, each containing 8 processors running at 1.44GHz, a total of 16384 registers, and 16 KB of shared memory per SM.

4.3 GPU-Based Parallel Pricing Framework

The key point in Algorithm 3.1 is that, over each time period $[T_{(\alpha-1)^+}, T_{\alpha^-}]$ of the tenor structure, we have multiple, entirely independent, pricing sub-problems (processes) to solve, each of which corresponds to a discrete value a_y, $y = 0, \ldots, w$. Hence, within each time period of the tenor structure, it is natural to assign each of the $w + 1$ pricing processes to a separate host/GPU. However, communication between these pricing pro-

cesses is required across each date of the tenor structure, due to the interpolation (3.5) along the a-direction.

In the following presentation, we assume that the total number of available hosts of the cluster is at least $w + 1$, each host having a respective GPU associated with it. Under the MPI framework, assume that a group of $w + 1$ parallel pricing processes has been created, with the y-th process being associated with the discrete value a_y, $y = 0, \ldots, w$. Here, the quantities y, $y = 0, \ldots, w$, are referred to as *ranks* of the processes in the group. For each instance of $\alpha, \alpha = \beta, \ldots, 1$, to proceed from T_α to $T_{\alpha-1}$, assume that the values $u_\alpha^{y,\alpha}(T_{\alpha^+}; a_y)$, $y = 0, \ldots, w$, have been computed at the previous period of the tenor structure, and are available in the yth host/GPU. Also assume that the appropriate kernels have been launched by the hosts on the respective GPUs. Then, the parallel implementation of Algorithm 3.1 for one instance of α can be described by the following stages:

Stage 1: each thread in each GPU updates its quantity $\bar{a}_y$ via (3.4), then determines the ranks of those processes from which it will require to receive data in order to apply the interpolation (3.5); each GPU appropriately collects the ranks' data from all its threads, so that each process knows collectively the ranks of those processes from which it will require to receive data to apply (3.5);
Stage 2: each host copies the ranks' data from its GPU global memory to the host memory.
Stage 3: the hosts perform communication amongst each other via MPI, so that each host receives the data needed for the interpolation (3.5) associated with the host's process.
Stage 4: each host copies the relevant data form its host memory to its GPU global memory.
Stage 5: each thread in each GPU carries out the interpolation (3.5).
Stage 6: each thread in each GPU computes the PRDC coupons via (3.6).
Stage 7: each GPU solves its associated PDE (2.9) from T_{α^-} to $T_{(\alpha-1)^+}$ with the terminal condition obtained from Stage 6.
Stage 8: each thread in each GPU (possibly) applies linear interpolation along the s-direction as given on Line 10 of Algorithm 3.1.
Stage 9: each thread in each GPU computes the funding payments via (3.7) or (3.8).

Note that, Stage 3 involves communication among hosts using MPI, while all other stages take place in each host/GPU, in parallel with and independently from other hosts/GPUs.

We now give more details of the implementation of the above stages. For presentation purposes, we denote by $\mathbf{u}_{\alpha^+}^y$ the vector of data corresponding to a_y, $y = 0, \ldots, w$, i.e. the vector of data of the process y, available at time T_{α^+} as it results from the computations during the last time period $[T_{\alpha^+}, T_{(\alpha+1)^-}]$.

4.4 Stages 1 and 2

For each process $y, y = 0, \ldots, w$, i.e. for each host/GPU, assume that we have an array of size $w + 1$ in the host memory, referred to as the array $RECV\_FROM$. The $\bar{y}$th entry of the array $RECV\_FROM$ corresponds to the discrete value $a_{\bar{y}}$, $\bar{y} = 0, \ldots, w$,

i.e. it corresponds to the process with rank $\bar{y}$ of the group. The entries of the array are of binary type, and are pre-set to a certain value, e.g. 0. The array is copied from the host memory to the device memory before the kernel of Stage 1 is launched.

We partition the computational grid of size $n \times p \times q$ into 2-D blocks of size $n_b \times p_b$. We let the kernel generate a $\texttt{ceil}\left(\frac{n}{n_b}\right) \times \texttt{ceil}\left(\frac{pq}{p_b}\right)$ grid of threadblocks, where `ceil` denotes the ceiling function. All gridpoints of a $n_b \times p_b$ 2-D block are assigned to one threadblock only, with one thread for each gridpoint.

