Modeling multi-factor financial derivatives by a Partial Differential Equation approach with efficient implementation on Graphics Processing Units

by

Duy Minh Dang

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto

Abstract

Modeling multi-factor financial derivatives by a
Partial Differential Equation approach with
efficient implementation on Graphics Processing Units

Duy Minh Dang
Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto
2011

This thesis develops efficient modeling frameworks via a Partial Differential Equation (PDE) approach for multi-factor financial derivatives, with emphasis on three-factor models, and studies highly efficient implementations of the numerical methods on novel high-performance computer architectures, with particular focus on Graphics Processing Units (GPUs) and multi-GPU platforms/clusters of GPUs. Two important classes of multi-factor financial instruments are considered: cross-currency/foreign exchange (FX) interest rate derivatives and multi-asset options.

For cross-currency interest rate derivatives, the focus of the thesis is on Power Reverse Dual Currency (PRDC) swaps with three of the most popular exotic features, namely Bermudan cancelability, knockout, and FX Target Redemption. The modeling of PRDC swaps using one-factor Gaussian models for the domestic and foreign interest short rates, and a one-factor skew model for the spot FX rate results in a time-dependent parabolic PDE in three space dimensions. Our proposed PDE pricing framework is based on partitioning the pricing problem into several independent pricing subproblems over each time period of the swap's tenor structure, with possible communication at the end of the time period. Each of these subproblems requires a solution of the model PDE. We

then develop a highly efficient GPU-based parallelization of the Alternating Direction Implicit (ADI) timestepping methods for solving the model PDE. To further handle the substantially increased computational requirements due to the exotic features, we extend the pricing procedures to multi-GPU platforms/clusters of GPUs to solve each of these independent subproblems on a separate GPU. Numerical results indicate that the proposed GPU-based parallel numerical methods are highly efficient and provide significant increase in performance over CPU-based methods when pricing PRDC swaps. An analysis of the impact of the FX volatility skew on the price of PRDC swaps is provided.

In the second part of the thesis, we develop efficient pricing algorithms for multi-asset options under the Black-Scholes-Merton framework, with strong emphasis on multi-asset American options. Our proposed pricing approach is built upon a combination of (i) a discrete penalty approach for the linear complementarity problem arising due to the free boundary and (ii) a GPU-based parallel ADI Approximate Factorization technique for the solution of the linear algebraic system arising from each penalty iteration. A timestep size selector implemented efficiently on GPUs is used to further increase the efficiency of the methods. We demonstrate the efficiency and accuracy of the proposed GPU-based parallel numerical methods by pricing American options written on three assets.

Dedication

To my parents for their sacrifice

which allowed me to embark on this journey.

To my wife for her unconditional love and endless support

every step of the way.

Acknowledgements

To my supervisors, Professors Christina Christara and Ken Jackson, I owe a tremendous debt of gratitude for their invaluable research guidance, inspiration, encouragement, and support. It has been a great pleasure and a very rewarding experience to work with these two wonderful advisors.

I would like to sincerely thank my PhD committee members, Professors Wayne Enright and Alex Kreinin, for their valuable input into this thesis during the past four years. I would also like to thank the external examiner, Professor John Chadam, for raising interesting points in his report, that led to improvements in the thesis, and Professor Luis Seco for agreeing to be on my Ph.D. final oral examination committee, and posing challenging questions during the oral. Special thanks to Dr. Asif Lakhany of Algorithmics Inc. for introducing me to the problem of pricing PRDC swaps, and for many interesting discussions and research ideas, and to Dr. Vladimir Piterbarg of Barclays Capital, for useful discussions regarding PRDC swaps.

During my PhD studies, I have been able to present my work at various leading international conferences. I am very grateful for the financial support from Professors Christina Christara and Ken Jackson, as well as from the Natural Science and Engineering Research Council (NSERC) of Canada. I am thankful for the comments and suggestions I received from various conference participants and journal referees.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The rapid growth of the financial markets over the past few decades has spawned many intellectually challenging problems to be solved. These problems have evolved from the simple one-factor Black-Scholes-Merton equation [7, 50] to highly-complex multi-factor models with constraints that originate from important practical applications. For instance, option contracts can have more than one underlying asset and different types of payoff functions, including, in particular, path-dependent payoffs. Also, option contracts with optionalities, such as early exercise features, have become very popular. Moreover, the financial markets have become more diverse, with trading not only of stocks, but also of numerous types of financial derivatives. For example, multi-currency interest rate derivatives, especially those with exotic features, such as Bermudan cancelability, have become increasingly important and are traded in large quantities in Over-the-Counter (OTC) markets. These challenging problems are now spawning radical changes in computational methods in finance: more mathematically sophisticated and efficient computational methods are in great demand for the valuation and risk-management of complex financial instruments.

Closed-form solutions, such as the Black-Scholes-Merton [7, 50] formula for vanilla European put and call options, are not available for most financial derivatives. Hence,

such derivatives must be priced by numerical techniques. Although several pricing approaches can be used, such as Monte Carlo (MC) simulation [19, 21, 48], or tree-based (lattice) methods [32, 39], for problems in low dimensions, i.e. less than five dimensions, the Partial Differential Equation (PDE) approach is a very popular choice, due to its efficiency and global nature. In addition, accurate hedging parameters, such as delta and gamma, which are essential for risk-management of the financial derivatives, are generally much easier to compute via a PDE approach than via other methods.

When solving multi-factor problems in finance by a PDE approach, each stochastic factor in the model gives rise to a spatial variable in the PDE. Due to the "curse of dimensionality" associated with high-dimensional PDEs, the pricing of such derivatives via the PDE approach is challenging. In addition, for many financial contracts, such as multi-currency interest rate derivatives, additional complexity may arise from multiple cash flow dates and exotic features. Moreover, when stochastic processes in the pricing model are correlated, as is common in financial modeling, the resulting PDE possesses cross spatial derivatives, which makes solving the associated problems numerically even more challenging.

For the numerical solution of low-dimensional PDE models in finance, such as a three-factor model for PRDC swaps, the valuation of the securities can be efficiently calculated by utilizing a level-splitting scheme of the Alternating Direction Implicit (ADI) type along the time-dimension, the computation of which requires the solution of a sequence of tridiagonal linear systems at each timestep. Examples of applications of different ADI schemes in finance can be found in [5, 17, 35, 46, 66]. Among them, the most popular are perhaps the ADI schemes proposed by Craig and Sneyd in [13] and by Hundsdorfer and Verwer [33, 34], because they can handle effectively cross spatial derivatives. More specifically, these two schemes, when combined with second-order central finite differences (FD) for the discretization of the space variables, are unconditionally stable and

efficient second-order methods in both space and time when applied to PDEs with cross derivative terms. However, a disadvantage of the Craig and Sneyd scheme is that it cannot maintain both unconditional stability and second-order accuracy when the number of spatial dimensions is greater than three [13, 36], which potentially prevents extending the method to higher-dimensional applications. In addition, it has been noted in [35] that the Craig and Sneyd scheme may exhibit undesirable convergence behavior when the payoff functions are non-smooth, which is quite common for financial applications. Hence smoothing techniques, such as Rannacher timestepping [60], may be required. On the other hand, the ADI scheme introduced by Hundsdorfer and Verwer is unconditionally stable for arbitrary spatial dimensions [36], and, at the same time, also effectively damps the errors introduced by non-smooth payoff functions [35]. It is worth noting that classical ADI algorithms, such as the Douglas and Rachford scheme [15], although unconditionally stable, are only first-order in time and second-order in space when cross spatial derivatives are present.

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. [1, 2, 51, 67]). The literature on GPU-based PDE methods for pricing options written on multiple assets is rather sparse, with scattered work presented at conferences or workshops [20]. The literature on GPU-based PDE methods for pricing multi-currency interest rate derivatives is even less developed.

In a broad sense, the thesis presents new and highly efficient modeling frameworks via a PDE approach for multi-factor financial derivatives that can be easily tailored

and extended to a variety of applications. More specifically, the thesis studies (i) new and efficient PDE-based pricing methods for two important classes of multi-factor financial derivatives, namely cross-currency/foreign-exchange (FX) interest rate derivatives and multi-asset options, i.e. options written on several assets; (ii) a highly efficient implementation of the aforementioned computational methods on novel high-performance computer architectures, such as GPUs and multi-GPU platforms/clusters of GPUs, to further increase their efficiency; and (iii) an investigation of important modeling issues pertaining to FX interest rate derivatives, such as the sensitivity of the computed prices of these derivatives to the FX volatility skew.

In the remainder of this chapter, we present a brief introduction to the areas of cross-currency/foreign-exchange interest rate derivatives and multi-asset options. Also, we discuss the motivation for the research of the thesis and highlight the main contributions of this work to the area of numerical modeling of multi-factor financial derivatives. The outline of the thesis is presented towards the end of this chapter.

1.1 Cross-currency/FX Interest Rate Derivatives

In the current era of wildly fluctuating exchange rates, cross-currency interest rate derivatives, especially FX interest rate derivatives, are of enormous practical importance. The emphasis of the research work in this area is on long-dated (maturities of 30 years or more) FX interest rate derivatives, namely Power Reverse Dual Currency (PRDC) swaps, one of the most widely traded and liquid cross-currency interest rate derivatives [64]. As such, the modeling of such instruments is of great interest to practitioners and academics alike.

Cross-currency/FX interest rate derivatives can be viewed as financial contracts whose values are contingent on the evolution of the two interest rates, namely the domestic and foreign interest rates, and the spot FX rate that links the two currencies. While a wide

variety of interest rate derivatives are traded on the financial markets, interest rate swaps occupy a position of central importance in the OTC derivatives market [32]. An interest rate swap in general, and a cross-currency/FX interest rate swap in particular, can be viewed as an agreement between two parties to exchange fund flows in the future. The agreement specifies (i) the set of dates when the fund flows are exchanged, referred to as the swap's tenor structure, and (ii) the way in which they are to be calculated. In a cross-currency/FX interest rate swap, such as a PRDC swap, the calculation of the fund flows involves the future values of one or both interest rates, the spot FX rate between the two currencies, and possibly, other market variables.

As long-dated FX interest rate derivatives, such as PRDC swaps, are exposed to moves in both the spot FX rate and the interest rates in both currencies, multi-factor pricing models must have at least three factors, namely the domestic and foreign interest rates and the spot FX rate. The most common pricing approach for long-dated FX interest rate derivatives is MC simulation. The open literature on pricing methods for cross-currency interest rate swaps via a PDE approach is very sparse [14, 49]; discussions focus on "vanilla" cross-currency swaps or swaptions only. The practical importance of highly complex cross-currency/FX interest rate derivatives, such as PRDC swaps, especially those with exotic features, and the lack of published work in the literature on efficient PDE-based pricing frameworks for such derivatives formed the main motivation for our research in this area.

In this thesis, we discuss the modeling of PRDC swaps using two one-factor Gaussian models for the two stochastic interest short rates, and a one-factor FX skew model with a local volatility function for the spot FX rate as proposed in [58]. (The focus of [58] is efficient calibration techniques for local volatility functions in a cross-currency framework. The numerical solution of PRDC swaps was not considered there.) This pricing model gives rise to a time-dependent parabolic PDE in three space dimensions. Variations of

PRDC swaps with exotic features, such as Bermudan cancelability, knockout, or FX Target Redemption (FX-TARN), are much more popular than "vanilla" PRDC swaps. While pricing "vanilla" PRDC swaps presents by itself a computational challenge, due to the high-dimensionality of the PDE and multiple fund flow dates of the swaps' tenor structure, the exotic features significantly increase the complexity of the pricing, due to their very different natures, as well as their levels of suitability to a PDE-based pricing approach.

In the first part of the thesis, we develop a comprehensive and highly efficient PDE-based pricing framework for long-dated FX interest rate swaps, with strong emphasis on PRDC swaps. The three most popular exotic features, namely Bermudan cancelability, knockout and FX-TARN, are investigated in detail. The first contribution of this research is a flexible PDE pricing framework that can efficiently handle the early exercise features of Bermudan cancelability, as well as strong path-dependency of the FX-TARN feature. More specifically, our general PDE pricing framework for these derivatives is based on partitioning the pricing problem into multiple independent pricing subproblems over each time period of the swap's tenor structure, each of which requires the solution of the model-dependent PDE. In our case, the model PDE is a three-dimensional time-dependent parabolic PDE with all cross derivatives, due to the correlation between stochastic processes in the pricing model. In particular, over each time period of the swap's tenor structure, the pricing of a Bermudan cancelable PRDC swap can be divided into two independent pricing subproblems, while the pricing of an FX-TARN PRDC swap can be divided into multiple independent pricing subproblems, with possible communication at the end of the time period. Each of these subproblems can be solved efficiently using the second-order central FD methods for the spatial discretization combined with the ADI timestepping technique for the time discretization of the model PDE. We focus on the ADI scheme introduced by Hundsdorfer and Verwer [34], due to its favorable

characteristics.

The second contribution of this work is a highly efficient implementation of the PDE-based pricing framework on novel high-performance computer architectures, namely GPUs and multi-GPU platforms/clusters of GPUs. More specifically, 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 solving the model PDE via a parallelization of the ADI timestepping technique. The main components of our GPU-based parallelization of the ADI scheme are (i) an efficient parallel implementation of the explicit Euler predictor step, and (ii) a parallel solver for the independent tridiagonal systems arising in the three implicit, but unidirectional, corrector steps. Although we focus on the ADI scheme introduced by Hundsdorfer and Verwer [34], the parallelization method presented in this thesis can be easily tailored for other ADI schemes. To further handle the substantially increased computational requirements due to the exotic features, which give rise to multiple independent pricing subproblems/PDEs, we extend the pricing procedures to multi-GPU platforms/clusters of GPUs to solve these independent PDEs on a separate GPU, with possible communication at the dates of the swap's tenor structure. Numerical results indicate that the proposed GPU-based parallel pricing methods are very efficient.

An important issue in the modeling of long-dated FX interest rate derivatives is the sensitivity of the price of these derivatives to the skews observed in the FX volatility smiles. In this regard, the research also focuses on quantifying the exposure of long-dated FX interest rate swaps in general, and PRDC swaps in particular, to the FX volatility skew. More specifically, we study and compare the improvements of the FX skew model, in which a local volatility function is employed for generating the skews present in the FX volatility smiles, over the log-normal model. The results of our investigation indicate a strong sensitivity of the price of PRDC swaps to the skew of the FX volatility smiles.

These findings highlight the importance of having a proper FX skew model for pricing and risk managing PRDC swaps.

1.2 Multi-asset Options

A multi-asset option, i.e. an option written on more than one asset, is a contract between the holder and the writer that gives the right, but not an obligation, to the holder to buy or sell a specified basket of more than one underlying asset by a certain time for a given price. In particular, a multi-asset call option gives the holder the right to buy, whereas a multi-asset put option gives the holder the right to sell its basket of underlying assets, for a prescribed amount, known as the strike price. It is important to determine a fair price for an option accurately.

An important feature of such contracts is the time when the contract holders can exercise their rights. If this occurs only at the maturity date, the option is classified as a *European* option. If holders can exercise any time up to and including the maturity date, the option is said to be an *American* option.

Option pricing theory has advanced tremendously since the seminal work by Black and Scholes [7] and Merton [50]. At the forefront of these advances has been the development of option pricing solutions within the original framework of Black-Scholes and Merton, in which the interest rate is constant and the volatility is a deterministic functions of time and/or the underlying assets. Within this framework, the price of a multi-asset European option satisfies the so-call multi-dimensional Black-Scholes-Merton PDE [43, 73]. The solution of this time-dependent parabolic PDE in a high-dimensional application, such as options written on three assets, presents a computational challenge. In this case, the multi-dimensional Black-Scholes-Merton PDE can be efficiently solved using the GPU-based parallel ADI timestepping techniques in combination with FD methods for the space discretization, similar to those developed for PRDC swaps. However, most options

traded on exchanges are of American-style. For American options, the Black-Scholes-Merton model results in a free boundary problem, due to the early exercise feature of the options [66, 70]. Since explicit closed-form solutions to the American option pricing problem cannot be found in general, sophisticated numerical methods must be used for the pricing of an American option.

Using a PDE approach, the American option pricing problem can be formulated as a time-dependent linear complementarity problem (LCP) with the inequalities involving the Black-Scholes-Merton PDE and some additional constraints [69]. Consequently, the problem of pricing multi-asset American options, such as options written on three assets, is both mathematically challenging and computationally intensive. In the area of option pricing, the focus of the thesis is on multi-asset American options, due to the aforementioned challenges.

Recently, several approaches for handling the LCP have been developed. In particular, various penalty methods were discussed in [18, 54, 55, 74]. In the thesis, we adopt the penalty method of [18] to solve the LCP. In this approach, a penalty term is added to the discretized equations to enforce the early exercise constraint. The solution of the resulting discrete nonlinear equations at each timestep can be computed via a penalty iteration.[1] An advantage of the penalty method of [18] is that it is readily extendible to handle multi-factor problems. In a multi-dimensional application, applying direct methods, such as LU factorization, to solve the linear system arising at each penalty iteration can be computationally expensive. A very popular alternative is to use iterative methods, such as Biconjugate Gradient Stabilized (BiCGStab), in combination with a preconditioning technique, such as an Incomplete LU factorization [66]. Another possible approach is to employ ADI Approximate Factorization (AF) techniques, which involve solving only a few tridiagonal systems in each spatial dimension. It is rather surprising

[1]The penalty iteration described in [18] is essentially a Newton iteration, but, to be consistent with [18], we use the term "penalty iteration" throughout this thesis.

that, while these efficient techniques have been widely used in the numerical solution of multi-dimensional nonlinear PDEs arising in computational fluid dynamics [72], to the best of our knowledge, these techniques have not been successfully extended to multi-asset American option pricing. These shortcomings motivated our work.

In the second part of the thesis, we develop efficient pricing algorithms for multi-asset options under the Black-Scholes-Merton framework, with strong emphasis on multi-asset American options. Our pricing approach for multi-asset American options is built upon a combination of the discrete penalty approach of [18] for the LCP arising due to the free boundary and a GPU-based parallel ADI-AF technique combined with FD discretization methods for the solution of the linear algebraic system arising from each penalty iteration. Although we primarily focus on options written on three assets, many of the ideas and results developed here can be naturally extended to higher-dimensional applications with constraints.