Each thread of a threadblock of the kernel launched in this stage computes the quantity $\bar{a}_y$ associated with it via (3.4). If the quantity $\bar{a}_y$ satisfies $a_{\bar{y}} \leq \bar{a}_y \leq a_{\bar{y}+1}$ for some $\bar{y} \in \{y, \ldots, w\}$, the thread then changes the pre-set values of the $\bar{y}$ and $(\bar{y}+1)$st entries in the array $RECV\_FROM$ to 1. This procedure essentially marks the ranks of the processes from which some data are required by process y. Note that no data loadings from the global memory are required for this procedure.

The approach adopted here suggests a $(w+1-y)$-iteration loop in the kernel. During each iteration, each threadblock works with a pair of $a_{\bar{y}}$ and $a_{\bar{y}+1}$. Note that, although it may happen that multiple threads try to write to the same memory location of an entry of the array at the same time, it is guaranteed that one of the writes will succeed. Although we do not know which one, it does not matter for our purposes. Consequently, this approach suffices and works well.

After the kernel of Stage 1 has ended, Stage 2 takes place, in which the array $RECV\_FROM$ is copied back to the host memory for use in Stage 3.

4.5 Stages 3 and 4

At this point, each host has the array $RECV\_FROM$ corresponding to its process. Next, each process is to determine the ranks of those processes which need its data.

To handle this issue, consider a fictitious $(w+1) \times (w+1)$ matrix, for which the $\tilde{y}$th row, $\tilde{y} = 0, \ldots, w$, is the array $RECV\_FROM$ of the process of rank $\tilde{y}$. We observe that the yth column of this matrix, referred to as the array $SEND\_TO$, marks the ranks of processes which need the yth process data.

To form the array $SEND\_TO$ in each host, all hosts perform collective communication via MPI, essentially a parallel matrix transposition using the function `MPI_Alltoall(···)`.

Now, each process has in its host memory the arrays $RECV\_FROM$ and $SEND\_TO$, in addition to the vector $\mathbf{u}^y_{\alpha^+}$. Thus, each process can easily perform data exchange with the appropriate processes, by looping through all the "marked" entries of the arrays $RECV\_FROM$ and $SEND\_TO$. In our implementation, we use `MPI_Send(···)` and `MPI_Recv(···)`.

At this point, process y has in its host memory all the vectors of data it needs to carry out the interpolation scheme (3.5). By the data exchange procedure described above, these vectors are stored in a buffer in increasing order with respect to their associated ranks (or discrete values of a). For presentation purposes, we assume that a total of $k-1$, $k \geq 1$, vectors of data were fetched by process y from other processes during Stage 3. We denote the sorted by index list of k vectors, including the vector $\mathbf{u}^y_{\alpha^+}$, by $\{\mathbf{u}^{y_1}_{\alpha^+}, \ldots, \mathbf{u}^{y_k}_{\alpha^+}\}$, where y_j, $j = 1, \ldots, k$, are in $\{y, \ldots, w\}$, with $y_1 = y$, and $y_1 < y_2 < \cdots < y_k$. This concludes Stage 3.

In Stage 4, these vectors are then copied from the process' host memory to the global memory of the respective GPU, before the kernel for Stage 5 is launched.

4.6 Stages 5 and 6

In Stage 5, for a GPU-based implementation of the interpolation procedure, we adopt the same partitioning approach and assignment of gridpoints to threads as in Stage 1 described earlier. Recall that, in Stage 1, each thread has already computed the quantity $\bar{a}_y$ associated with it using (3.4). The interpolation (3.5) can be achieved by a k-iteration loop in the kernel. During the jth iteration of the k-iteration loop in the kernel, each thread in a threadblock performs linear interpolations, first along the s-direction, then along the a-direction, using the corresponding values in $\mathbf{u}_{\alpha^+}^{y_j}$ and $\mathbf{u}_{\alpha^+}^{y_{j+1}}$. Note that full memory coalescence is achieved for the data loading of this stage [21].

In Stage 6, using the same partitioning, each thread then computes the PRDC coupons via (3.6), independently from the others.

4.7 Stage 7

We now discuss a GPU-based parallel algorithm for the solution of the model PDE problem. The parallelism in a GPU for this stage is based on an efficient parallelization of the computation of each timestep of the ADI scheme (3.1a)–(3.1d) developed in our paper [7]. Below, we summarize our implementation. For details and discussions of related issues, such as memory coalescing and possible improvements, of our implementation, we refer the reader to [7].

4.7.1 ADI timestepping on GPUs

The HV scheme (3.1a)–(3.1d) can be divided into two phases. The first phase consists of a forward Euler step (predictor step (3.1a)), followed by three implicit, but unidirectional, corrector steps (3.1b), the purpose of which is to stabilize the predictor step. The second phase (i.e. (3.1c)-(3.1d)) restores second-order convergence of the discretization method if the model PDE contains mixed derivatives. Step (3.1e) is 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 only summarize our GPU parallelization of the first phase. For presentation purposes, let

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

and notice that $\mathbf{v}_0 = \mathbf{u}^{m-1} + \sum_{i=0}^{3} \mathbf{w}_i + \Delta\tau_m \mathbf{g}^m$. It is worth noting that the vectors $\mathbf{w}_i, \mathbf{v}_i$, $i = 0,1,2,3$, and $\widehat{\mathbf{v}}_i$, $i = 1,2,3$, depend on τ, but, to simplify the notation, we do not indicate the superscript for the timestep index. Our CUDA implementation of the first phase consists of the following steps:

1. 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 $\mathbf{w}_i$, $i = 0, 1, 2, 3$, and $\mathbf{v}_0$.
2. Step a.2: Set $\widehat{\mathbf{v}}_1 = \mathbf{v}_0 - \theta \mathbf{w}_1$ and solve $\widehat{\mathbf{A}}_1^m \mathbf{v}_1 = \widehat{\mathbf{v}}_1$;
3. Step a.3: Set $\widehat{\mathbf{v}}_2 = \mathbf{v}_1 - \theta \mathbf{w}_2$ and solve $\widehat{\mathbf{A}}_2^m \mathbf{v}_2 = \widehat{\mathbf{v}}_2$;
4. Step a.4: Set $\widehat{\mathbf{v}}_3 = \mathbf{v}_2 - \theta \mathbf{w}_3$ and solve $\widehat{\mathbf{A}}_3^m \mathbf{v}_3 = \widehat{\mathbf{v}}_3$;

First phase - Step a.1

We partition the computational grid of size $n \times p \times q$ into three-dimensional (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. 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 threadblock only, with one thread for each stack of q gridpoints. Note that, since each 3-D block has a total of q $n_b \times p_b$ tiles and each threadblock is of size $n_b \times p_b$, the approach that we use here suggests a q-iteration loop in the kernel. During each iteration of this loop, each thread of a threadblock carries out all the computations/work associated with one gridpoint, and each threadblock processes one $n_b \times p_b$ tile.

Regarding the construction of the matrices $\mathbf{A}_i^m$, $i = 0, 1, 2, 3$, and $\widehat{\mathbf{A}}_i^m$, $i = 1, 2, 3$, note that each of these matrices has a total of npq rows, with each row corresponding to a gridpoint of the computational domain. Our approach is to assign each of the threads to assemble q rows of each of the matrices (a total of three entries per row of each matrix, since all matrices are tridiagonal). More specifically, during each iteration of the q-iteration loop in the kernel, each group of $n_b p_b$ rows corresponding to a tile is assembled in parallel by a $n_b \times p_b$ threadblock, with one thread for each row. That is, a total of np consecutive rows are constructed in parallel by the threadblocks during each iteration.

Regarding the parallel computation of the vectors $\mathbf{w}_i$, $i = 0, 1, 2, 3$, it is important to emphasize that, to calculate the values corresponding to gridpoints of the kth tile (i.e. the tile on the kth s-r_d plane), the data of the two adjacent tiles in the r_f-direction (i.e. the $(k-1)$st and the $(k+1)$st tiles) are needed as well. Since 16KB of shared memory available per multiprocessor are not sufficient to store many data tiles, each threadblock works with three data tiles of size $n_b \times p_b$ at a time and proceeds in the r_f-direction. As a result, we utilize a three-plane loading strategy. More specifically, during the kth iteration of the q-iteration loop in the kernel, assuming the data corresponding to the kth and $(k-1)$st tiles in the shared memory from the previous iteration, each threadblock