The ADI-AF techniques developed in this thesis can be viewed as being based on the idea of the splitting techniques of the ADI timestepping methods described in the previous section, but at the (discrete) matrix level. More specifically, using FD methods for the space discretization and a standard timestepping technique, such as Crank-Nicolson (CN), the pricing of an American option written on three assets via the penalty approach of [18] requires the solution of a matrix problem of the form

$$\mathbf{A}\mathbf{v} = \mathbf{b}$$

at each penalty iteration. Here, $\mathbf{v}$ is the vector of unknowns, $\mathbf{A}$ is the matrix of FD approximation to the differential operator, and $\mathbf{b}$ is a vector of known values. We then develop a technique to approximately factorize the matrix $\mathbf{A}$ into a product of 3 tridiagonal matrices of the form $\mathbf{A}_1\mathbf{A}_2\mathbf{A}_3$, where the matrix $\mathbf{A}_i$, $i = 1, \ldots, 3$, is the part of $\mathbf{A}$ that corresponds to the FD discretization of the spatial derivative in the ith spatial

direction. Then, instead of solving $\mathbf{Av} = \mathbf{b}$, we solve

$$\mathbf{A}_1\mathbf{A}_2\mathbf{A}_3\mathbf{v} = \mathbf{b} + \mathbf{c},$$

where the vector $\mathbf{c}$ is a correction term arising from the approximate factorization of the matrix $\mathbf{A}$. The solution process can be efficiently realized via a sequence of tridiagonal solutions as

$$\begin{cases} \mathbf{A_1}\mathbf{v}_1 &= \mathbf{b} + \mathbf{c}, \\ \mathbf{A_2}\mathbf{v}_2 &= \mathbf{v}_1, \\ \mathbf{A_3}\mathbf{v} &= \mathbf{v}_2. \end{cases}$$

The idea behind the ADI-AF technique presented above can be easily generalized to higher-dimensional applications.

A GPU-based parallelization of the ADI-AF scheme described above can be viewed as a natural extension of the parallelization of the ADI timestepping method discussed earlier for PRDC swaps. More specifically, the computation of the vector $\mathbf{b} + \mathbf{c}$ resembles the explicit Euler predictor step, while the solutions of the tridiagonal systems is essentially the same as the three implicit corrector steps, each of which involves solving a block-diagonal system with tridiagonal blocks along a spatial dimension. A timestep size selector, efficiently implemented on the GPU, is used to further increase the performance of the methods.

1.3 Thesis Outline

The remainder of the thesis is organized as follows. Chapter 2 first presents an introduction to dynamics of PRDC swaps and popular exotic features, and then introduces a particular three-factor pricing model with the FX volatility skew obtained via a local volatility function and the associated PDE. Chapter 3 develops PDE-based pricing algorithms for PRDC swaps with exotic features. Efficient implementation on a multi-GPU

platform/GPU cluster of these pricing algorithms are presented in Chapter 4. In particular, a GPU-based parallelization of the ADI timestepping scheme is discussed in great detail in this chapter. Numerical results for pricing PRDC swaps together with relevant discussions and analyses are given in Chapter 5. Chapter 6 discusses the pricing of European and American options written on three assets under the Black-Scholes-Merton framework, with strong emphasis on American options. Numerical results for multi-asset options are presented in Chapter 7. Chapter 8 summarizes the main contributions of the thesis and outlines possible directions for further research. Commonly used abbreviations and notations are presented in Appendix A. A glossary of relevant terms is given in Appendix B. Appendix C derives relevant FD approximations and matrix formulas used in this thesis. Supplementary numerical results for PRDC swaps obtained with preconditioned iterative methods and relevant discussions are presented in Appendix D. Miscellaneous derivations are given in Appendix E. A diagram of the main components of the thesis and the dependence between chapters is give in Figure 1.3.1.

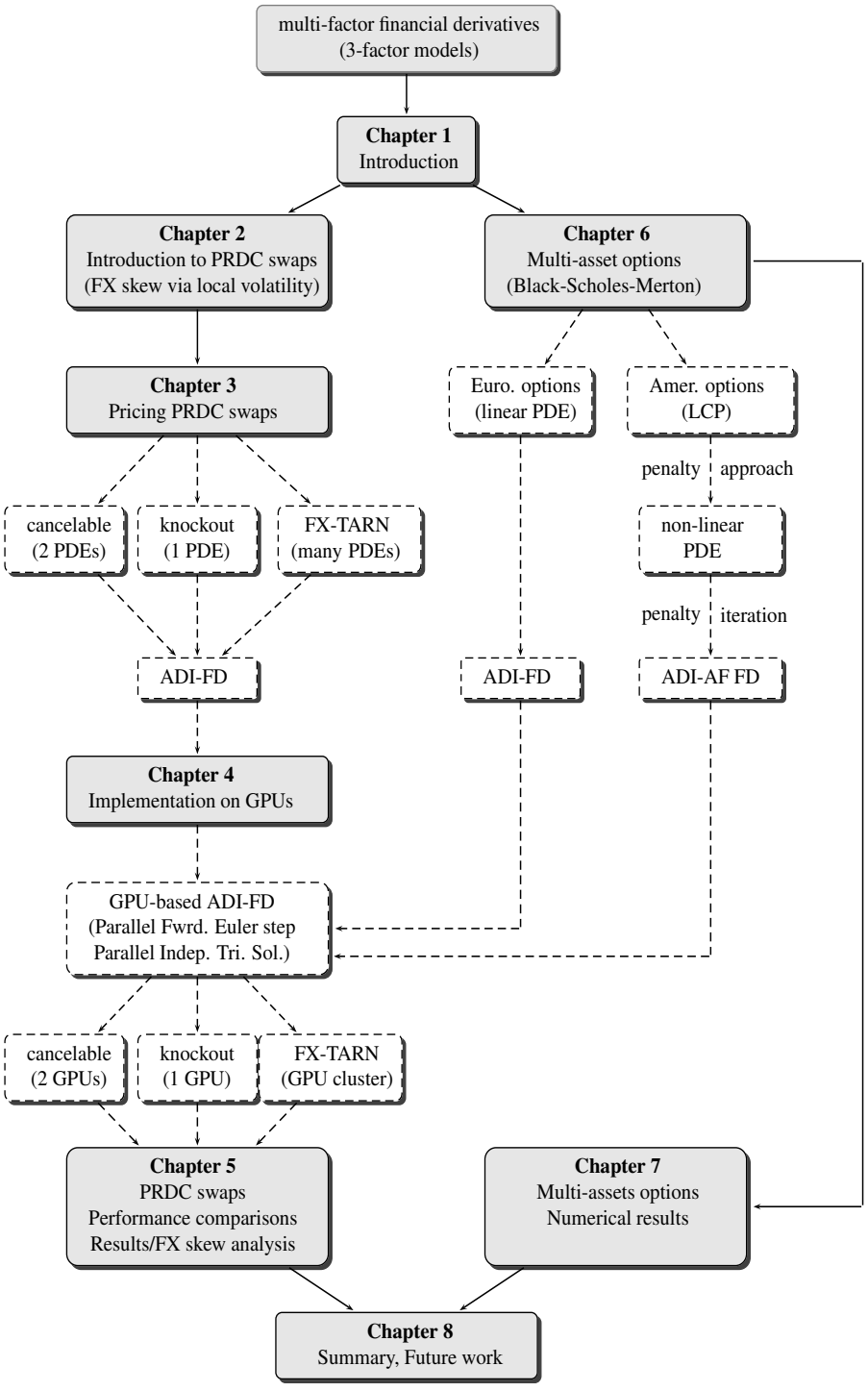


Figure 1.3.1: Important components and dependence between chapters of the thesis.

Chapter 2

Introduction to PRDC Swaps

The purpose of this chapter is twofold. In Section 2.1, we review several fundamental definitions and concepts of interest rate theory, and introduce relevant interest rate instruments, namely interest rate swaps and Bermudan swaptions. The aim of this part is to provide the background needed for the description of PRDC swaps in the next section, as well as the discussions of the pricing algorithms for PRDC swaps, and the analysis of their prices presented in later chapters. In Section 2.2 of this chapter, we first discuss the dynamics of PRDC swaps and popular exotic features, and then present the FX skew model of [58], an overview of the model calibration, and a derivation of the associated pricing PDE.

2.1 Preliminaries

Since the theory on interest rate modeling is vast and a very large range of interest rate instruments are actively traded, only fundamental concepts and derivatives relevant to this thesis are presented. We refer interested readers to [4, 8] for detailed discussions on these subjects.

2.1.1 Definitions and Notation

The Bank Account and the Short Rate

We denote by $B(t)$ the value of a bank account at time $t \geq 0$. We assume that $B(0) = 1$ and that the bank account evolves according to the differential equation

$$dB(t) = r(t)B(t)dt, \tag{2.1}$$

where $r(t)$ is a positive function of time. As a result, we have

$$B(t) = e^{\int_0^t r(s)ds}.$$

In the above formulas, $r(t)$ is the instantaneous interest rate at which the value of a bank account accrues. This rate is referred to as the *instantaneous spot rate*, or more commonly as the *short rate*.

Zero-Coupon Bonds

A T-maturity zero-coupon bond is a contract that guarantees its holder the payment of one unit of currency at time T, with no intermediate payments. The contract value at time $t \in [0, T]$ is denoted by $P(t, T)$. Clearly, $P(t, T) < 1$, for all $t < T$, and $P(T, T) = 1$ for all T. It is also clear that $P(t, T)$ is the time t value of one unit of currency to be paid at time T, the maturity of the contract.

Year Fraction

We denote by $\nu(t, T)$ the chosen time measure between t and $T \geq t$, which is usually referred to as the *year fraction* between the dates t and T. The particular choice that is made to measure the time between two dates is referred to as the *day-count convention*. Examples of day-count conventions are:

- Actual/365: With this convention, a year is 365 days long and the year fraction between two dates is the actual number of days between them divided by 365.

- Actual/360: In this case, a year is assumed to be 360 days long.

Simply-Compounded Spot Interest Rate

The simply compounded spot interest rate prevailing at time $t \geq 0$ for the maturity $T \geq t$, denoted by $L(t,T)$, is the constant rate at which an investment has to be made to produce an amount of one unit of currency at maturity, starting from $P(t,T)$ units of currency at time t, when accruing occurs proportionally to the investment time. That is,

$$L(t,T) = \frac{1 - P(t,T)}{\nu(t,T)P(t,T)}. \tag{2.2}$$

It can be shown that the short rate and the simply-compounded spot interest rate are related via [8]

$$r(t) = \lim_{T \to t^+} L(t,T). \tag{2.3}$$

Unless otherwise stated, in this thesis, we assume that the simply-compounded spot interest rates $L(t,T)$ are the London Interbank Offered Rates (LIBOR) quoted in the interbank market.

Tenor Structure

Most interest rate derivatives involve multiple fund flows taking place on a set of fixed dates, usually equidistantly spaced, often referred to as a *tenor structure*,

$$T_0 = 0 < T_1 < \cdots < T_\beta < T_{\beta+1}, \tag{2.4}$$

with $\nu_\alpha = \nu(T_{\alpha-1}, T_\alpha) = T_\alpha - T_{\alpha-1}, \quad \alpha = 1, 2, \ldots, \beta+1$. Here, each of T_α, $\alpha = 0, 1, \ldots, \beta+1$, is referred to as a *date* of the tenor structure; 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. Each of the time intervals $[T_{\alpha-1}, T_\alpha]$, $\alpha = 1, 2, \ldots, \beta+1$, is called a *period* of the tenor structure.

To simplify the notation, we denote by $L(T_\alpha)$ the LIBOR rate $L(T_{\alpha-1}, T_\alpha)$ for the α-th period, $\alpha = 1, \ldots, \beta + 1$. Similarly, we use $P(T_\alpha)$ as a short-hand notation for $P(T_{\alpha-1}, T_\alpha)$, $\alpha = 1, \ldots, \beta + 1$. For use later in the thesis, 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 after and before, respectively, the date T_α.

In the context of multi-currency markets, we consider an economy with two currencies, "domestic" and "foreign". Unless otherwise stated, in the thesis, the sub-scripts d and f are used to indicate domestic and foreign, respectively. For instance, $P_i(t, T)$, $i = d, f$, are the prices at time t in their respective currencies, of the domestic and foreign zero-coupon bonds, respectively, with maturity T.

Spot FX Rate

We denote by $s(t)$, $t \geq 0$, the spot FX rate, the number of units of domestic currency per one unit of foreign currency prevailing at time t. Essentially, the spot FX rate at time t is the rate at which values in the foreign currency are converted into the domestic currency at time t.

When the spot FX rate decreases, we say the domestic currency *strengthens* against the foreign currency. Conversely, when the spot FX rate increases, we say the domestic currency *weakens* against the foreign currency.

Forward FX Rate

We denote by $F(t, T)$ the forward FX rate prevailing at time $t \geq 0$ for maturity $T \geq t$. This is the number of units of domestic currency per one unit of foreign currency as quoted at time t for exchange at time T. Following no-arbitrage arguments, it is straight-forward to obtain the following formula for the forward FX rate [4, 63]:

$$F(t, T) = \frac{P_f(t, T)}{P_d(t, T)} s(t). \tag{2.5}$$

2.1.2 Interest Rate Derivatives

In this subsection, we describe the dynamics of two relevant, also very popular, interest rate derivatives, namely interest rate swaps and Bermudan swaptions, in the context of single-currency markets. The discussion in this subsection provides the background details needed to understand the dynamics and the pricing of PRDC swaps described later. Other popular interest rate instruments, such as caps and floors, are not relevant to the research developed in this thesis, and hence omitted.

Interest Rate Swaps

A *swap* is a generic term for an OTC derivative in which two parties agree to exchange one stream of fund flows for another stream of fund flows according to a pre-arranged formula. These streams are often referred to as the *legs* of the swap. A "vanilla" *fixed-for-floating* interest rate swap, usually referred to as a fixed-for-floating swap, is a swap in which one leg is a stream of fixed rate payments, i.e. the *fixed leg*, whereas, the other one is based on a floating rate, i.e. the *floating leg*, most often LIBOR. These two legs are denominated in the same currency, have the same notional, and expire on the same date. In the description which follows, the fixed and floating legs occur on the same set of dates with the same year fractions. Although the generalization to a different set of dates and day-count conventions for the two legs is straightforward, we present the simplified version of the two legs to avoid cumbersome notation irrelevant to PRDC swaps.

There are two parties involved in a fixed-for-floating swap, namely the payer of the fixed leg, who is also the receiver of the floating leg, and the receiver of fixed leg, who is also the payer of the floating leg. Given the tenor structure (2.4), in a fixed-for-floating swap, at each date $T_\alpha, \alpha = 1, \ldots, \beta + 1$ of the tenor structure, the payer of the fixed leg pays out the amount $\nu_\alpha K N$ corresponding to a pre-agreed fixed interest rate K, on a notional value N and with a year fraction ν_α between $T_{\alpha-1}$ and T_α, in return for the

floating rate payment $\nu_\alpha L(T_\alpha)N$, corresponding to the LIBOR rate $L(T_\alpha)$ on the notional value N for the period $[T_{\alpha-1}, T_\alpha]$, as observed at time $T_{\alpha-1}$, and defined by

$$L(T_\alpha) = \frac{1 - P(T_\alpha)}{\nu_\alpha P(T_\alpha)}, \tag{2.6}$$

which follows from (2.2). Note that $L(T_\alpha)$ 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". A diagram of fund flows in a fixed-for-floating interest rate swap is presented in Figure 2.1.1.

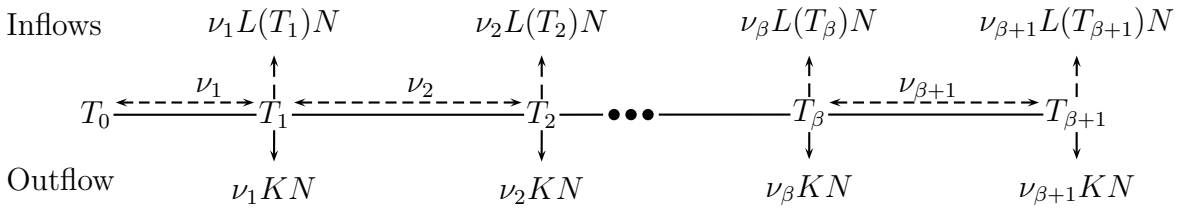


Figure 2.1.1: Fund flows in a generic fixed-for-floating interest rate swap. Inflows and outflows are with respect to the point-of-view of the fixed leg payer.

The pricing of fixed-for-floating swaps is relatively simple. In fact, the price of a fixed-for-floating swap at time t can be obtained using only the term structure of interest rates observed on that date, and is not affected by the dynamics of the interest rates. In general, the price at time T_0 of a fixed-for-floating swap is reflected via an exchange of a fixed rate coupon between the the two parties of the swap at time T_0 that is not included in the description above. (See Remark 2.2.2 on page 23 for a related discussion in the context of PRDC swaps.) There is a unique value of the fixed interest rate K that makes the fixed-for-floating swap have value zero to both parties, i.e. no fund exchange at time T_0. This value of the the interest rate is referred to as the *forward swap rate*. More details can be found in the literature, e.g. [4, 8, 52]. In practice, fixed-for-floating swaps are often used by speculators to make bets on the future direction of interest rates, or by hedgers to transform fixed rate obligations into floating ones, or vice versa.

European and Bermudan Swaptions

A *European swaption* gives the holder of the option a right, but not an obligation, to enter an interest rate swap, referred to as the *underlying swap*, at a pre-determined future time called the *swaption maturity*. A *payer* swaption is an option to pay the fixed leg (and receive the floating leg) of a fixed-for-floating swap. On the other hand, a *receiver* swaption is an option to receive the fixed leg (and pay the floating leg) of a fixed-for-floating swap. Usually, the swaption maturity coincides with the first date T_0 of the tenor structure of the underlying swap.

A *Bermudan swaption* is an extension of a European swaption, which gives its holder an option to enter the underlying swap on any of several pre-specified dates, i.e. the set of *exercise dates*, provided that this right has not already been exercised at a previous time, instead of just one pre-determined date as in a European swaption. Note that in practice, the set of exercise dates of a Bermudan swaption may have no overlap at all with the set of dates of the underlying swap's tenor structure. However, for the sake of simplicity, we omit some generality of a Bermudan swaption, while focusing on the features relevant to this thesis.

More specifically, given an interest rate swap with the tenor structure (2.4), the holder of a standard Bermudan swaption has the right, but not an obligation, to exercise it on any of the dates $\{T_\alpha\}_{\alpha=1}^{\beta}$. Once exercised on the date T_α, the option disappears, and the holder enters the part of the underlying swap remaining after the exercise date T_α, i.e. the holder is exposed to all fund flows of the underlying swap scheduled on or after $T_{\alpha+1}$. The period up to T_1 is known as the *lock-out* or *no-call* period. In practice, a Bermudan swaption with $T_1 = 1$ year and $T_{\beta+1} = 10$ years is usually referred to as a "10 no-call 1" Bermudan swaption. An illustration of a Bermudan swaption is presented in Figure 2.1.2.