1. loads from the global memory into its shared memory the old data (vector $\mathbf{u}^{m-1}$) corresponding to the $(k+1)$st tile,
2. computes and stores new values (vectors $\mathbf{w}_i$, $i = 0, 1, 2, 3$ and $\mathbf{v}_0$) for the kth tile using data of the $(k-1)$st, kth and $(k+1)$st tiles,
3. copies the newly computed data of the kth tile (vectors $\mathbf{w}_i$, $i = 1, 2, 3$ and $\mathbf{v}_0$) from the shared memory to the global memory, and frees the shared memory locations taken by the data of the $(k-1)$st tile, so that they can be used in the next iteration.

Note that the data loading approach for Step a.1 is not fully coalesced, although it is highly effective. (We believe it is impossible to attain full memory coalescing for the data-loading part of this phase.)

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

The data partitioning for each of Steps a.2, a.3 and a.4 is different from that for Step a.1 and is motivated by the block structure of the tridiagonal matrices $\widehat{\mathbf{A}}_i^m$, $i = 1, 2, 3$, respectively. For example, $\widehat{\mathbf{A}}_1^m$ has pq diagonal blocks, each block being $n \times n$ tridiagonal, thus the solution of $\widehat{\mathbf{A}}_1^m \mathbf{v}_1 = \widehat{\mathbf{v}}_1$, i.e. Step a.2, is computed by first partitioning $\widehat{\mathbf{A}}_1^m$ and $\widehat{\mathbf{v}}_1$ into pq independent $n \times n$ tridiagonal systems, and then assigning each tridiagonal system to one of the pq threads generated, i.e. each thread is assigned n gridpoints along the s-direction.

Regarding the memory coalescing for Steps a.2, a.3 and a.4, note that, in the current implementation, the data between Steps a.1, a.2, a.3 and a.4 are ordered in the s-, then the r_d-, then the r_f-direction. As a result, the data partitionings for the tridiagonal solves in the r_d- and r_f-direction, i.e. for solving $\widehat{\mathbf{A}}_i^m \mathbf{v}_i = \widehat{\mathbf{v}}_i$, $i = 2, 3$, allow full memory coalescence, while the data partitioning for solving $\widehat{\mathbf{A}}_1^m \mathbf{v}_1 = \widehat{\mathbf{v}}_1$ does not.

4.7.2 Timestep Selector on GPUs

As for the timestep selector (3.2), the key part in implementing it on the GPU involves finding the minimum element of an array of real numbers. In this regard, we adapt the parallel reduction technique discussed in [22]. The idea is to partition the array into multiple sub-arrays of size s_t, each of which is assigned to a 1-D threadblock of the same size. During the first kernel launch, each threadblock carries out the reduction operation via a tree-based approach to find the minimum of the corresponding sub-array and writes the intermediate result to a location in an array in the global memory. This array of intermediate minimum elements is then processed in the same manner by passing it on to a kernel again. This process is repeated until the array of partial minimums can be handled by a kernel launch with only one threadblock of size s_t, after which the minimum element of the initial array is found. More details about the implementation of the timestep selector can be found in our paper [8].

4.8 Stages 8 and 9

The GPU-based implementation for these stages is straightforward, since each thread of a threadblock can work independently from the others, i.e. neither communication between threads nor between processes is required. We use the same partitioning approach and assignment of gridpoints to threads employed in Stage 1. This approach allows for full memory coalescence of the loading of data from the global memory.

5 Numerical Results

As parameters to the model, we consider the same interest rates, correlation parameters, and the local volatility function as given in [12]. 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 $\sigma_d(t) = 0.7\%$, $\kappa_d(t) = 0.0\%$, $\sigma_f(t) = 1.2\%$, $\kappa_f(t) = 5.0\%$, $\rho_{df} = 25\%$, $\rho_{ds} = -15\%$, $\rho_{fs} = -15\%$. The initial spot FX rate is set to $s(0) = 105.00$, and

Table 1. The parameters $\xi(t)$ and $\varsigma(t)$ for the local volatility function (2.8). (Table C in [12].)

	period (years)									
	(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%

the initial domestic and foreign short rate are 0.02 (2%) and 0.05 (5%), respectively, which follows from the respective interest rate curve. The parameters $\xi(t)$ and $\varsigma(t)$ for the local volatility function are assumed to be piecewise constant and given in Table 1. Note that the forward FX rate $F(0,t)$ defined by (2.5) and $\theta_i(t)$, $i = d, f$, in (2.7), and the domestic LIBOR rate (2.3) are fully determined by the above information [14].

We consider the tenor structure (2.1) that has the following properties: (i) $\nu_\alpha = 1$ (year), $\alpha = 1, \ldots, \beta + 1$ and (ii) $\beta = 29$ (years). Features of the PRDC swap are: the domestic and foreign coupons are $c_d = 2.25\%, c_f = 4.50\%$ and $c_d = 8.1, c_f = 9.00\%$, with the cap a_c being set to 50% and 10%, respectively, of the notional.

The truncated computational domain Ω is defined by setting $s_\infty = 5s(0) = 525.0$, $r_{d,\infty} = 10r_d(0) = 0.2$, and $r_{f,\infty} = 10r_f(0) = 0.5$. The grid sizes and the number of timesteps reported in the tables in this section are for each time period of the Table 1. Note that, since the timestep size selector (3.2) is used, the number of timesteps reported is the average number of timesteps for all sub-problems over all time periods of the swap's tenor structure.

We report the quantity "value", which is the value of the financial instrument. In pricing PRDC swaps, this quantity is expressed as a percentage of the notional N_d. Since in our case, an accurate reference solution is not available, to provide an estimate of the convergence rate of the algorithm, we also compute the quantity "$\log_\eta$ ratio" which provides an estimate of the convergence rate of the algorithm by measuring the differences in prices on successively finer grids, referred to as "change". More specifically, this quantity is defined by

$$\log_\eta \text{ ratio} = \log_\eta \left(\frac{u_{approx}(\Delta x) - u_{approx}(\frac{\Delta x}{\eta})}{u_{approx}(\frac{\Delta x}{\eta}) - u_{approx}(\frac{\Delta x}{\eta^2})} \right),$$

where $u_{approx}(\Delta x)$ is the approximate solution computed with discretization stepsize Δx. For second-order methods, such as those considered in this paper, the quantity $\log_\eta$ ratio is expected to be about 2.

5.1 Convergence of Computed Prices

In this subsection, we demonstrate the correctness of our implementation. In Table 2, we present pricing results for FX-TARN PRDC swaps for two different combinations of c_d, c_f and a_c. In both cases, the number of sub-intervals in the a-direction is 30, i.e. $w = 29$ in (3.3). We note, for both cases, the computed prices exhibit second-order convergence, as expected from the ADI timestepping methods and the interpolation scheme.

Table 2. Values of the FX-TARN PRDC swap. The total of GPUs used is $w + 1 = 30$.

				$c_d = 8.1, c_f = 9.00\%, a_c = 10\%$			$c_d = 2.25\%, c_f = 4.50\%, a_c = 50\%$		
l (τ)	$n+1$ (s)	$p+1$ (r_d)	$q+1$ (r_f)	value (%)	change	$\log_2$ ratio	value (%)	change	$\log_2$ ratio
6	30	15	15	18.521			-4.487		
12	60	30	30	18.609	8.8e-04		-4.409	7.8e-04	
23	120	60	60	18.631	2.2e-04	1.9	-4.389	2.0e-04	1.9
47	240	120	120	18.637	5.9e-05	1.9	-4.384	5.4e-05	1.9

The central question, of course, is whether the approximations of prices of FX-TARN PRDC swaps computed by the PDE method converge to the exact prices. To verify this, we compare our PDE-computed prices with prices obtained using MC simulations. More specifically, using MC simulations, with 10^6 simulation paths for the spot FX rate, the timestep size being $1/512$, and using antithetic variates as the variance reduction technique, the benchmark prices for the FX-TARN PRDC swaps are 18.638% (std. dev. = 0.021), and -4.383% (std. dev. = 0.020), respectively for the case $c_d = 8.1, c_f = 9.00\%$ and $c_d = 2.25\%, c_f = 4.50\%$[5]. The 95% confidence intervals for the two cases are $[18.635, 18.641]$ and $[-4.386, -4.379]$, respectively, which contain our PDE-computed prices.