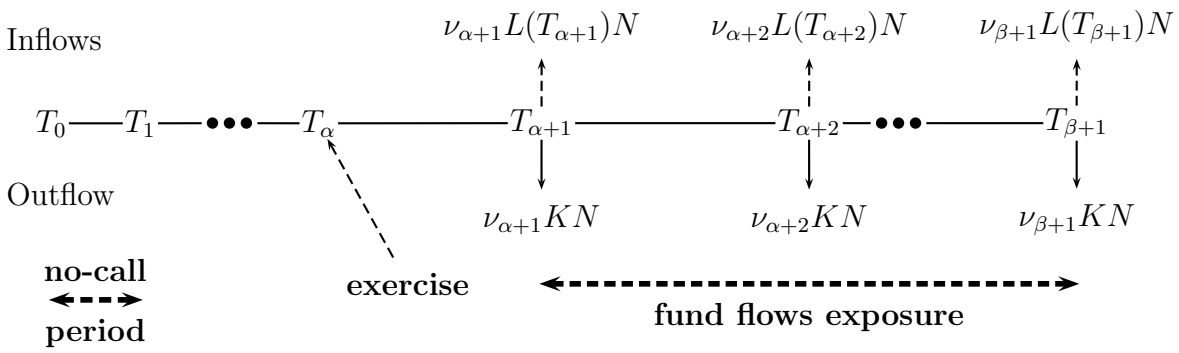


Figure 2.1.2: An example of a Bermudan payer swaption exercised at time T_α and associated fund flows exposure from the perspective of the holder of the option.

In the above, we have kept the description of Bermudan swaptions general enough so that it can be applied in the context of multi-currency markets, in which case the underlying swaps are multi-currency swaps, such as PRDC swaps, instead of single-currency fixed-for-floating swaps. Bermudan swaptions are, by far, one of the most actively traded exotic interest rate derivatives [4]. Demand for Bermudan swaptions comes from different segments of the interest rate markets. For example, Bermudan swaptions frequently arise as embedded options in interest rate swaps with Bermudan cancelable features, i.e. swaps that can be canceled by one party on a schedule of dates of the swaps' tenor structures. In this case, the other party of the swap can be viewed as selling to the first party a Bermudan-style option to cancel the underlying swaps in order to obtain a better rate of return in the form of a higher initial coupon during the no-call period. We shall discuss this point in more detail in Section 3.4 in the context of Bermudan cancelable PRDC swaps.

2.2 PRDC Swaps

In this section, we describe in detail the coupon structure, popular exotic variations, as well as the economics of PRDC swaps. Finally, we present a pricing model, an overview

of the calibration of the model, and derive the associated pricing PDE for PRDC swaps.

2.2.1 Introduction

Description of PRDC Swaps

A "vanilla" PRDC swap [64] is similar to a "vanilla" single-currency fixed-for-floating interest rate swap, in which both parties, namely the issuer and the investor, agree that the issuer pays the investor a stream of so-called PRDC coupon amounts, and in return, receives the investor's domestic LIBOR payments. Usually, the issuer of a PRDC swap is a bank. In analogy with a "vanilla" single-currency swap described in the previous section, the stream of PRDC coupons, referred to as the *PRDC coupon leg*, plays the role of the fixed leg, while the stream of the investor's domestic LIBOR payments plays the role of the floating leg. However, in a PRDC swap, the PRDC coupon amounts are linked to the spot FX rate prevailing when the PRDC coupon rate is set. Both the PRDC coupon and the floating rates are applied on the same domestic currency notional, denoted by N_d . Due to the dependence of the PRDC coupon amounts on the spot FX rates, PRDC swaps belong to the class of *FX interest rate hybrid derivatives*, referred to as hybrids. Another important distinguishing feature of PRDC swaps is that a swap's maturity is usually very long, often 30 years or more, while the maturity of a regular interest rate swap is usually much shorter, typically less than 5 years. Unless otherwise stated, we investigate PRDC swaps from the perspective of the issuer of the PRDC coupons. From this perspective, the investor's domestic LIBOR payments represent the stream of fund inflows, and hence, are usually referred to as the *funding leg*. We use this term throughout the thesis. Below, we first briefly discuss the funding leg, then describe the structure of the PRDC coupon rates in detail.

Given the tenor structure (2.4), 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. Similar to the floating leg in a "vanilla" single-currency fixed-for-floating interest rate swap, the funding leg pays the amount $\nu_\alpha L_d(T_\alpha)N_d$ at time T_α for the period $[T_{\alpha-1}, T_\alpha]$. Here, $L_d(T_\alpha)$ denotes the domestic LIBOR rate for the period $[T_{\alpha-1}, T_\alpha]$, as observed at time $T_{\alpha-1}$, and is defined by

$$L_d(T_\alpha) = \frac{1 - P_d(T_\alpha)}{\nu_\alpha P_d(T_\alpha)}, \tag{2.7}$$

which follows from (2.2).

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

$$C_\alpha = \min\Big(\max\Big(c_f \frac{s(T_\alpha)}{f_\alpha} - c_d, b_f\Big), b_c\Big), \tag{2.8}$$

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 $f_\alpha \equiv f(T_\alpha)$ is usually set to the forward FX rate $F(0, T_\alpha)$ defined by (2.5). All parameters c_d, c_f, b_f and b_c in (2.8) can vary from coupon to coupon, i.e. they may depend on $\{T_\alpha\}_{\alpha=1}^{\beta}$. However, to simplify the notation, we do not indicate the time-dependence of these parameters.

Remark 2.2.1. 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 on $T_{\beta+1}$. However, to be consistent with [58], we follow the same presentation.

A diagram of fund flows in a "vanilla" PRDC swap is presented in Figure 2.2.1.

Remark 2.2.2. 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

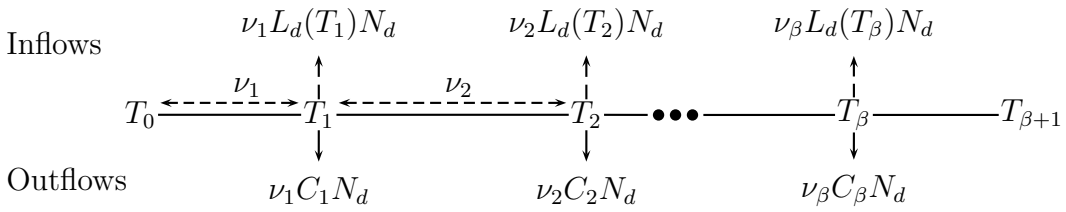


Figure 2.2.1: Fund flows in a "vanilla" PRDC swap. Inflows and outflows are with respect to the point-of-view of the PRDC coupon issuer, usually a bank.

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.

In the so-called *standard structure*, which is based on the most commonly used parameter settings, $b_f = 0$ and $b_c = \infty$, by letting $h_\alpha = \dfrac{c_f}{f_\alpha}$ and $e_\alpha = \dfrac{f_\alpha c_d}{c_f}$, the PRDC coupon rate C_α can be viewed as a call option on FX rates, since, in this case, C_α reduces to

$$C_\alpha = h_\alpha \max(s(T_\alpha) - e_\alpha, 0). \tag{2.9}$$

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 (2.9), the *option notional* h_α determines the overall level of the coupon payment, while the strike e_α determines the likelihood of the positiveness of the coupon. It is important to emphasize that, if the strike e_α is low compared to $s(T_\alpha)$, the PRDC 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 also, and hence, the overall level of the PRDC coupon amount paid at time T_α is small. This is a *low-leverage* situation, from the perspective of the investor. On the other hand, if both e_α and h_α are high, then we have a *high-leverage* situation. Typically, the levels of leverage could be modified by changing the sizes of c_d and c_f. We discuss in detail the impact of leverage levels on the prices of

PRDC swaps in Subsection 5.3.2.

Exotic Variations

It is important to emphasize that PRDC swaps usually have additional exotic features. Currently, the three most popular exotic features are Bermudan cancelable, knockout and FX-TARN. All three features allow, under different conditions, the pre-mature termination of the underlying swap after a no-call period, usually $[T_0, T_{1^-}]$.

Bermudan Cancelable PRDC Swaps

In a Bermudan cancelable PRDC swap, the issuer of the PRDC coupons has the right to cancel the PRDC swap at any of the dates $\{T_\alpha\}_{\alpha=1}^{\beta}$ of the swap's tenor structure *after* the exchange of fund flows scheduled on that date.

Knockout PRDC Swaps

An example of a knockout provision is an *up-and-out* FX-linked barrier: the associated PRDC swap pre-maturely terminates on the first date $T_\alpha, \alpha = 1, 2, \ldots, \beta$, of the tenor structure on which the spot FX rate $s(T_\alpha)$ exceeds a specified level. Different variations of the knockout feature may allow the termination of the PRDC swap to occur either before or after the occurrence of any exchange of fund flows scheduled on that date. However, knockout provisions usually stipulate that the knockout PRDC swap terminates *after* the exchange of fund flows scheduled on that date. Although the upper barrier can be time-dependent, i.e. a moving barrier, for simplicity, in this thesis, we consider a constant upper barrier only.

FX-TARN PRDC Swaps

In a FX-TARN PRDC swap, the PRDC coupon amount, $\nu_\alpha C_\alpha N_d$, is recorded. The PRDC swap is terminated on the first date $\{T_\alpha\}_{\alpha=1}^{\beta}$ when the accumulated PRDC coupon amount, including the coupon amount scheduled on that date, reaches a pre-determined target cap. A typical range of the target cap is from 10% to 50% of the notional N_d,

depending on the leverage levels.

Economics of PRDC Swaps

The rise of PRDC swaps as well as the continuing interest in these structured products are closely related to the search for yield enhancements by domestic currency investors when the interest rate for the domestic currency is low relative to the interest rate for the foreign currency. In this case, a yield enhancement opportunity in a PRDC swap for the domestic currency investor arises from the speculation that the forward FX rate $F(0,t)$ between the domestic and foreign currencies could be substantially lower than the respective spot FX rate $s(t)$, as the maturity t of the forward FX rate increases. More specifically, 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. United States Dollar (USD) or Australian Dollar (AUD)), the forward FX rate curve $F(0,t), t > 0$, decreases steeply as t increases, as reflected by formula (2.5), 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 (2.8): 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.

The exotic features, such as those described earlier, provide protection, from the perspective of the issuer, against excessive movements in the spot FX rate via a possibility of early termination of the swap. However, from the perspective of the investor, these exotic features can be viewed as an additional yield-enhancing mechanism which provides a higher rate of return in the form of a higher fixed rate coupon paid by the issuer to the investor during the no-call period, usually at time T_0. More specifically, in a PRDC swap with an exotic feature, the issuer can be viewed as "buying" from the investor

certain right to protect themselves against unfavorable movements in the spot FX rate. For instance, as explained later in Section 3.4, in a Bermudan cancelable PRDC swap, the issuer essentially buys from the investor a Bermudan option to cancel the underlying PRDC swap. As a result, a positive value (to the issuer) from such a position is generated and contributes to a higher positive initial fixed rate coupon at time T_0, i.e. a higher fund inflow for the investor at time T_0. This initial fixed rate coupon paid by the issuer to the investor is usually much higher than the rate of return that the investor can obtain anywhere else. In addition, the investor benefits even more from an exotic feature if the swap terminates quickly. For example, if the underlying PRDC swap is terminated at time T_1, the investor essentially pays a low domestic LIBOR payment $\nu_1 L_d(T_1)N_d$ and receives a very high initial fixed rate coupon on top of the PRDC coupon amount $\nu_1 C_1 N_d$. Thus, exotic features are very attractive to investors. We discuss this further in Subsection 5.3.2.

In the following subsections, we first present a pricing model and an overview of the model's calibration, and then derive the associated pricing PDE for PRDC swaps.

2.2.2 The Model

As cross-currency interest rate derivatives in general, and FX interest rate hybrids in particular, are exposed to moves in both the spot FX rate and the interest rate in both currencies, multi-factor pricing models having at least three factors must be used. The current standard practice for modeling FX interest rate hybrids is to use two one-factor Gaussian models for the two term structures and either a one-factor log-normal model (e.g., [38, 64]) or a one-factor skew model employing a local volatility function [58] for the spot FX rate. However, long-dated FX options, such as those embedded in PRDC swaps, exhibit a significant skew, which cannot be well captured by the log-normal distribution [24, 32, 58]. In addition, FX interest rate hybrids with exotic features, such as those

described earlier, are particularly sensitive to the FX volatility skew, due to the specific choice of strikes of the PRDC coupon rates, as well as the exotic features associated with the swaps (see, for example, [24, 58]). As a result, the assumption of log-normality of spot FX rates is questionable in the modeling of such derivatives. One way to rectify this deficiency is to incorporate FX volatility smiles into the model via a local volatility function, as first suggested in [58]. By using a local volatility model, this approach avoids introducing more stochastic factors into the model. Hence, it keeps the number of factors to the minimum, while providing better modeling for the skewness of the spot FX rate. Below, we present a multi-currency model with the FX volatility skew characterized by a local volatility function for the spot FX rate as proposed in [58]. We discuss the impact of the FX skew on the price of a PRDC swap with exotic features in Subsection 5.3.2.

We denote by $r_i(t), i = d, f$, the domestic and foreign short rates, respectively. Under the domestic risk-neutral measure, i.e. the measure with the domestic bank account as the numeraire[1], the dynamics of $s(t), r_d(t), r_f(t)$ are described by

$$\frac{ds(t)}{s(t)} = (r_d(t) - r_f(t))dt + \gamma(t, s(t))dW_s(t), \tag{2.10a}$$

$$dr_d(t) = (\theta_d(t) - \kappa_d(t)r_d(t))dt + \sigma_d(t)dW_d(t), \tag{2.10b}$$

$$dr_f(t) = (\theta_f(t) - \kappa_f(t)r_f(t) - \rho_{fs}(t)\sigma_f(t)\gamma(t, s(t)))dt + \sigma_f(t)dW_f(t), \tag{2.10c}$$

where $W_d(t)$, $W_f(t)$, and $W_s(t)$ are correlated Brownian motions with

$$dW_d(t)dW_s(t) = \rho_{ds}dt, \quad dW_f(t)dW_s(t) = \rho_{fs}dt, \quad dW_d(t)dW_f(t) = \rho_{df}dt.$$

The short rates follow the mean-reverting Hull-White model [31] 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 "quanto" drift adjustment, $-\rho_{fs}(t)\sigma_f(t)\gamma(t, s(t))$, for $dr_f(t)$ comes from changing the

[1]Intuitively, a numeraire is a positive non-dividend-paying reference asset, with respect to which all other asset prices are normalized. For a detailed discussion of measures and numeraires, see [63].

measure from the foreign risk-neutral measure to the domestic risk-neutral one. The local volatility function $\gamma(t, s(t))$ for the spot FX rate has the functional form [58]

$$\gamma(t, s(t)) = \xi(t)\Big(\frac{s(t)}{L_s(t)}\Big)^{\varsigma(t)-1}, \tag{2.11}$$

where $\xi(t)$ is the relative volatility function, $\varsigma(t)$ is the time-dependent constant elasticity of variance (CEV) parameter and $L_s(t)$ is a time-dependent scaling constant which is usually set to the forward FX rate $F(0,t)$, for convenience in calibration [58].

2.2.3 Calibration Overview

Before any model can be used, calibration of the model's parameters to specific market data is required. More specifically, the calibration of a model can be viewed as an estimation of the model's parameters using relevant market data. The calibration of the model (2.10) can be viewed as consisting of two separate tasks: (i) the calibration of the domestic and foreign short rate models (2.10b) and (2.10c), respectively, and (ii) the calibration of the local volatility function $\gamma(t, (s(t))$ in (2.10a). Note that the domestic and foreign short rate models are calibrated separately, and for convenience in calibration, usually under the respective risk-neutral measures. The correlation parameters ρ_{ds}, ρ_{fs} and ρ_{df} are typically chosen based on historical estimations. Since calibration of models is not the focus of this thesis, we only briefly discuss below the calibration of (i) and (ii).

Regarding the calibration of the domestic and foreign short rates, the parameters defining the volatility structures of interest rates in both currencies, i.e. the functions $\sigma_i(t), \kappa_i(t), i = d, f$, can be bootstrapped from European swaption market values for the respective currency[2], while $\theta_i(t)$, $i = d, f$, are determined by the current term structures of bond prices in the respective currency. More specifically, due to the analytical tractability of the one-factor Hull-White model, the Hull-White model's prices in the

[2]Since we are pricing swaps, it is natural to choose European swaptions for the calibration of the interest rate models.

respective currency for European swaptions are explicitly computable in terms of $\kappa_i(t)$ and $\sigma_i(t)$, $i = d, f$ [8]. The calibration for the one-factor Hull-White model for the domestic/foreign short rates to the respective European swaption market values can be posed as an optimization problem, in which we select values for the parameters $\kappa_i(t)$ and $\sigma_i(t)$ that best match the model's prices to the market prices in a least-squares sense. Popular algorithms, such as Gauss-Newton and Levenberg-Marquardt, can be used for such a least-squares optimization problem. Detailed discussions of these optimization techniques can be found in the literature, e.g. in [56]. Note that, in practice, we should only use the market prices of the most liquid swaptions, i.e. swaptions which are the most actively traded, since these prices are more reliable and carry more information about the current market situation than those of less liquid swaptions. Once $\sigma_i(t)$ and $\kappa_i(t)$ are obtained, $\theta_i(t)$ can be computed by using $\sigma_i(t)$, $\kappa_i(t)$ and the current term structures of bond prices in the respective currency (see Formula 3.34 in [8]).

While the methods to calibrate interest short rate models are well-known from single-currency interest rate modeling, the calibration of the local volatility function $\gamma(t, (s(t))$ is very challenging. Essentially, the local volatility function $\gamma(t, s(t))$ should be calibrated to FX options with different maturities and strikes traded on the market. One way to reduce the volume of data used in the calibration is to pick particular strikes and/or maturities of FX options relevant to the products we are pricing. Unfortunately, due to the long maturities of PRDC swaps and typical choice of strikes, FX options across a wide range of maturities and strikes must be used, a fact that makes the calibration of a local volatility in the context of PRDC swaps very cumbersome and time-consuming. In [58], an efficient calibration approach for the local volatility function $\gamma(t, s(t))$ based on the forward FX rate is proposed. This approach is built upon on the following two observations. First, under the domestic T-forward measure, i.e. the measure with $P_d(\cdot, T)$

as the numeraire [8], the forward FX rate $F(\cdot, T)$ is a martingale[3]. Hence, the dynamics of the forward FX rate $F(\cdot, T)$, which can be obtained by applying Itô's formula [63] to (2.5), admit a much simpler representation under the domestic T-forward measure than do the dynamics of the spot FX rate under the domestic risk-neutral measure (expressed by (2.10a)). Essentially, $F(t, T)$ follows a one-dimensional stochastic differential equation with no drift term, and with the diffusion coefficient given in terms of $\gamma(t, (s(t))$. Second, a European call option on the spot FX rate with maturity T, i.e. an option on $s(T)$, essentially a function of three spatial variables – the spot FX rate, the domestic and foreign interest rates, can be expressed as a European option on only the forward FX rate $F(T, T)$ under the domestic T-forward measure. This is a significant reduction in the complexity of the problem. Combined with the skew-averaging techniques, which essentially involve replacing time-dependent parameters with "effective" time-constant ones, this approach enables a fast calibration of the local volatility function $\gamma(t, (s(t))$. A more complete description of these calibration techniques can be found in [58].

2.2.4 The Associated PDE

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.