For the case $c_d = 2.25\%, c_f = 4.50\%$, the investor should pay a net coupon of about 4.384% of the notional to the issuer. (Note the negative values in this case.) However, for the case $c_d = 8.1, c_f = 9.00\%$, the issuer should pay the investor a net coupon of about 18.631% of the notional.

5.2 Performance Results

For FX - TARN PRDC swaps, due to the high computational requirements of the pricing algorithm, which make sequentially CPU-based computation practically infeasible, we do not develop CPU-based numerical methods in this case. Instead, we focus on numerical methods on a GPU cluster and on a single GPU. In this section, we provide details of the GPU versus GPU cluster performance comparison in pricing FX-TARN PRDC swaps.

Additional statistics collected in this subsection include the following. The quantities "GPU time" and "MPI-GPU time" respectively denote the total computation times, in seconds (s.), on a single GPU and on the GPU cluster with specifications as in Subsection 4.2 using MPI. The quantity "MPI-GPU speed up" is defined as the ratio of the "GPU time" over the respective "MPI-GPU time". The quantity "MPI-GPU efficiency" is defined as

$$\text{MPI-GPU efficiency} = \frac{1}{w+1}\frac{\text{GPU time}}{\text{MPI-GPU time}},$$

which represents the standard (fixed) efficiency of the parallel algorithm using $w + 1$ GPUs of the cluster.

[5] Our sequential code written in MATLAB for MC simulations took about 2 days to finish.

Table 3 presents some selected timing results for FX-TARN PRDC swaps for the case $c_d = 2.25\%, c_f = 4.50\%$ and $a_c = 50\%$. The timing results for the other case are approximately the same, and hence omitted. Note that the times in the brackets are the total times required for data exchange between processes using MPI functions.

It is evident that the MPI-GPU implementation on the cluster are significantly more efficient than the single-GPU implementation, with the asymptotic speedups being about 25 when using 30 GPUs (15 nodes) of the cluster. Note that, our single-GPU implementation typically attains a speed up of about 30-31 times over a CPU implementation for the largest grid considered here [6, 7]. This means that a sequentially CPU-based solver for the FX-TARN PRDC swap would take approximately 170000 (s.) ($\approx 5421.1 \times 32$), or about 2 days to finish. In practical situations, such time requirements are prohibitive.

It is important to emphasize that the GPU-MPI efficiency increases with finer grid sizes (Table 3, from 60% to 87%). This is to be expected, since a fixed number of GPUs, i.e. 30 GPUs, is used for all the experiments, whereas the problem size is increasing, allowing the GPUs to be used more efficiently.

Table 3. Timing results for the FX-TARN PRDC swaps for the case $c_d = 2.25\%, c_f = 4.50\%$ and $a_c = 50\%$. The times in the brackets are those required for data exchange between processes using MPI functions.

l	n	p	q	GPU	MPI-GPU		
					time	speed-	effi-
(τ)	(s)	(r_d)	(r_f)	(s.)	(s.)	up	ciency
12	60	30	30	114.5	6.1 (0.3)	19.3	60%
23	120	60	60	520.7	21.3 (1.8)	24.1	81%
47	240	120	120	5421.1	206.8 (8.2)	26.3	87%

6 Conclusions and Future Work

This paper presents a parallelization on clusters of GPUs of the PDE-based computation of the price of FX interest rate swaps with the FX-TARN feature under a three-factor model. Our PDE approach is to partition the pricing problem into several independent pricing sub-problems over each time period of the swap's tenor structure, with possible communication at the end of the time period. Our implementation of the pricing procedure on clusters of GPU involves (i) efficiently solving each independent sub-problems on a GPU via a parallelization of the ADI timestepping technique, and (ii) utilizing MPI for the communication between pricing processes at the end of each time period of the swap's tenor structure. The results of this paper show that GPU clusters can provide a significant increase in performance over GPUs, when pricing exotic cross-currency interest rate derivatives with path-dependence features.

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 [6]. 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 our proposed parallel pricing method is expected to deliver even larger speedups and better performance when pricing path-dependent foreign exchange interest rate derivatives.