Theorem 2.2.1. *Let $u \equiv u(s, r_d, r_f, t)$ denote the domestic value function of a security with a terminal payoff measurable with respect to the σ-algebra at maturity time T_{end} and without intermediate payments. Furthermore, assume that $u \in \mathcal{C}^{2,1}$ on $\mathbb{R}_+ \times \mathbb{R} \times \mathbb{R} \times [T_{start}, T_{end})$, i.e. u is at least twice differentiable with respect to the space variables and differentiable with respect to the time variable. Then on $\mathbb{R}_+ \times \mathbb{R} \times \mathbb{R} \times [T_{start}, T_{end})$, u*

[3]A martingale is a zero-drift stochastic process. An introduction to martingales in the context of applications in finance can be found in [63]. More mathematically rigorous discussions of (continuous-time) martingales can be found in [41].

satisfies the PDE

$$\begin{aligned}\frac{\partial u}{\partial t}+\mathcal{L}_s u \equiv \frac{\partial u}{\partial t}+\frac{1}{2}\gamma^2(t,s(t))s^2\frac{\partial^2 u}{\partial s^2}+\frac{1}{2}\sigma_d^2(t)\frac{\partial^2 u}{\partial r_d^2}+\frac{1}{2}\sigma_f^2(t)\frac{\partial^2 u}{\partial r_f^2}+\rho_{ds}\sigma_d(t)\gamma(t,s(t))s\frac{\partial^2 u}{\partial s\partial r_d}\\ +\rho_{fs}\sigma_f(t)\gamma(t,s(t))s\frac{\partial^2 u}{\partial s\partial r_f}+\rho_{df}\sigma_d(t)\sigma_f(t)\frac{\partial^2 u}{\partial r_d\partial r_f}+(r_d-r_f)s\frac{\partial u}{\partial s}\\ +\Big(\theta_d(t)-\kappa_d(t)r_d\Big)\frac{\partial u}{\partial r_d}+\Big(\theta_f(t)-\kappa_f(t)r_f-\rho_{fs}\sigma_f(t)\gamma(t,s(t))\Big)\frac{\partial u}{\partial r_f}-r_d u=0.\end{aligned} \tag{2.12}$$

Proof. Under the domestic risk-neutral measure, i.e the measure with $B_d(t)$, the domestic money account, as the numeraire, the normalized price process of any security is a martingale. Since it is an Itô process, it must have zero drift. Calculating the drift term of the normalized pricing process $\frac{u(s,r_d,r_f,t)}{B_d(t)}$ using Itô's formula [63] and setting it to zero gives us the PDE (2.12). □

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. 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 (2.12) becomes

$$\frac{\partial u}{\partial \tau} = \mathcal{L}_s u \tag{2.13}$$

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

$$\{(s,r_d,r_f,\tau)|s\geq 0,-\infty<r_d<\infty,-\infty<r_f<\infty,\tau\in[0,T]\}, \tag{2.14}$$

where $T = T_{end}-T_{start}$. To solve the PDE (2.13) numerically by finite difference methods, we must truncate the unbounded domain into a finite-sized computational one

$$\{(s,r_d,r_f,\tau)\in[0,s_\infty]\times[0,r_{d,\infty}]\times[0,r_{f,\infty}]\times[0,T]\}\equiv\Omega_s\times[0,T], \tag{2.15}$$

where s_∞, $r_{d,\infty}$ and $r_{f,\infty}$ are sufficiently large [71].

Since payoffs and fund flows are deal-specific, we defer specifying the terminal conditions until Chapter 3. The difficulty with choosing boundary conditions is that, for an arbitrary payoff, they are not known. A detailed analysis on the boundary conditions is not a focus of this thesis, 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 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 [14].

Remark 2.2.3. It is known [4, 8] that, for Gaussian interest rate models, such as the one-factor Hull-White model, the instantaneous short rate is not guaranteed to be non-negative. More specifically, in the Hull-White model, there is a positive probability that the instantaneous short rate can be negative, which is clearly a drawback of this model. However, this probability is almost negligible in practice (see Subsection 3.3.1 of [8]). So, to obtain a bounded computational domain, we have set the lower boundaries of the computational domain to be zero in the r_d- and r_f-directions (see (2.15)). This is similar to how we truncate the computational domain at the upper boundaries, excluding unlikely events.

We conclude this chapter by noting that the Cox-Ingersoll-Ross (CIR) model [11, 12], which guarantees positive instantaneous short rates, can be used for the domestic and foreign short rates in the pricing model (2.10). The numerical methods developed in this thesis are also expected to work well in this case. It would be interesting to compare the effects of various choices for the interest short rate models on the prices of PRDC swaps. We plan to investigate this issue further in the future.

Chapter 3

Pricing PRDC Swaps

3.1 Introduction

For a structured interest rate derivative with multiple fund flows, such as a "vanilla" PRDC swap, its value at time T_0 is the sum of time T_0 values of individual fund flows scheduled at each time T_α, $\alpha = 0, 1, \ldots, \beta$, of the swap's tenor structure. When pricing such a derivative using a PDE approach, a general pricing principle is to progress backward in time from the time of the last fund flow (T_β in the case of a PRDC swap, for example) to the initial time T_0 of the tenor structure. In this approach, a model-dependent PDE, such as the PDE (2.12), is solved backward in time over each period $[T_{(\alpha-1)^+}, T_{\alpha^-}]$ of the tenor structure, i.e. from time $T_{end} = T_{\alpha^-}$ to time $T_{start} = T_{(\alpha-1)^+}$, with an appropriate terminal condition at time T_{α^-}. These terminal conditions usually include certain conditions to take into account possible fund flows at each time T_α. A general pricing framework for cross-currency swaps and swaptions can be found in [14].

In pricing a PRDC swap with exotic features, a similar pricing framework could be used. However, pricing such a PRDC swap is significantly more complex than pricing a "vanilla" PRDC swap, since, at each of the times $\{T_\alpha\}_{\alpha=1}^{\beta}$, in addition to multiple fund flows of the PRDC coupons and the funding payments, a possibility of early termination

of the underlying swap must be included. In addition, due to the very different natures of these exotic features, as well as their levels of suitability to a PDE-based pricing approach, different frameworks may need to be developed and special treatments may need to be employed for the pricing of such derivatives via a PDE approach.

The outline of this chapter is as follows. In Section 3.2, we present the discretization of the pricing PDE (2.13). We then discuss in detail PDE-based pricing algorithms for "vanilla" PRDC swaps in Section 3.3, and for PRDC swaps with exotic features, namely Bermudan cancelable, knockout and FX-TARN, in Sections 3.4, 3.5 and 3.6. Although in practice, combinations of these exotic features are available, we only present pricing algorithms for PRDC swaps with a single exotic feature. These algorithms can be used as the building blocks for developing pricing algorithms for PRDC swaps with combined exotic features.

3.2 Discretization

For the rest of the thesis, we adopt the following notation. Let the number of subintervals be $n+1$, $p+1$, $q+1$ and l in the s-, r_d-, r_f- and τ-directions, respectively. The uniform grid mesh widths in the respective directions are denoted by $\Delta s = \frac{s_\infty}{n+1}$, $\Delta r_d = \frac{r_{d,\infty}}{p+1}$, $\Delta r_f = \frac{r_{f,\infty}}{q+1}$ and $\Delta\tau = \frac{T}{l}$. Let the gridpoint values of a FD approximation to the solution u be 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 $i = 0, \ldots, n+1$, $j = 0, \ldots, p+1$, $k = 0, \ldots, q+1$, $m = 0, 1, \ldots, l$.

3.2.1 Space Discretization: Finite Difference Schemes

For the discretization of the space variables in the differential operator $\mathcal{L}_s$, we employ second-order central differences in the interior of the rectangular domain Ω_s. Second-

order FD approximations to the first and second partial derivatives of the space variables in (2.13) 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. For example, at the reference point $(s_i, r_{dj}, r_{fk}, \tau_m)$, the first and second partial derivatives with respect to the spot FX rate (i.e. $\dfrac{\partial u}{\partial s}$ and $\dfrac{\partial^2 u}{\partial s^2}$) are approximated by

$$\left.\frac{\partial u}{\partial s}\right|_{i,j,k}^{m} \approx \frac{u_{i+1,j,k}^{m} - u_{i-1,j,k}^{m}}{2\Delta s}, \tag{3.1a}$$

$$\left.\frac{\partial^2 u}{\partial s^2}\right|_{i,j,k}^{m} \approx \frac{u_{i+1,j,k}^{m} - 2u_{i,j,k}^{m} + u_{i-1,j,k}^{m}}{(\Delta s)^2}, \tag{3.1b}$$

while the cross derivative $\frac{\partial^2 u}{\partial s \partial r_d}$ is approximated by

$$\left.\frac{\partial^2 u}{\partial s \partial r_d}\right|_{i,j,k}^{m} \approx \frac{u_{i+1,j+1,k}^{m} + u_{i-1,j-1,k}^{m} - u_{i-1,j+1,k}^{m} - u_{i+1,j-1,k}^{m}}{4\Delta s \Delta r_d}, \tag{3.2}$$

which can be viewed as obtained by successively applying the FD scheme (3.1a) for the first derivatives in the s- and r_d-directions. Similar approximations can be obtained for the remaining spatial derivatives. It is important to note that, through Taylor expansions, it can be verified that each of the formulas (3.1) and (3.2) has a second-order truncation error, provided that the function u is sufficiently smooth. The derivations of these formulas can be found in Appendix C.1. An illustration of the computational stencil at the reference point $(s_i, r_{dj}, r_{fk}, \cdot)$ is presented in Figure 3.2.1

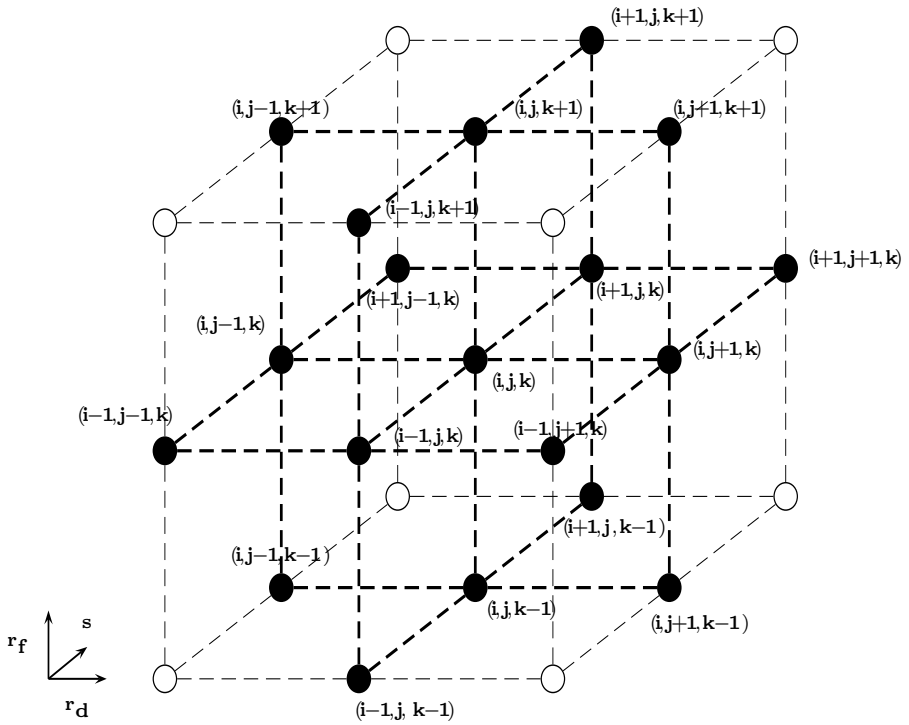


Figure 3.2.1: The spatial computational stencil at the reference gridpoint $(s_i, r_{dj}, r_{fk}, \cdot)$.

3.2.2 Time Discretization: ADI schemes

Several techniques, such as CN or ADI timestepping methods, can be used for the time discretization of the PDE (2.13). Let $\mathbf{u}^m$ denote the vector of values of the unknown price at time τ_m on the mesh Ω_s 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 $\mathcal{L}_s$ at τ_m. The standard explicit formula for $\mathbf{A}^m$ is presented in Appendix C.2. The CN method defines an approximation $\mathbf{u}^m$ to the true solution u^m, $m = 1, 2, \ldots, l+1$, by solving the linear system[1]

$$(\mathbf{I} - \frac{1}{2}\Delta\tau\mathbf{A}^m)\mathbf{u}^m = (\mathbf{I} + \frac{1}{2}\Delta\tau\mathbf{A}^{m-1})\mathbf{u}^{m-1} + \frac{1}{2}\Delta\tau(\mathbf{g}^m + \mathbf{g}^{m-1}), \tag{3.3}$$

[1]For a detailed description of the CN timestepping method, see relevant discussions in Subsection 6.3.3 in the context of multi-asset American options written on three assets.

where $\mathbf{I}$ denotes the identity matrix of the same size as $\mathbf{A}^m$. The vectors $\mathbf{g}^{m-1}$ and $\mathbf{g}^m$ are obtained from the boundary conditions. Applying direct methods, such as *LU* factorization, to solve this linear system can be computationally very expensive for several reasons: (i) the matrix $\mathbf{I}-\frac{1}{2}\Delta\tau\mathbf{A}^m$ possesses a bandwidth proportional to $\min\{np, nq, pq\}$, depending on the ordering of the gridpoints, (ii) sparse solvers suffer considerable fill-in when solving systems derived from PDEs of the form (2.13), and (iii) this matrix needs to be factored at each timestep because of its dependence on the timestep index m arising from time-dependent coefficients in the PDE (2.13). To avoid the high computational cost of direct methods, iterative methods, such as the Generalized Minimal Residual method (GMRES) or the Conjugate Gradient method, can be combined with a preconditioning technique, such as the Fast Fourier Transform (FFT) or an Incomplete LU factorization [25, 62], to solve (3.3). The reader is referred to Appendix D for a detailed discussion of this approach. An alternative approach is to apply an AF technique, in which the matrix $\mathbf{I} - \frac{1}{2}\Delta\tau\mathbf{A}^m$ is approximately factored into a product of three tridiagonal matrices. Hence, the computational cost is directly proportional to the number of gridpoints. We investigate this approach in the context of pricing multi-asset American options in Chapter 6 .

For the time discretization of the PDE (2.13), we employ the ADI timestepping technique. For this purpose, we decompose the matrix $\mathbf{A}^m$ into four submatrices:

$$\mathbf{A}^m = \mathbf{A}_0^m + \mathbf{A}_1^m + \mathbf{A}_2^m + \mathbf{A}_3^m.$$

The matrix $\mathbf{A}_0^m$ is the part of $\mathbf{A}$ that comes from the FD discretization of the cross derivative terms in (2.13), 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 $\mathcal{L}_s u$ is distributed evenly over $\mathbf{A}_1^m$, $\mathbf{A}_2^m$ and $\mathbf{A}_3^m$. The standard explicit formula for $\mathbf{A}_i^m$, $i = 0, 1, 2, 3$, is presented in Appendix C.2.

We consider the splitting scheme based on the Hundsdorfer and Verwer (HV) approach

[33, 36, 34], referred to henceforth as the *HV scheme*. 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[2]

$$\begin{cases} \mathbf{v}_0 = \mathbf{u}^{m-1} + \Delta\tau(\mathbf{A}^{m-1}\mathbf{u}^{m-1} + \mathbf{g}^{m-1}), & (3.4a) \\ (\mathbf{I} - \theta\Delta\tau\mathbf{A}_i^m)\mathbf{v}_i = \mathbf{v}_{i-1} - \theta\Delta\tau\mathbf{A}_i^{m-1}\mathbf{u}^{m-1} + \theta\Delta\tau(\mathbf{g}_i^m - \mathbf{g}_i^{m-1}), \quad i = 1,2,3, & (3.4b) \\ \widetilde{\mathbf{v}}_0 = \mathbf{v}_0 + \frac{1}{2}\Delta\tau(\mathbf{A}^m\mathbf{v}_3 - \mathbf{A}^{m-1}\mathbf{u}^{m-1}) + \frac{1}{2}\Delta\tau(\mathbf{g}^m - \mathbf{g}^{m-1}), & (3.4c) \\ (\mathbf{I} - \theta\Delta\tau\mathbf{A}_i^m)\widetilde{\mathbf{v}}_i = \widetilde{\mathbf{v}}_{i-1} - \theta\Delta\tau\mathbf{A}_i^m\mathbf{v}_3, \quad i = 1,2,3, & (3.4d) \\ \mathbf{u}^m = \widetilde{\mathbf{v}}_3. & (3.4e) \end{cases}$$

In (3.4), 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. An illustration of the tridiagonal solves in Steps (3.4b) and (3.4d) is given in Figure 3.2.2.

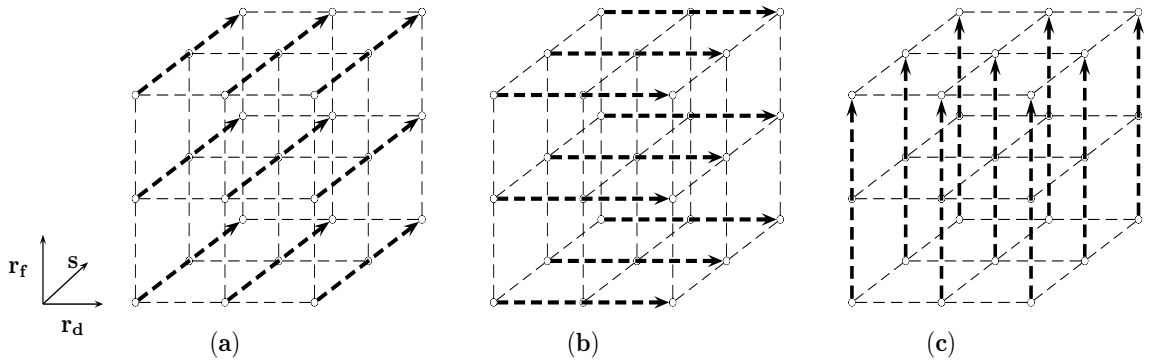


Figure 3.2.2: Tridiagonal solves along each spatial dimension in Steps (3.4b) and (3.4d): **(a)** along s when $i = 1$, **(b)** along r_d when $i = 2$, and **(c)** along along r_f when $i = 3$.

The parameter θ in (3.4) is directly related to the stability and accuracy of the ADI scheme. As mentioned in [33], a recommended range for θ is $\frac{1}{2} \leq \theta \leq 1$, where $\theta = \frac{1}{2}$ is the most accurate, but $\theta = 1$ gives more damping of the error terms arising from non-smooth initial conditions (i.e. the payoff function). In the experiments, for most cases,

[2]This is the scheme (1.4) in [36] with $\mu = \frac{1}{2}$.

we observe that $\theta = \frac{1}{2}$ works fine, except in the case of knockout PRDC swaps, where the payoff functions are discontinuous at each date of the tenor structure. In such a case, to take advantage of the damping property of the HV scheme when $\theta = 1$, we apply the HV scheme with $\theta = 1$ for the first few (usually two) initial timesteps and then switch to $\theta = \frac{1}{2}$ for the remaining timesteps, depending on the payoff functions. We refer to this timestepping technique as *HV smoothing*. We discuss the discontinuities in the payoffs of knockout PRDC swaps and the application of the HV smoothing technique in more detail in Section 3.5. We emphasize that choosing the parameter $\theta = 1$ gives a "partially" fully implicit timestepping method only, not a fully implicit one. Hence, HV smoothing is not the same as Rannacher smoothing [60], which initially uses a few (usually two or three) steps of fully implicit timestepping before switching to another timestepping method, such as CN. Note that the HV scheme has been proved to be unconditionally stable for arbitrary spatial dimension [36].

The HV splitting scheme treats the cross 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 first two lines of (3.4) can be viewed as an explicit Euler predictor step followed by three implicit, but unidirectional, corrector steps aiming to stabilize the predictor step. Several well-known ADI methods, such as the Douglas and Rachford method [15], are special instances of these two steps. The purpose of the additional stages that compute $\widetilde{\mathbf{v}}_i, i = 0, \ldots, 3$, is to restore second-order convergence for the general case with cross-derivatives, while retaining the unconditional stability of the scheme. The FD discretization for the spatial variables described in (3.1) 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 $\mathbf{A}_3^m$. See Appendix C.1.) 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 CN.

We conclude this section by noting that, from the point of view of designing a parallel algorithm, the block diagonal structure of the matrices $\mathbf{A}_i^m$, $i = 1, 2, 3$, gives rise to a natural, efficient parallelization for the solutions of the linear systems in Steps (3.4b) and (3.4d). However, it is less obvious how Steps (3.4a) and (3.4c) can be efficiently parallelized. We address this point in more detail in Chapter 4.

3.3 Vanilla PRDC Swaps

In this section, we consider the pricing of "vanilla" PRDC swaps. Let $u_\alpha^c(t)$ and $u_\alpha^f(t)$ be the values at time t of all PRDC coupons and funding 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 to the issuer, taking into account the fact that the PRDC coupons being paid by the issuer and the LIBOR payments being received by the issuer.

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 $\nu_\alpha L_d(T_\alpha)N_d$ at time $\{T_\alpha\}_{\alpha=1}^\beta$ to receiving and paying the amount N_d at times $T_{\alpha-1}$ and T_α, respectively. A description of this method is presented in Appendix E.1. Following this approach, the quantity $u_0^f(T_0)$ can be computed by

$$u_0^f(T_0) = (1 - P_d(T_0, T_\beta))N_d. \tag{3.5}$$

On the other hand, the PRDC coupon leg of a "vanilla" PRDC swap can be viewed as a portfolio of simple FX options with different maturities, i.e. with maturities $\{T_\alpha\}_{\alpha=1}^\beta$, and hence, the quantity $u_0^c(T_0)$ can be obtained by progressing backward in time from time T_β to time T_0, as specified in Algorithm 3.3.1.

Algorithm 3.3.1 Algorithm for computing the coupon part of "vanilla" PRDC swaps.

1: set $u^c_\beta(T_{\beta^+}) = 0$;

2: **for** $\alpha = \beta, \ldots, 1$ **do**

3: set
$$u^c_{\alpha-1}(T_{\alpha^-}) = u^c_\alpha(T_{\alpha^+}) + \nu_\alpha C_\alpha N_d; \tag{3.6}$$

4: solve the PDE (2.12) backward in time with the terminal condition (3.6) from T_{α^-} to $T_{(\alpha-1)^+}$ using the ADI scheme (3.4) for each time τ_m, $m = 1, \ldots, l$, to obtain $u^c_{\alpha-1}(T_{(\alpha-1)^+})$.

5: **end for**

6: set $u^c_0(T_0) = u^c_0(T_{0^+})$;

3.4 Bermudan Cancelable PRDC Swaps

As described in Section 2.2, in a Bermudan cancelable PRDC swap with the tenor structure (2.4), the issuer has an option to cancel the underlying swap at any of the dates $\{T_\alpha\}_{\alpha=1}^{\beta}$ after the exchange of fund flows scheduled on that date. Below, we first discuss a key observation in pricing Bermudan cancelable PRDC swaps, and then present a PDE-based pricing algorithm for these derivatives.

3.4.1 Key Observation

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, in our case, is a "vanilla" PRDC swap, and (ii) entering into the offsetting (opposite) swap at time T_α, i.e. a "vanilla" swap with the reversed fund flows at each of the remaining dates $\{T_{\alpha+1}, \ldots T_\beta\}$ of the tenor structure. An illustration of this observation is presented in Figure 3.4.1. It is obvious from Figure 3.4.1 that the total effect of continuing the underlying swap at time T_α and entering the opposite swap at time T_α is a zero net fund flow at each of the dates $\{T_{\alpha+1}, \ldots T_\beta\}$. This is exactly the same as canceling the swap

at time T_α.

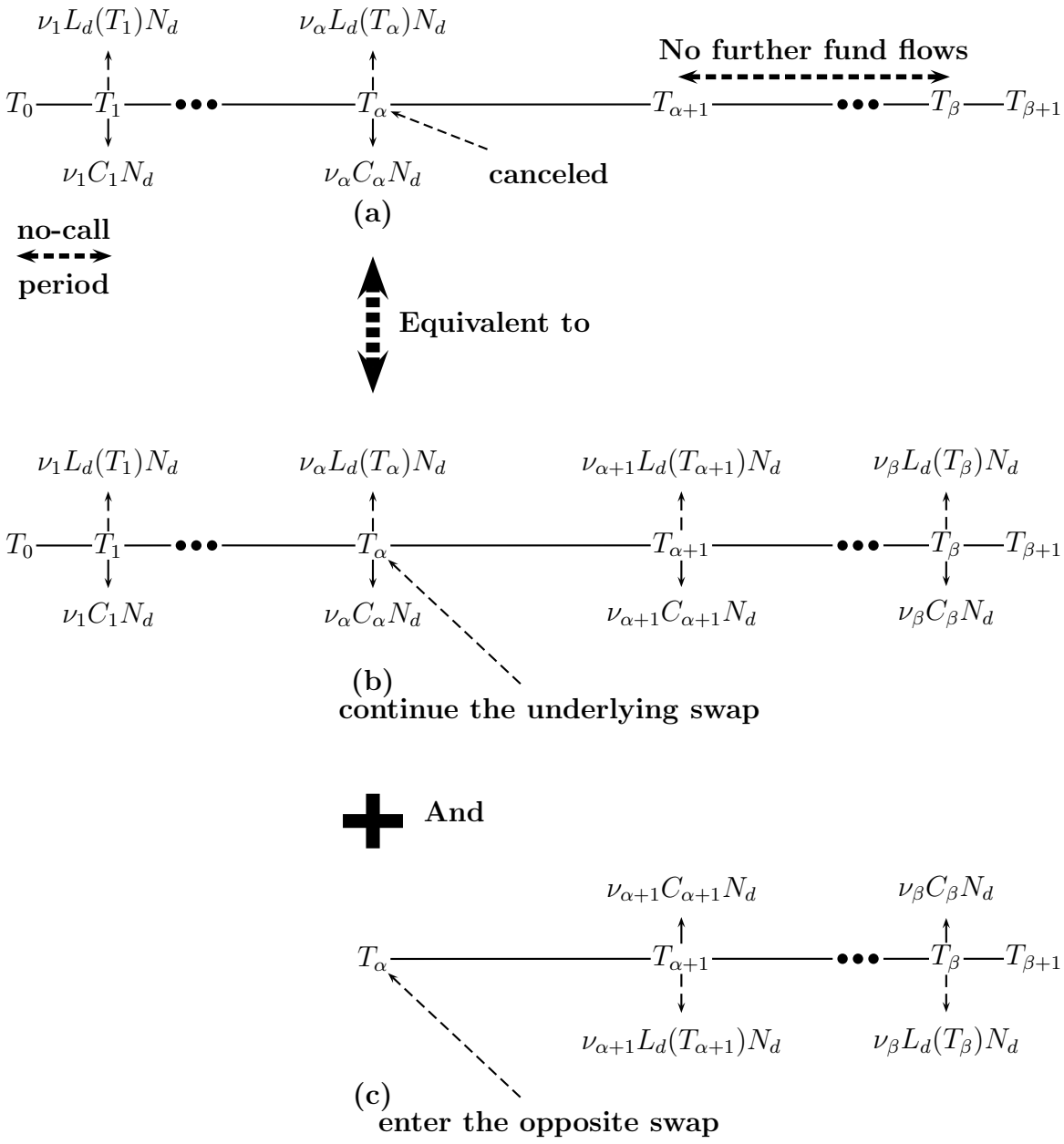


Figure 3.4.1: Key observation in pricing Bermudan cancelable PRDC swaps: canceling the swap at time T_α **(a)** is equivalent to continuing the swap **(b)** and entering the opposite swap at time T_α **(c)**.

3.4.2 A PDE Pricing Algorithm

Following the above observation, 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 its holder, i.e. the PRDC coupon issuer, the right, but not an obligation, to cancel the underlying swap, i.e. an option to enter the offsetting swap, at any of the dates $\{T_\alpha\}_{\alpha=1}^{\beta}$. This option is essentially a Bermudan swaption, as described on page 20, Subsection 2.1.2.

By the above argument, from the perspective of the issuer, 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. Subproblem (i) can be solved via Algorithm 3.3.1 described on page 42, Section 3.3. 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 $T_{\alpha+1}$. The quantity $u_\alpha^e(t)$ can be computed by

$$u_\alpha^e(t) = -(u_\alpha^f(t) - u_\alpha^c(t)), \tag{3.7}$$

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 $\{T_{\alpha+1}, \ldots, T_{\beta-1}\}$ as exercise opportunities, i.e. the option is still alive at time T_α. In particular, the quantity $u_0^h(T_0)$ is the value of the offsetting Bermudan swaption we are interested in at time T_0. If the option has not been exercised up to and including time T_α, then the value $u_{\alpha-1}^h(T_{\alpha^+})$ is equal to $u_\alpha^h(T_{\alpha^+})$ (the "hold value"). On the other hand,

if the option is exercised at time T_α, then the value $u^h_{\alpha-1}(T_{\alpha^+})$ is $u^e_\alpha(T_{\alpha^+})$ (the "exercise value"). 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 $u^e_\alpha(T_\alpha)$ exceeds the value $u^h_\alpha(T_\alpha)$. As a result, the condition for whether or not to exercise the offsetting Bermudan swaption at T_α is enforced by

$$u^h_{\alpha-1}(T_{\alpha^+}) = \max(u^h_\alpha(T_\alpha), u^e_\alpha(T_\alpha)).$$

The quantity $u^h_{\alpha-1}(T_{\alpha^+})$ is the payoff for the offsetting Bermudan swaption at each of $\{T_\alpha\}_{\alpha=1}^{\beta-1}$. A backward pricing algorithm for the associated offsetting Bermudan swaption is described in Algorithm 3.4.1. The value of the offsetting Bermudan swaption is $u^h_0(T_0)$, and the value of the Bermudan cancelable PRDC swap is $u^h_0(T_0) + (u^f_0(T_0) - u^c_0(T_0))$.

Algorithm 3.4.1 Algorithm for computing the offsetting Bermudan swaption.

1: set $u^h_\beta(T_{(\beta)^+}) = 0$ and $u^e_\beta(T_{(\beta)^+}) = 0$;

2: **for** $\alpha = \beta, \ldots, 1$ **do**

3: set $$u^h_{\alpha-1}(T_{\alpha^-}) = \max(u^h_\alpha(T_{\alpha^+}), u^e_\alpha(T_{\alpha^+})); \tag{3.8}$$

4: solve the PDE (2.12) backward in time with the terminal condition (3.8) from T_{α^-} to $T_{(\alpha-1)^+}$ using the ADI scheme (3.4) for each time τ_m, $m = 1, \ldots, l$, to obtain $u^h_{\alpha-1}(T_{(\alpha-1)^+})$;

5: compute $u^e_{\alpha-1}(T_{(\alpha-1)^+})$ by (3.7), where $u^c_{\alpha-1}(T_{(\alpha-1)^+})$ is computed from Line 4 of Algorithm 3.3.1, and $u^f_{\alpha-1}(T_{(\alpha-1)^+})$ is computed by the "fixed notional" method.

6: **end for**

7: set $u^h_0(T_0) = u^h_0(T_{0^+})$;

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 Chapter 4, 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.

3.5 Knockout PRDC Swaps

As described in Section 2.2, in a knockout PRDC swap with the tenor structure (2.4), at each of the dates $\{T_\alpha\}_{\alpha=1}^{\beta}$, a barrier condition, which stipulates that, after the exchange of fund flows scheduled on that date, the swap terminates if the spot FX rate $s(T_\alpha)$ is greater than the upper barrier, hereinafter denoted by B_u, must be enforced.

Let $u_\alpha^k(t)$ be the value at time t of a knockout PRDC swap that has $\{T_{\alpha+1}, \ldots, T_\beta\}$ as knockout opportunities, i.e. the swap is still alive at time T_α. In particular, the quantity $u_0^k(T_0)$ is the value of the knockout PRDC swap that we are interested in at time T_0. If the PRDC swap has not been knocked out up to and including time T_α, the value $u_{\alpha-1}^k(T_{\alpha^+})$ is equal to $u_\alpha(T_{\alpha^+})$. On the other hand, if $s(T_\alpha) > B_u$, i.e. the swap knocks out at time T_α, the quantity $u_{\alpha-1}^k(T_{\alpha^+})$ is zero. That is, the condition for a possibility of early termination of a knockout PRDC swap at each of the times $\{T_\alpha\}_{\alpha=1}^{\beta}$ is enforced by

$$u_{\alpha-1}^k(T_{\alpha^+}) = \begin{cases} 0 & \text{if } s(T_\alpha) > B_u, \\ u_\alpha^k(T_{\alpha^+}) & \text{otherwise.} \end{cases}$$

The quantity $u_{\alpha-1}^k(T_{\alpha^+})$ is part of the terminal condition for the solution of the PDE over the next period $[T_{(\alpha-1)^+}, T_{\alpha^-}]$, as described below.

We now consider the backward pricing algorithm for knockout PRDC swaps from time T_{α^-} to $T_{(\alpha-1)^+}$. One may attempt to start the backward algorithm at time T_{α^-} with the payoff

$$\bar{u}_{\alpha-1}^k(T_{\alpha^-}) \equiv u_{\alpha-1}^k(T_{\alpha^+}) + \nu_\alpha L_d(T_\alpha) N_d - \nu_\alpha C_\alpha N_d,$$

where $\nu_\alpha L_d(T_\alpha) N_d$ and $\nu_\alpha C_\alpha N_d$ are the funding payment and PRDC coupon amount scheduled at time T_α, respectively. Unfortunately, this is a path-dependent payoff, since

the LIBOR rate $L_d(T_\alpha)$ is determined at time $T_{\alpha-1}$, but the LIBOR payment takes place at time T_α. To overcome this difficulty, over each period of the swap's tenor structure, we consider the pricing of the funding leg and the PRDC coupon leg separately. More specifically, the value at time $T_{\alpha-1}$ of the funding payment scheduled at time T_α can be obtained via the "fixed notional" method. Following this approach, the value at time $T_{(\alpha-1)^+}$ of the funding payment scheduled on T_α is simply given by

$$(1 - P_d(T_\alpha))N_d. \tag{3.9}$$

On the other hand, the value at time $T_{(\alpha-1)^+}$ of the PRDC coupon $\nu_\alpha N_d C_\alpha$ is computed by solving the PDE (2.12). To this end, let $\bar{\bar{u}}_{\alpha-1}(T_{(\alpha-1)^+})$ be the value obtained by solving the PDE (2.12) backward in time from time T_{α^-} to time $T_{(\alpha-1)^+}$ with terminal condition

$$\bar{u}^k_{\alpha-1}(T_{\alpha^-}) \equiv u^k_{\alpha-1}(T_{\alpha^+}) - \nu_\alpha C_\alpha N_d.$$

The value of the knockout PRDC swap at time $T_{(\alpha-1)^+}$ is then given by

$$u_{\alpha-1}(T_{(\alpha-1)^+}) \equiv \bar{\bar{u}}_{\alpha-1}(T_{(\alpha-1)^+}) + (1 - P_d(T_\alpha))N_d.$$

A backward pricing algorithm for knockout PRDC swaps is presented in Algorithm 3.5.1.

Remark 3.5.1. It is important to note that, due to (3.10), the payoff (3.11) resembles that of a digital option. It is well-known that discontinuities in a digital-type payoff function can result in a reduction of the observed order of convergence of a numerical scheme [59]. In the context of option pricing, to restore the expected order of convergence, a remedy is to have the strike price positioned midway between the gridpoints [59, 66], a technique referred to as the *grid shifting technique*. We adopt this technique in our numerical method: the grids are chosen so that the fixed upper barrier B_u lies midway between the gridpoints in the spot FX rate, i.e. the s-direction. It is not necessary to have B_u as a midpoint of the grid in the r_d- and/or r_f-directions, since the digital

Algorithm 3.5.1 Algorithm for computing knockout PRDC swaps.

1: set $u^k_\beta(T_{\beta^+}) = 0$;

2: **for** $\alpha = \beta, \ldots, 1$ **do**

3: set

$$u^k_{\alpha-1}(T_{(\alpha)^+}) = \begin{cases} 0 & \text{if} \quad s(T_\alpha) > B_u, \\ u^k_\alpha(T_{(\alpha)^+}) & \text{otherwise;} \end{cases} \tag{3.10}$$

4: set

$$\bar{u}^k_{\alpha-1}(T_{\alpha^-}) = u^k_{\alpha-1}(T_{\alpha^+}) - \nu_\alpha N_d C_\alpha; \tag{3.11}$$

5: solve the PDE (2.12) with the terminal condition (3.11) backward in time from T_{α^-} to $T_{(\alpha-1)^+}$ using the ADI scheme (3.4) for each time τ_m, $m = 1, \ldots, l$, to obtain $\bar{\bar{u}}^k_{\alpha-1}(T_{(\alpha-1)^+})$;

6: set $u^k_{\alpha-1}(T_{(\alpha-1)^+}) = \bar{\bar{u}}^k_{\alpha-1}(T_{(\alpha-1)^+}) + (1 - P_d(T_\alpha))N_d$;

7: **end for**

8: set $u^k_0(T_0) = u^k_0(T_{0^+})$;

condition of the payoff function (3.10) depends only on the spot FX rate $s(t)$. Although other techniques for smoothing the discontinuities in the initial data, such as averaging and projection methods [59], can be used, we adopted the grid shifting technique for our experiments due to its simplicity and effectiveness. In addition, it is worth pointing out that, with the discontinuities in the payoff functions being introduced at each of the times $\{T_\alpha\}_{\alpha=1}^{\beta}$, in our experiments, we apply the HV smoothing technique for each of the dates $\{T_\alpha\}_{\alpha=1}^{\beta}$ of the tenor structure when knockouts are possible. This is similar to the techniques discussed in [6] in the context of discrete barrier options. Our numerical results presented in Chapter 5 show that this technique provides good damping and works well for PRDC swaps with a knockout provision.