References

[1] Sippel, J., Ohkoshi, S.: All power to PRDC notes. Risk Magazine 15(11), 1–3 (2002)

[2] Piterbarg, V.V.: TARNs: Models, Valuation, Risk Sensitivities. Wilmott Magazine 14, 62–71 (2004)

[3] Abbas-Turki, L.A., Vialle, S., Lapeyre, B., Mercier, P.: 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, pp. 1–8. IEEE Computer Society (2009)

[4] Murakowski, D., Brouwer, W., Natoli, V.: 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, pp. 1–4. IEEE Computer Society (2010)

[5] Tian, Y., Zhu, Z., Klebaner, F.C., Hamza, K.: Pricing barrier and American options under the SABR model on the graphics processing units. Concurrency and Computation: Practice and Experience, 867–879 (2012)

[6] Dang, D.M., Christara, C., Jackson, K.: Graphics processing unit pricing of exotic cross-currency interest rate derivatives with a foreign exchange volatility skew model. Journal of Concurrency and Computation: Practice and Experience (to appear, 2013), `http://onlinelibrary.wiley.com/doi/10.1002/cpe.2824/abstract`

[7] Dang, D.M., Christara, C., Jackson, K.: A parallel implementation on GPUs of ADI finite difference methods for parabolic PDEs with applications in finance. Canadian Applied Mathematics Quarterly (CAMQ) 17(4), 627–660 (2009)

[8] Dang, D.M., Christara, C., Jackson, K.: An efficient graphics processing unit-based parallel algorithm for pricing multi-asset American options. Journal of Concurrency and Computation: Practice and Experience 24(8), 849–866 (2012)

[9] Egloff, D.: GPUs in financial computing part III: ADI solvers on GPUs with application to stochastic volatility. Wilmott, 50–53 (March 2011)

[10] Egloff, D.: Pricing financial derivatives with high performance finite difference solvers on GPUs. In: Hwu, W.-M.W. (ed.) GPU Computing Gems Jade Edition. Applications of GPU Computing Series, pp. 309–322 (2012)

[11] Dang, D.M., Christara, C., Jackson, K., Lakhany, A.: An efficient numerical PDE approach for pricing foreign exchange interest rate hybrid derivatives. To appear in the Journal of Computational Finance (2012), `http://ssrn.com/abstract=2028519`

[12] Piterbarg, V.: Smiling hybrids. Risk Magazine 19(5), 66–70 (2006)

[13] Gropp, W., Lusk, E., Skjellum, A.: Using MPI-2: Advanced Features of the Message Passing Interface, 1st edn. MIT Press (1999)

[14] Andersen, L.B., Piterbarg, V.V.: Interest Rate Modeling, 1st edn. Atlantic Financial Press (2010)

[15] Dang, D.M., Christara, C.C., Jackson, K., Lakhany, A.: A PDE pricing framework for cross-currency interest rate derivatives. In: Proceedings of the 10th International Conference in Computational Science (ICCS). Procedia Computer Sciences, vol. 1, pp. 2371–2380. Elsevier (2010)

[16] Hull, J., White, A.: One factor interest rate models and the valuation of interest rate derivative securities. Journal of Financial and Quantitative Analysis 28(2), 235–254 (1993)
[17] Haentjens, T., In 't Hout, K.J.: Alternating direction implicit finite difference schemes for the Heston-Hull-White partial differential equation. Journal of Computational Finance 16(1), 83–110 (2012)
[18] Hundsdorfer, W.: Accuracy and stability of splitting with stabilizing corrections. Appl. Numer. Math. 42, 213–233 (2002)
[19] In 't Hout, K.J., Welfert, B.D.: Unconditional stability of second-order ADI schemes applied to multi-dimensional diffusion equations with mixed derivative terms. Appl. Numer. Math. 59, 677–692 (2009)
[20] NVIDIA: NVIDIA Compute Unified Device Architecture: Programming Guide Version 3.2. NVIDIA Developer Web Site (2010), `http://developer.nvidia.com/object/gpucomputing.html`
[21] Dang, D.M.: Modeling multi-factor financial derivatives by a Partial Differential Equation approach with efficient implementation on Graphics Processing Units. PhD thesis, Department of Computer Science, University of Toronto, Toronto, Ontario, Canada (2011)
[22] Harris, M., Sengupta, S., Owens, J.D.: Parallel prefix sum (scan) with CUDA. In: GPU Gems 3, pp. 851–877. NVIDIA (2007)