3.6 FX-TARN PRDC Swaps

In a PRDC swap with a FX-TARN feature, the sum of all PRDC coupon amounts $\nu_\alpha C_\alpha N_d$ paid to date is recorded, and the 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 the pre-determined target cap, hereinafter denoted by A_c. A diagram illustrating the dynamics of FX-TARN PRDC swaps is presented in Figure 3.6.1.

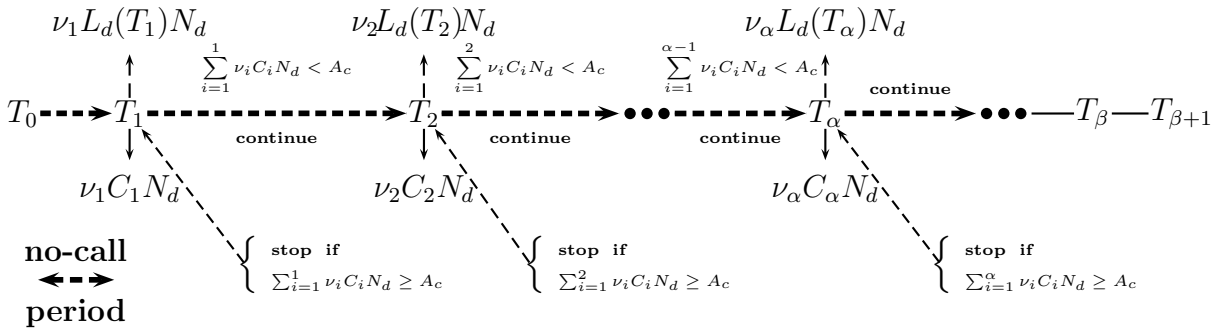


Figure 3.6.1: Fund flows and possibilities of pre-mature termination in a FX-TARN PRDC swap.

The fluctuations in the spot FX rate lead to uncertainty in how much the PRDC coupon amounts will be on each date of the tenor structure, and, in turn, uncertainty in whether and when the PRDC swap will be pre-maturely terminated. The uncertainty of the early termination date is governed by a path-dependent variable, the running accumulated PRDC coupon sum. Due to the path-dependence of the TARN feature, Monte-Carlo simulation is currently the only method used for the valuation of cross-currency/FX interest rate hybrids with this feature.

We observe a similarity between the TARN feature of a PRDC swap and the knock-out feature of an Asian barrier option which is governed by the average asset value [75]. Following [75], our PDE pricing approach for FX-TARN PRDC swaps is based on an auxiliary path dependent state variable, hereinafter denoted by $a(t)$, $0 \leq a(t) < A_c$, which represents the accumulated PRDC coupon amount. This variable stays constant between dates of the swap's tenor structure and is updated on each date of the tenor structure to reflect the PRDC coupon amount known on that date. It can be used to determine the pre-mature termination of the underlying swap on that date.

Due to the fact that the early termination date of a FX-TARN PRDC swap is governed by the path-dependent variable $a(t)$, the main challenge in pricing a FX-TARN PRDC swap using a PDE approach is to handle efficiently the path-dependency of the problem. It is worth emphasizing that MC simulation is well-suited for path-dependent financial problems, such as those arising in Asian option pricing, since MC simulation computes the paths forward in time. On the other hand, the PDE approach is a natural choice for pricing financial derivatives with early exercise features, such as American-style options, since these features are most easily handled if the solution is computed backward in time from the maturity date to the start date. For example, we have seen in Section (3.4) that a PDE approach can naturally handle the pricing of a Bermudan swaption which is an American-style derivative with multiple, albeit discrete, exercise opportunities. Below, we first introduce further notation, then discuss the updating rules at dates of the swaps' tenor structure, and a key observation in pricing FX-TARN PRDC swaps. Finally, we present a PDE pricing algorithm for these products based on the auxiliary variable $a(t)$.

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 $a(t)$. We denote by $u \equiv u(s, r_d, r_f, t; a)$ the domestic value function of a FX-TARN PRDC swap. For presentation

purposes, we further adopt the following notation

$$a_{\alpha^+} \equiv a(T_{\alpha^+}); \quad a_{\alpha^-} \equiv a(T_{\alpha^-}); \quad u_{\alpha^+} \equiv u(s, r_d, r_f, T_{\alpha^+}; a_{\alpha^+}); \quad u_{\alpha^-} \equiv u(s, r_d, r_f, T_{\alpha^-}; a_{\alpha^-}).$$

Since a is constant between dates of the tenor structure, we have $a_{(\alpha-1)^+} = a_{\alpha^-}$.

3.6.1 Updating Rules

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 $a(t)$. More specifically, apart from dates $\{T_\alpha\}_{\alpha=1}^{\beta}$, for any fixed value of a, the function u satisfies the model-dependent PDE (2.12). On each of the dates $\{T_\alpha\}_{\alpha=1}^{\beta}$, assuming that $a_{\alpha^-} < A_c$, i.e. the swap is still alive at time T_{α^-}, the quantity a changes according to the updating rule

$$a_{\alpha^+} = a_{\alpha^-} + \min(A_c - a_{\alpha^-}, \nu_\alpha C_\alpha N_d) \equiv a_{(\alpha-1)^+} + \min(A_c - a_{\alpha^-}, \nu_\alpha C_\alpha N_d), \tag{3.12}$$

taking into account the fact that $a_{\alpha^-} = a_{(\alpha-1)^+}$. The quantity $\min(A_c - a_{\alpha^-}, \nu_\alpha C_\alpha N_d)$ in (3.12) is the actual PRDC coupon amount paid at T_α, taking into account the target cap A_c for the total coupon amount. When $a_{\alpha^+} = A_c$, the swap terminates. By no-arbitrage arguments, across each date $\{T_\alpha\}_{\alpha=1}^{\beta}$, u must satisfy the updating rule

$$u_{\alpha^+} = u_{\alpha^-} + \nu_\alpha L_d(T_\alpha) N_d - \min(A_c - a_{\alpha^-}, \nu_\alpha C_\alpha N_d). \tag{3.13}$$

A diagram illustrating the updating rules (3.12) and (3.13) is presented in Figure 3.6.2.

3.6.2 Key Observation

In the context of pricing interest rate swaps via a PDE approach in general, the purpose of the backward procedure from the last date of exchange of fund flows (e.g. T_β in our case) to the date $T_{(\alpha-1)^+}$, $\alpha = \beta, \ldots 1$, is essentially to compute the value at time $T_{(\alpha-1)^+}$

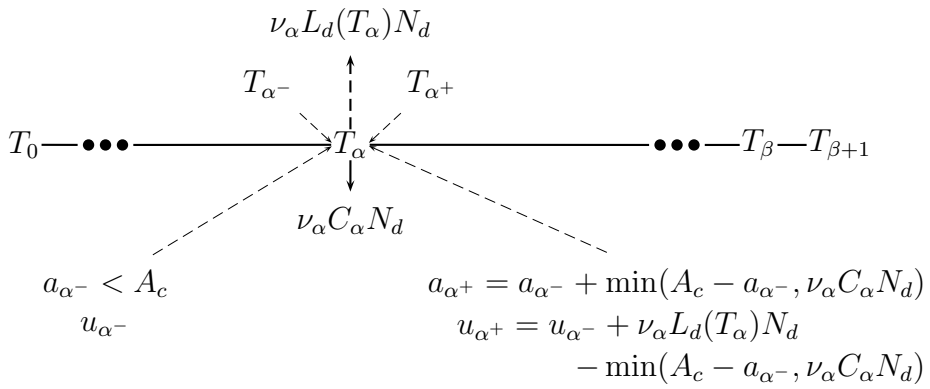


Figure 3.6.2: Updating rules in a FX-TARN PRDC swap.

of all the fund flows scheduled on or after T_α in the swaps' tenor structure. In the context of FX-TARN PRDC swaps, if the swaps are pre-maturely terminated by the time $T_{(\alpha-1)^+}$, there are no further fund flows scheduled on or after T_α, and hence, this value is zero. That is, over each period $[T_{(\alpha-1)^+}, T_{\alpha^-}]$ of the swaps' tenor structure, the backward procedure which computes the solution backward in time from T_{α^-} to $T_{(\alpha-1)^+}$ needs to be concerned only with swaps that are still alive at time $T_{(\alpha-1)^+}$, i.e. swaps that have $0 \leq a_{(\alpha-1)^+} < A_c$. 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 for other variables. This key observation leads to the following general PDE pricing framework for a FX-TARN PRDC swap:

(i) Across each date of $\{T_\alpha\}_{\alpha=\beta}^1$, for <u>each</u> discretized value of the variable a, apply the updating rules (3.12-3.13) on the swap values to

   (a) take into account the fund flows scheduled on that date;

   (b) reflect changes in the accumulated PRDC coupon amount, and the possibility of early termination;

   (c) obtain terminal conditions for the solution of the PDE from time T_{α^-} to $T_{(\alpha-1)^+}$ (see Step (ii) below);

(ii) Over each period $[T_{(\alpha-1)^+}, T_{\alpha^-}]$, $\alpha = \beta, \ldots, 1$, of the swap's tenor structure, for each discretized value of the variable a, solve the model PDE (2.12) backward in time from T_{α^-} to $T_{(\alpha-1)^+}$, with the corresponding terminal condition obtained in Step (i).

Below, we describe in detail a PDE-based pricing algorithm for FX-TARN PRDC swaps.

3.6.3 A PDE Pricing Algorithm

We further adopt the following notation. Let the interval $[0, A_c]$, for a given number of subintervals $w+1$, be partitioned by

$$0 = a_0 < a_1 < \ldots < a_w < a_{w+1} = A_c. \tag{3.14}$$

Note that, for all periods of the swap's tenor structure, we have the fixed set of gridpoints (3.14) in the a-direction. 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.

If a FX-TARN PRDC swap has not been pre-maturely terminated by time T_α, i.e. $a_{\alpha^+} < A_c$, the value $u_{\alpha-1}(T_{\alpha^+}; a_{(\alpha-1)^+})$ is given by

$$u_{\alpha-1}(T_{\alpha^+}; a_{(\alpha-1)^+}) = u_\alpha(T_{\alpha^+}; a_{\alpha^+}) \equiv u_\alpha(T_{\alpha^+}; a_{(\alpha-1)^+} + \min(A_c - a_{(\alpha-1)^+}, \nu_\alpha C_\alpha N_d)),$$

according to the updating rule (3.12). On the other hand, if the swap is terminated at time T_α, we then have $u_{\alpha-1}(T_{\alpha^+}; a_{(\alpha-1)^+})$ is equal to zero. That is, the condition for a possibility early termination of a FX-TARN PRDC swap at each of the times $\{T_\alpha\}_{\alpha=1}^{\beta}$ is enforced by

$$u_{\alpha-1}(T_{\alpha^+}; a_{(\alpha-1)^+}) = \begin{cases} 0 & \text{if } a_{\alpha^+} \geq A_c, \\ u_\alpha(T_{\alpha^+}; a_{\alpha^+}) & \text{otherwise,} \end{cases} \tag{3.15}$$

where $a_{\alpha^+} = a_{(\alpha-1)^+} + \min(A_c - a_{(\alpha-1)^+}, \nu_\alpha C_\alpha N_d)$.

One may attempt to start the backward algorithm at time T_{α^-} with the payoff

$$u_{\alpha-1}(T_{\alpha^+}; a_{(\alpha-1)^+}) + \nu_\alpha L_d(T_\alpha) N_d - \nu_\alpha C_\alpha N_d.$$

However, there are several issues with this payoff. First of all, this is a path-dependent payoff, similar to those arising in pricing knockout PRDC swaps (see Section 3.5). To overcome this difficulty, over each period of the tenor structure, we can consider the valuations of the funding payment and the PRDC coupon parts separately, as we do when pricing knockout PRDC swaps. The second issue arises from the fact that the set of gridpoints in the a-direction is fixed, i.e. $a_{(\alpha-1)^+}$ is typically a gridpoint of (3.14) excluding A_c, and hence, the quantity $a_{\alpha^+} = a_{(\alpha-1)^+} + \min(A_c - a_{(\alpha-1)^+}, \nu_\alpha C_\alpha N_d)$ in (3.15) may not be a gridpoint in the a-direction, i.e. not a gridpoint of (3.14). As a result, the value

$$u_\alpha(T_{\alpha^+}; a_{\alpha^+}) \equiv u_\alpha(T_{\alpha^+}; a_{(\alpha-1)^+} + \min(A_c - a_{(\alpha-1)^+}, \nu_\alpha C_\alpha N_d))$$

of (3.15) may not be immediately available. Below, we illustrate how to enforce (3.15) using only the fixed set of gridpoints (3.14) and discuss the backward procedure for FX-TARN PRDC swaps from time T_{α^-} to $T_{(\alpha-1)^+}$.

Assume that $u_\alpha(T_{\alpha^+}; a_y)$, $y = 0, \ldots, w$, are computed at the previous period of the tenor structure, i.e. these are available at T_{α^+}. For each a_y, $y = 0, \ldots, w$, we first find the corresponding quantity $\bar{a}_y$ specified by

$$\bar{a}_y = a_y + \min(A_c - a_y, \nu_\alpha C_\alpha N_d).$$

Note that the quantity $\bar{a}_y$ depends on T_α, but, to simplify the notation, we do not indicate its time dependence. We then find $u_{\alpha-1}(T_{\alpha^+}; \bar{a}_y)$ by using $u_\alpha(T_{\alpha^+}; a_y)$, $y = 0, \ldots, w+1$. More specifically, if $\bar{a}_y \geq A_c$, i.e. the swap is terminated pre-maturely at time T_α, and hence, $u_{\alpha-1}(T_{\alpha^+}; \bar{a}_y)$ is zero. On the other hand, if $\bar{a}_y < A_c$, the swap is not pre-maturely

terminated at time T_α. In this case, $\bar{a}_y$ may fall between the computational grid points in the a-direction, i.e. $a_{\bar{y}} \leq \bar{a}_y \leq a_{\bar{y}+1}$, for some $\bar{y}$ in $\{0, \ldots, w\}$. To approximate $u_{\alpha-1}(T_{\alpha^+}; \bar{a}_y)$, we apply linear interpolation [75]

$$u_{\alpha-1}(T_{\alpha^+}; \bar{a}_y) = \frac{\bar{a}_y - a_{\bar{y}}}{a_{\bar{y}+1} - a_{\bar{y}}} u_\alpha(T_{\alpha^+}; a_{\bar{y}+1}) + \frac{a_{\bar{y}+1} - \bar{a}_y}{a_{\bar{y}+1} - a_{\bar{y}}} u_\alpha(T_{\alpha^+}; a_{\bar{y}}). \tag{3.16}$$

Note that, in the special case that $\bar{y} = w$, we set $u_\alpha(T_{\alpha^+}; a_{\bar{y}+1}) \equiv u_\alpha(T_{\alpha^+}; A_c) = 0$. The above procedure essentially enforces (3.15), within the accuracy of linear interpolation.

Regarding the backward procedure, we first take into account the PRDC coupon payment by computing

$$\hat{u}_{\alpha-1}(T_{\alpha^-}; a_y) = u_{\alpha-1}(T_{\alpha^+}; \bar{a}_y) - \min(A_c - a_y, \nu_\alpha C_\alpha N_d),$$

which becomes the terminal condition for the PDE (2.12). We then solve this PDE backward in time from T_{α^-} to $T_{(\alpha-1)^+}$ using the the ADI scheme (3.4) for each time τ_m, $m = 1, \ldots, l$, to obtain $\hat{\hat{u}}_{\alpha-1}(T_{(\alpha-1)^+}; a_y)$. Finally, we incorporate the funding leg payment by computing

$$u_{\alpha-1}(T_{(\alpha-1)^+}; a_y) = \hat{\hat{u}}_{\alpha-1}(T_{(\alpha-1)^+}; a_y) + (1 - P_d(T_\alpha))N_d.$$

A backward pricing algorithm for FX-TARN PRDC swaps is presented in Algorithm 3.6.1.

We conclude this section by noting that, from the point of view of designing a parallel algorithm, by dividing the pricing of FX-TARN PRDC swaps into multiple independent pricing subproblems during each period of the tenor structure, we can run these pricing processes, each for a subproblem, in parallel with communication only at $\{T_\alpha\}_{\alpha=1}^{\beta-1}$ in order to carry out the interpolation (3.16) (also see Remark 3.6.1 on page 55). In Chapter 4, we describe in more detail how this approach is very well-suited for the parallel pricing of FX-TARN PRDC swaps on a cluster of GPUs.

Remark 3.6.1. Since $a_y \leq \bar{a}_y$, for the interpolation scheme (3.16), the process associated with a_y would possibly need, besides its own data $u_\alpha(T_{\alpha^+}; a_y)$, some data from some

pricing processes associated with $a_{y+1} \dots a_w$, i.e. $u_\alpha(T_{\alpha^+}; a_{y+1}), \dots u_\alpha(T_{\alpha^+}; a_w)$, but not the other way around. For example, the process associated with a_0 would potentially need data from all other processes (a total of w processes), while the process associated with a_w do not require any data from any other process. Figure 3.6.3 gives a pictorial illustration of this observation.

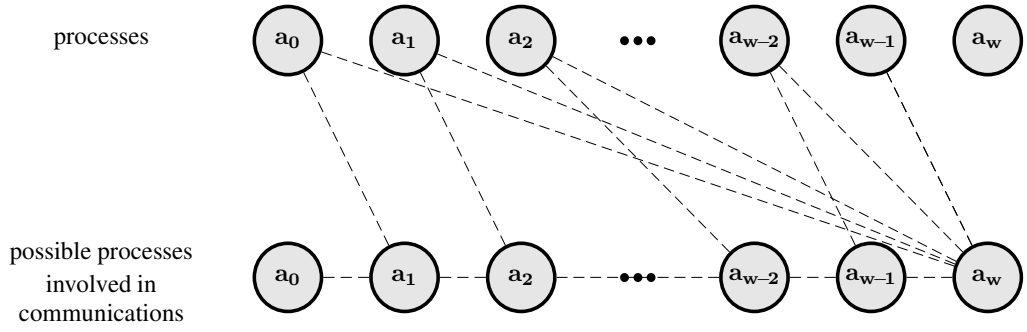


Figure 3.6.3: Possible required communications for interpolations between pricing processes.

Algorithm 3.6.1 Backward algorithm for computing FX-TARN PRDC swaps.

1: set $u_\beta(T_{\beta^+}; a_y) = 0$, $y = 0, \ldots, w$;

2: **for** $\alpha = \beta, \ldots, 1$ **do**

3: **for** each a_y, $y = 0, \ldots, w$, **do**

4: set

$$\bar{a}_y = a_y + \min(A_c - a_y, \nu_\alpha C_\alpha N_d); \tag{3.17}$$

5: set

$$u_{\alpha-1}(T_{\alpha^+}; \bar{a}_y) = \begin{cases} 0 & \text{if } \bar{a}_y \geq A_c \\ \dfrac{\bar{a}_y - a_{\bar{y}}}{a_{\bar{y}+1} - a_{\bar{y}}} u_\alpha(T_{\alpha^+}; a_{\bar{y}+1}) & \\ \qquad + \dfrac{a_{\bar{y}+1} - \bar{a}_y}{a_{\bar{y}+1} - a_{\bar{y}}} u_\alpha(T_{\alpha^+}; a_{\bar{y}}) & \text{if } a_{\bar{y}} \leq \bar{a}_y \leq a_{\bar{y}+1}, \\ & \bar{y} \in \{0, \ldots, w\}. \end{cases} \tag{3.18}$$

6: set

$$\hat{u}_{\alpha-1}(T_{\alpha^-}; a_y) = u_{\alpha-1}(T_{\alpha^+}; \bar{a}_y) - \min(A_c - a_y, \nu_\alpha C_\alpha N_d), \tag{3.19}$$

7: solve the PDE (2.12) with the terminal condition (3.19) from T_{α^-} to $T_{(\alpha-1)^+}$ using the ADI scheme (3.4) for each time τ_m, $m = 1, \ldots, l$, to obtain $\hat{\hat{u}}_{\alpha-1}(T_{(\alpha-1)^+}; a_y)$;

8: set

$$u_{\alpha-1}(T_{(\alpha-1)^+}; a_y) = \hat{\hat{u}}_{\alpha-1}(T_{(\alpha-1)^+}; a_y) + (1 - P_d(T_\alpha))N_d; \tag{3.20}$$

9: **end for**

10: **end for**

11: set $u_0(T_0; 0) = u_0(T_{0^+}; 0)$;

Chapter 4

Efficient Implementation on GPUs

The focus of this chapter is on the detailed description of an efficient implementation on GPUs of the pricing algorithms presented in the previous chapter. This chapter is organized as follows. In Section 4.1, we first summarize some key properties of the GPU device architecture and the CUDA Application Programming Interface (API) necessary to understand our implementation on GPUs of the pricing algorithms. We then provide details about the GPU cluster that we used for the experiments. In Section 4.2, we present general frameworks of our pricing algorithms for PRDC swaps on multi-GPU platforms/clusters of GPUs. A GPU-based implementation of the ADI timestepping scheme (3.4) is presented in Section 4.3. An implementation of the GPU-based interpolation technique used for FX-TARN PRDC swaps is discussed in Section 4.4. Section 4.5 provides details about other related implementations. Although the focus of the implementation of the numerical methods for pricing FX-TARN PRDC swaps is on clusters of GPUs, in Section 4.6, we briefly present an implementation for pricing FX-TARN PRDC swaps on a single GPU.

4.1 Background

4.1.1 GPU Device Architecture

Although the focus in this subsection is on NVIDIA products, offerings from other GPU manufacturers, such as ATI, are similar. This section is intended as a short overview of the basic concepts only. More detailed descriptions of the architecture and the programming model can be found, for example, in [45, 57].

The modern GPU can be viewed as a set of independent streaming multiprocessors (SMs) [57]. One such SM contains, amongst other things, several scalar processors which can execute integer and single- and double-precision floating-point arithmetic, several registers, a multi-threaded instruction unit (I/U), and shared memory. The latest GPUs, such as those based on the "Fermi" architecture, also have L1/L2 caches. A graphical illustration of the layout of a GPU is given in Figure 4.1.1. The shared memory can be accessed by all scalar processors of a multiprocessor, while the registers have processor scope. The device (or global) memory can be accessed by all processors on the chip. Furthermore, constant cache, a small part of the device memory dedicated to storing constants, is also available. Note that the constant cache is read-only and has faster access than the shared memory.

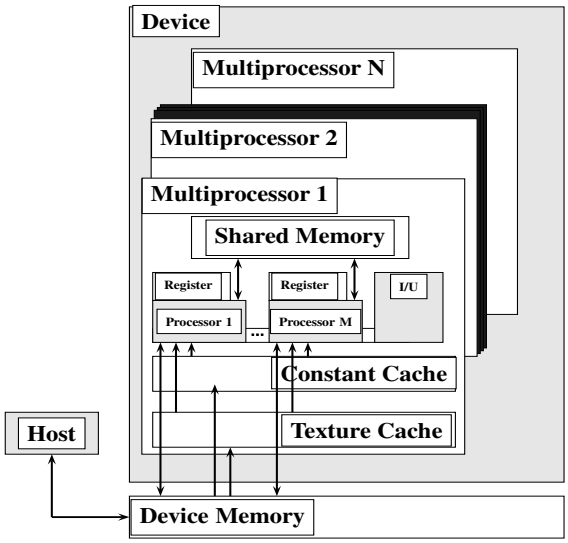


Figure 4.1.1: Architectural visualization of a GPU device and memory [57].

Typically, the CPU (the *host*) runs the program skeleton and offloads the more com-

putationally demanding code sections to the GPUs (the *device*). Functions that run on the device are called *kernel functions* or simply *kernels*. When a C program using CUDA extensions and running on the CPU 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*. A grid of threadblocks could be one- or two-dimensional, with up to 65,535 blocks in each dimension. The total size of a threadblock is limited to 512 threads, with the flexibility of distributing these elements into one-, two-, or three-dimensional arrays, depending on the problem being solved. The CUDA framework assigns unique coordinates, referred to as *threadId* and *blockId* to each thread and each threadblock, respectively, which are accessible in standard C language via built-in variables. From the programmer's point-of-view, the main functionality of blockIds and threadIds is to provide threads with a means to distinguish among themselves when executing the same kernels. This makes it possible for each thread to identify its share of work in a SIMD application. An example of grids of threadblocks and associated threadIDs and blockIDs is given in Figure 4.1.2. In this example, the first kernel (Kernel 1) generates a 2×3 grid of threadblocks, and each of the threadblocks consists of 15 threads arranged on a 3×5 array. In practice, each CUDA grid typically comprises thousands to millions of GPU threads per kernel invocation.

Threads in a threadblock are executed by processors within a single SM. One or more threadblocks may be assigned to the same SM at a time, depending on the available resources and the size of each threadblock. All threads in a threadblock can read from and write to any shared memory location assigned to that threadblock. Consequently, threads

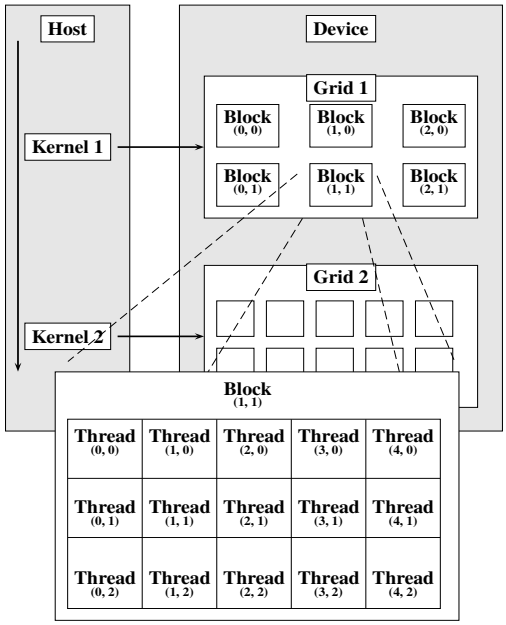


Figure 4.1.2: An example of grids of threadblocks and associated threadIds and blockIds.

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, even when the two threadblocks are assigned to the same SM.

Within the CUDA framework, threadblocks can execute in any order relative to each other, which allows transparent scalability in the parallelism of CUDA kernels. Regarding the execution timing of threads within each threadblock, the correctness of executing a kernel should be independent of whether certain threads execute in synchrony with each other. However, due to various hardware considerations, the current generation of CUDA devices actually does bundle multiple threads for execution 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 will be the sum of the processing times for all the branches. This must be avoided whenever possible. We return to this point in the next section, where we discuss memory coalescing.

4.1.2 Details of the GPU Cluster

All of the experiments in this thesis, including those for multi-asset options presented in Chapter 7, were carried out on the Shared Hierarchical Academic Research Computing Network (SHARCNET) GPU cluster "Angel". We used the cluster for both the CPU and GPU code development and testing. The cluster has 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. 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. Each pair of the T10 GPUs is attached to a node via a PCI Express 2.0x16 link. As such, there is a T10 GPU per quad-core Xeon processor, or a total of two T10 GPUs per node.

- 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.

It is important to note that, in practice, the number of nodes available for the experiments varies from time to time, and for most of the time, only about 20 nodes (or 40 GPUs T10) are available.

4.2 GPU-based Parallel Pricing Framework

In this section, we outline the pricing frameworks for PRDC swaps on a multi-GPU platform/cluster of GPUs. The focus of this section is on Bermudan cancelable and FX-TARN PRDC swaps, since, over each period of the swaps' tenor structure, the pricing problem of these derivatives can be partitioned into several independent pricing subproblems, each of which can be solved efficiently on a GPU (see Sections 3.4 and 3.6). On the other hand, the pricing problem of knock-out PRDC swaps can be solved efficiently on a single GPU, and its pricing over each time period can be viewed as similar to that of an independent subproblem of FX-TARN PRDC swaps (see Remark 4.2.2 on page 69).

A typical application on a multi-GPU platform or a cluster of GPUs usually involves (i) running multiple processes on separate GPUs in parallel, and (ii) communication between different processes/GPUs. Regarding (i), in a typical GPU code, there usually are two levels of parallelism. The first one is between the host (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. Regarding (ii), it is important to note that, currently, there is no mechanism for one GPU to directly communicate with another GPU without explicitly sending the data through the host, if they are attached to the same host (Scenario 1). Moreover, if the two GPUs are on different network nodes, the data have to be sent over the network as well (Scenario 2). As a result, certain message-passing libraries and tools for data exchange and communication between different processes need to be employed. The pricing frameworks for Bermudan cancelable and FX-TARN PRDC swaps presented in the following sections illustrate in detail the two aforementioned scenarios.

4.2.1 Bermudan Cancelable PRDC Swaps

In pricing Bermudan cancelable PRDC swaps, for each period $[T_{(\alpha-1)^+}, T_{\alpha^-}]$, we need to solve the PDE (2.12) backward in time from T_{α^-} to $T_{(\alpha-1)^+}$ with payoffs (3.6) and (3.8) to obtain $u^c_{\alpha-1}(T_{(\alpha-1)^+})$ and $u^h_{\alpha-1}(T_{(\alpha-1)^+})$, respectively (Lines 4 and 5 of Algorithm 3.4.1). These two pricing processes are entirely independent within each time period of the tenor structure, and require communication only at the end of the time period. Thus, for each time period, it is natural to assign the two pricing processes to separate GPUs. Note that, since there are two GPUs attached to a node of the cluster, our pricing approach for Bermudan cancelable PRDC swaps requires communication between GPUs attached to the same host.

More specifically, for each time period, the GPU-based pricing of a Bermudan cancelable PRDC swap consists of four phases. During the first phase, the host assigns the two pricing processes to separate GPUs, using the Master-Slave model. During the second phase, on the first GPU (GPU 0), the PRDC coupons are included via (3.6), while on the second GPU (GPU 1), the early exercise condition (3.8) is enforced. Note that this phase uses the Peer model as the parallelization paradigm, and its implementation is described in Sections 4.5. During the third phase, the PDE (2.12) is solved simultaneously in each GPU from T_{α^-} to $T_{(\alpha-1)^+}$ with terminal condition (3.6) or (3.8), respectively, using the Peer model for each timestep. The parallelism in each GPU for this phase is based on an efficient parallelization of the computation of each timestep of the ADI timestepping technique (3.4a)–(3.4d). A detailed description of the parallel algorithm of this phase is presented in Section 4.3. During the last phase, which concludes the GPU-based parallel procedure via the Master-Slave model, the host (the CPU) collects the pricing results for the past period $[T_{(\alpha-1)^+}, T_{\alpha^-}]$ and performs data exchange for next period $[T_{(\alpha-2)^+}, T_{(\alpha-1)^-}]$. Figure 4.2.1 provides an illustration of our pricing approach.

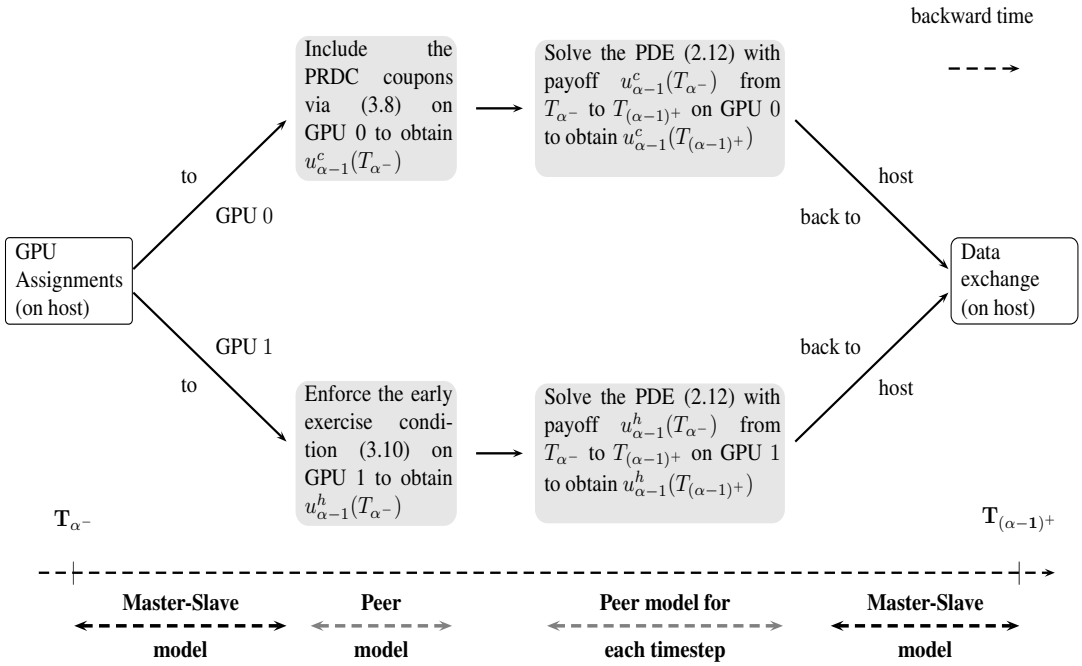


Figure 4.2.1: Four phases of the pricing of PRDC swaps with Bermudan cancelable features over each time period $[T_{(\alpha-1)^+}, T_{\alpha^-}]$ of the tenor structure and the associated parallelization paradigm for each phase.

In our pricing approach, over each period of the swap's tenor structure, two pieces of a single-GPU code run on two GPUs, each of which is essentially for the solution of a PDE and some related works. It is desirable to start both processes 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.

4.2.2 FX-TARN PRDC Swaps

The key point in Algorithm 3.6.1 is that, over each time period $[T_{(\alpha-1)^+}, T_{\alpha^-}]$ of the tenor structure, we have multiple, entirely independent, pricing subproblems to solve, each of which corresponds to the discrete value a_y, $y = 0, \ldots, w$. Each of these independent sub-

problems involves essentially (i) performing interpolation using some of the $u_\alpha(T_{\alpha^+}; a_y)$, $y = 0, \ldots, w$, assuming the relevant data are available, to obtain $u_{\alpha-1}(T_{\alpha^+}; \bar{a}_y)$, where $\bar{a}_y$ is defined by (3.17); (ii) updating the terminal condition (3.19) using $u_{\alpha-1}(T_{\alpha^+}; \bar{a}_y)$ obtained from (i), (iii) solving the PDE (2.12) backward in time from T_{α^-} to $T_{(\alpha-1)^+}$ with this terminal condition, and (iv) incorporating the funding payment via (3.20) (see Lines 4–8 in Algorithm 3.6.1). All of these steps can be efficiently implemented on a GPU.

From the above observations, within each time period of the tenor structure, it is natural to assign these $w + 1$ pricing processes to separate GPUs. In a practical application, $w + 1$ is typically much larger than two, the number of GPUs attached to a node of the cluster. As a result, to solve these $w + 1$ subproblems simultaneously, GPUs on different nodes of the cluster must be employed. However, communication between these pricing processes is required at each date of the tenor structure, due to the interpolation (3.18). In this case, certain message-passing libraries and tools for communication between different network nodes need to be employed. In this thesis, we utilize the Message Passing Interface (MPI) [22, 23], a widely used message passing library standard. The combination of MPI, which is for communication between the processes, and CUDA, which is for the GPU-based parallel computation within each process, gives rise to a hybrid MPI-CUDA implementation of the backward pricing Algorithm 3.6.1 for FX-TARN PRDC swaps.

We now discuss the pricing framework for FX-TARN PRDC swaps over each time period $[T_{(\alpha-1)^+}, T_{\alpha^-}]$ of the tenor structure. In the following presentation, we assume that the total number of available nodes of the cluster is at least $w + 1$. 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. To

proceed from T_α to $T_{\alpha-1}$, assume that $u_\alpha(T_{\alpha^+}; a_y)$, $y = 0, \ldots, w$, are computed at the previous period of the tenor structure. From a parallelization perspective, the hybrid MPI-CUDA implementation of Algorithm 3.6.1 consists of two stages, described in detail below.

The first stage, referred to as *Stage I*, involves communication between processes of the group via MPI at time T_{α^+}. More specifically, the purpose of this stage is for each process to collect data, i.e. some of the $u_\alpha(T_{\alpha^+}; a_y)$, $y = 0, \ldots, w$, from other processes for interpolation occurring in the second stage. Note that these data are assumed to be in the respective host memory. The data exchange between processes could be implemented via either a two-sided data transfer approach using MPI send and receive operations, such as `MPI_Send(···)`, `MPI_Recv(···)` (blocking operations), `MPI_Isend(···)` and `MPI_Irecv(···)` (non-blocking operations),[1] or a single-sided data transfer approach using `MPI_Put(···)` and `MPI_Get(···)`.[2] At the end of the first stage, each process of the group has in its host memory the data it needs to perform interpolation.

Remark 4.2.1. Note that, in Stage I, we do not carry out an all-to-all data exchange between processes, but rather, only the necessary communication between processes. This is due to the fact that a process may need some data from a higher-rank process, but not the other way around (see Remark 3.6.1 on page 55 and Figure 3.6.3). As a result, in our implementation, there is a "marking" phase for Stage I in which each process of the group determines the ranks of the other (higher-rank) processes whose data it needs for interpolation. This "marking" phase happens at T_{α^+}, and is carried out by each process on the associated GPU using its own data $u_\alpha(T_{\alpha^+}; a_y)$, $y = 0, \ldots, w$, before copying the data to the process' host memory. A detailed description of this procedure is presented

[1] A non-blocking send or receive function returns immediately, regardless of whether the send or receive has finished. On the other hand, blocking send and receive functions do not return until the operations have been completed.

[2] In a two-sided data transfer, one processor executes an `MPI_Send(···)`/`MPI_Isend(···)` and a *different* processor executes an `MPI_Recv(···)`/`MPI_Irecv(···)`. On the other hand, in a singled-sided data transfer, one processor has to call only one of the single-sided data transfer routines.

in Subsection 4.4.1.

In the second stage, referred to as *Stage II*, all the pricing processes run in parallel on separate GPUs, each of which is for an independent pricing subproblem. Note that each node involved in the pricing is associated with two processes, taking into account the fact there are two GPUs per node. The second stage of the pricing framework can be viewed as consisting of the following seven phases.

1. During the first phase, the host of each node assigns the two associated pricing processes to separate GPUs using the Master-Slave model.

2. During the second phase, the interpolation scheme (3.18) is applied. This phase corresponds to Line 5 of Algorithm 3.6.1. Details of the implementation of the second phase are given in Section 4.4.

3. During the third phase, the PRDC coupons are included via (3.19). This phase corresponds to Line 6 of Algorithm 3.6.1. Details of the implementation of the third phase are given in Section 4.5. Note that the second and third phases use the Peer model as the parallelization paradigm.

4. During the fourth phase, the PDE (2.12) is solved from T_{α^-} to $T_{(\alpha-1)^+}$ with the terminal condition obtained from the third phase using the Peer model. This phase corresponds to Line 7 of Algorithm 3.6.1. The parallelism in a GPU for this phase is based on an efficient parallelization of the computation of each timestep of the ADI timestepping technique (3.4a)–(3.4d). A detailed description of the parallel algorithm of this phase is presented in Section 4.3.

5. During the fifth phase, the funding payments are included via (3.20) using the Peer model. This phase corresponds to Line 8 of Algorithm 3.6.1. The implementation of this phase is similar to that of the third phase and is described in Section 4.5.

6. The "marking" for the data exchange between processes in the next time period occurs in the sixth phase, details of which are given in Subsection 4.4.1 (also see Remark 4.2.1 on page 67).

7. During the last phase, which concludes the GPU-based parallel procedure via the Master-Slave model, the host of each node collects the pricing results of the two associated pricing processes for the past period $[T_{(\alpha-1)^+}, T_{\alpha^-}]$, and prepares for the data exchange between processes of the group for the next period $[T_{(\alpha-2)^+}, T_{(\alpha-1)^-}]$.

Figure 4.2.2 provides an illustration of our approach. To keep the figure clear, we only show the phases of the ith process in the group.

Remark 4.2.2. Over each time period of the swap's tenor structure, the pricing of a knockout PRDC swap on a GPU can be viewed as similar to the pricing of a pricing subproblem described above. More specifically, the computation of Lines 3–6 of Algorithm 3.5.1 for pricing knockout PRDC swaps are essentially similar to the three steps on Lines 5–8 (the third, fourth and fifth phases), respectively, of Algorithm 3.6.1 for a subproblem in pricing FX-TARN PRDC swaps.

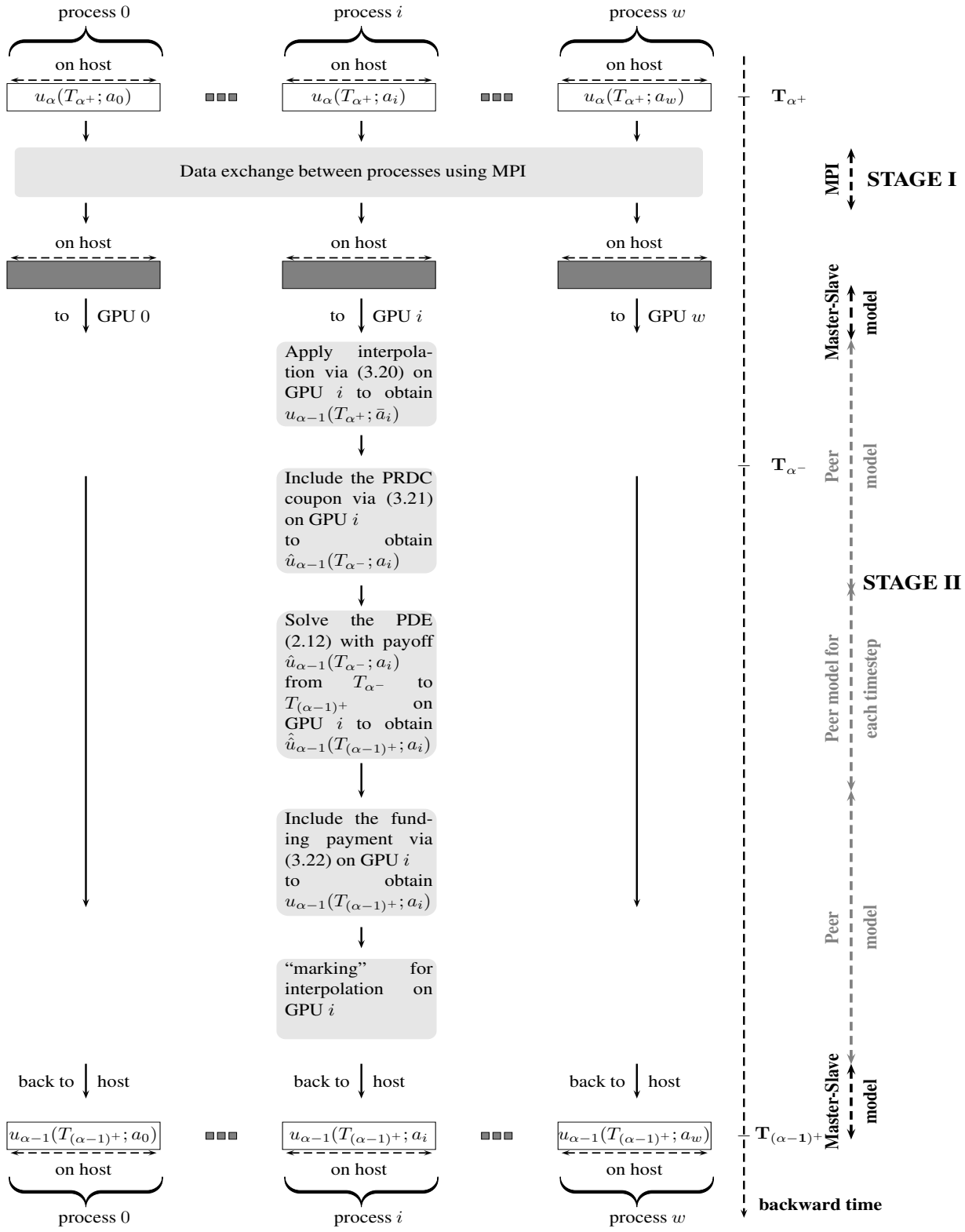


Figure 4.2.2: The pricing framework of FX-TARN PRDC swaps over each time period $[T_{(\alpha-1)^+}, T_{\alpha^-}]$ of the tenor structure.

4.3 ADI Timestepping Schemes

We now discuss a GPU-based parallel algorithm for the solution of the PDE (2.13). We emphasize that we do not aim to parallelize across time, but rather we focus on the parallelism within one timestep, via a parallelization of the HV scheme ((3.4a)-(3.4e)).

As mentioned earlier, the HV scheme can be divided into two phases. The first phase consists of a forward Euler step (predictor step (3.4a)), followed by three implicit, but unidirectional, corrector steps (3.4b), the purpose of which is to stabilize the predictor step. The second phase (i.e. (3.4c) and (3.4d)) restores second-order convergence of the discretization method, since the PDE (2.13) contains cross derivatives. Step (3.4e) 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 independent tridiagonal system solves. Hence, for brevity, we focus on describing a GPU parallelization of the first phase, only briefly mentioning the second phase of parallelization.

For presentation purposes, let

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

and notice that

$$\mathbf{v}_0 = \mathbf{u}^{m-1} + \sum_{i=0}^{3} \mathbf{w}_i + \Delta\tau\mathbf{g}^m.$$

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.

The first phase of the HV scheme consists of the following steps:

(i) Step a.1: Compute the matrices $\mathbf{A}_i^m$, $i = 1, 2, 3$, and $\widehat{\mathbf{A}}_i^m$, $i = 1, 2, 3$, and the vectors $\mathbf{w}_i$, $i = 0, 1, 2, 3$, and $\mathbf{v}_0$.

(ii) Step a.2: Set $\widehat{\mathbf{v}}_1 = \mathbf{v}_0 - \theta \mathbf{w}_1$ and solve $\widehat{\mathbf{A}}_1^m \mathbf{v}_1 = \widehat{\mathbf{v}}_1$;

(iii) Step a.3: Set $\widehat{\mathbf{v}}_2 = \mathbf{v}_1 - \theta \mathbf{w}_2$ and solve $\widehat{\mathbf{A}}_2^m \mathbf{v}_2 = \widehat{\mathbf{v}}_2$;

(iv) Step a.4: Set $\widehat{\mathbf{v}}_3 = \mathbf{v}_2 - \theta \mathbf{w}_3$ and solve $\widehat{\mathbf{A}}_3^m \mathbf{v}_3 = \widehat{\mathbf{v}}_3$;

Each of the Steps a.2, a.3, and a.4, when considered individually, is inherently parallelizable, due to the block diagonal structure of each of the matrices $\widehat{\mathbf{A}}_i^m$, $i = 1, 2, 3$. However, as discussed later in Remark 4.3.2, heavy communication (associated with copying from/to the global memory) is required between these steps. On the other hand, Step a.1 is not as straightforward to parallelize, but requires much less communication between the computation of the vectors $\mathbf{w}_i$, $i = 0, 1, 2, 3$. See Remark 4.3.2 for a discussion. Below, we discuss in more detail how to implement these steps on a GPU with particular focus on Step a.1, since Steps a.2, a.3, and a.4 are relatively straightforward.

4.3.1 First Phase - Step a.1

In this subsection, we first describe the partitioning of the computational grid into a grid of blocks, then we discuss the assignment of gridpoints to threads of threadblocks. We next illustrate how the matrices $\mathbf{A}_i^m$, $i = 0, 1, 2, 3$, and $\widehat{\mathbf{A}}_i^m$, $i = 1, 2, 3$, are assembled. Then we present an efficient GPU-based computation algorithm for the vectors $\mathbf{w}_i$, $i = 0, 1, 2, 3$, and $\mathbf{v}_0$.

Partitioning of the Computational Grid and Assignment of Gridpoints to Threads

Recall that we have a discretization grid of $n \times p \times q$ points. We can view a set of q consecutive gridpoints in the r_f-direction as a "stack" of q gridpoints. Clearly, there are np such stacks in the discretization grid. The general idea for distributing the data and computation of Step a.1 is to assign the work associated with each stack of q gridpoints

(and the respective rows of matrices $\mathbf{A}_i^{m-1}$, $i = 0, 1, 2, 3$, $\widehat{\mathbf{A}}_i^{m-1}$, $i = 1, 2, 3$, and components of vectors $\mathbf{u}^{m-1}$, $\mathbf{w}_i$, $i = 0, 1, 2, 3$, and $\mathbf{v}_0$) to a different thread. Thus, there is an one-to-one correspondence between the set of stacks and the set of threads. Moreover, just as CUDA threads are grouped into threadblocks, we also group gridpoints into blocks. 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, 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 threadblock only, with one thread for each stack of q gridpoints, i.e. there is an one-to-one correspondence between the set of 3-D blocks of gridpoints and the set of threadblocks (see also Figure 4.3.1). 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.

Figure 4.3.1 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 $p_b = 2$. In this example, 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. The kernel generates a grid of threadblocks of size $\texttt{ceil}\left(\frac{n}{n_b}\right) \times \texttt{ceil}\left(\frac{p}{p_b}\right) \equiv \frac{8}{4} \times \frac{8}{2} \equiv 2 \times 4$, with each 2-D threadblock having size 4×2. Each 3-D block of gridpoints is assigned to a threadblock which carries out all the computations/work associated with all gridpoints of the ten 4×2 tiles, in a 10-iteration loop in the kernel, proceeding tile-by-tile.

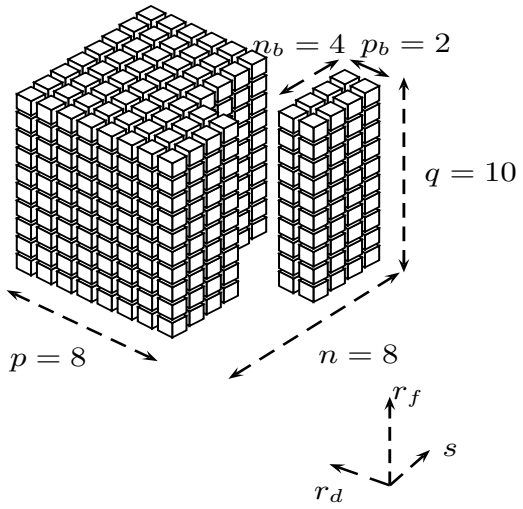


Figure 4.3.1: An illustration of the partitioning approach considered for the first phase, Step a.1.

Construction of the Matrices $\mathbf{A}_i^m$, $i = 1, 2, 3$, and $\widehat{\mathbf{A}}_i^m$, $i = 1, 2, 3$

Note that each of the matrices $\mathbf{A}_i^m$, $i = 0, 1, 2, 3$, and $\widehat{\mathbf{A}}_i^m$, $i = 1, 2, 3$, 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. In this way, the first np rows are processed in parallel during the first iteration, then the second np rows (from the $(np + 1)$st row to the $2np$-th row) are processed in a similar way during the second iteration, and so on.

Computation of the Vectors $\mathbf{w}_i$, $i = 0, 1, 2, 3$, and $\mathbf{v}_0$

Recall that the vectors $\mathbf{w}_i$, $i = 0, 1, 2, 3$, are given by

$$\mathbf{w}_i = \Delta\tau \mathbf{A}_i^{m-1}\mathbf{u}^{m-1} + \Delta\tau(\mathbf{g}_i^{m-1} - \mathbf{g}_i^m),$$

and the vector $\mathbf{v}_0$ by $\mathbf{v}_0 = \mathbf{u}^{m-1} + \sum_{i=0}^{3} \mathbf{w}_i$. Note that the data of the previous timestep (old data), i.e. the vector $\mathbf{u}^{m-1}$, and model constant parameters, if any, are available in the global memory and constant cache, respectively. We emphasize that the data copying from the host memory to the device memory occurs on the first timestep only, for the initial condition data, $\mathbf{u}^0$, and the model constants. Data for the subsequent timesteps and the ADI timestepping scheme (3.4) are stored on the device memory. The initial host-to-device copying can be achieved via CUDA functions `cudaMemcpy2D(···)` for data and `cudaMemcpyToSymbol(···)` for constants.

Before describing how the computation of the vectors $\mathbf{w}_i$, $i = 0, 1, 2, 3$, is carried out, we need to draw the reader's attention to certain facts.

(a) The FD discretization scheme used in this paper, as described by (3.1)–(3.2), gives rise to a 19-point stencil (see Figure 3.2.1). This implies that, from the point-of-view of a thread, the product of a row of $\mathbf{A}_i^{m-1}$ by $\mathbf{u}^{m-1}$ involves 19 components of $\mathbf{u}^{m-1}$, three of which are assigned to the thread, and 16 others are assigned to neighbouring threads.

(b) A threadblock carrying the computation of a stack of q tiles (the stack being in the r_f direction) needs the values of neighbouring gridpoints from adjacent tiles in the s and r_d directions, referred to as *halo* values.

(c) Threads within the same threadblock can communicate with each other effectively via the shared memory, while threads in different threadblocks cannot.

(d) Accessing the global memory is costly, and, therefore, the number of accesses to the global memory should be kept to a minimum. Also, global memory access coalescing is desirable whenever possible, as it minimizes the cost of global memory access. Memory coalescing is discussed in more detail below.

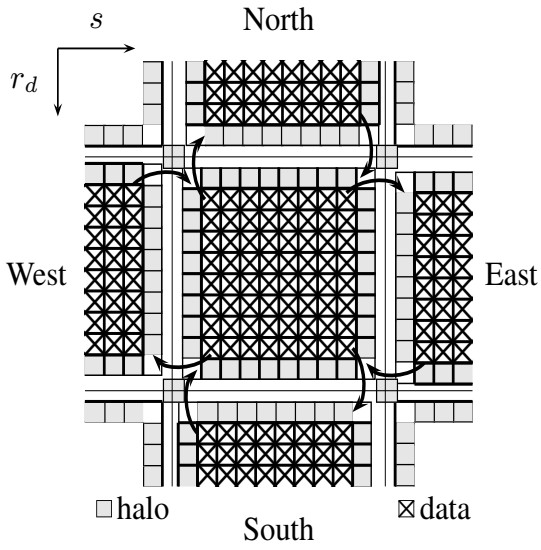


Figure 4.3.2: An example of $n_b \times p_b = 8 \times 8$ tiles with halos.

(e) 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.

Taking into account the above considerations, we adopt an effective data loading strategy, so that each thread of a threadblock has the necessary data to compute the associated components of $\mathbf{w}_i$, the number of global memory accesses is small, and partial memory coalescing is achieved. 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 and associated halos are in the shared memory from the previous iteration, each threadblock

1. loads from the global memory into its shared memory the old data (from vector $\mathbf{u}^{m-1}$) corresponding to the $(k+1)$st tile, and the associated halos (in the s- and r_d-directions), if any,

2. computes and stores new values (components of the 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, and of the associated halos, if any,

3. copies the newly computed data of the kth tile (components of the 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, and associated halos, if any, so that they can be used in the next iteration.

From the point-of-view of a thread, 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.

In Figure 4.3.2, an example illustrating the aforementioned data loading approach with tiles having size $n_b \times p_b \equiv 8 \times 8$ (marked in ⊠) with halos (marked in □) is presented. Note that, from the viewpoint of the central tile, the halos are not part of the tile, but come from adjacent tiles: the North, South, East, West halos are from neighboring tiles in those respective directions. As can be seen from Figure 4.3.2, during each iteration of the q-iteration loop in the kernel, only threads whose gridpoints are close to a boundary of a tile need to load into the shared memory the halo values along that boundary in addition to their own gridpoints' old data. Threads corresponding to interior gridpoints of a tile need to read in only their gridpoints' old data. Thus, old data corresponding to interior gridpoints of a tile (i.e. the majority of the gridpoints) are loaded once only. The old data corresponding to a point along the North, South, East, or West boundaries of a tile are loaded twice, once by the thread associated with the point, and once more by the thread of the neighbouring threadblock as a halo value. The data at the tiles' corners are loaded four times. This loading strategy keeps the repetition of data loading, and, therefore, the number of global memory accesses to a minimum. Furthermore, this

loading approach is partially coalesced, as is discussed in more detail towards the end of this section.

Once in the shared memory of a threadblock, the data corresponding to a gridpoint is accessed (at most) 19 times: three times by the thread to which the gridpoint is assigned, and 16 times by the 8 neighbouring threads in the same threadblock. We emphasize that these repeated shared memory accesses are cheap.

Remark 4.3.1. It is important to note that, in our implementation, barrier synchronization among threads in the same threadblock, which can be achieved by using the function `__syncthreads()`, is used in both the loading and computing phases of each iteration of the q-iteration loop in the kernel. The purpose of barrier synchronization is to ensure 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). This is essential because one thread may need the data loaded or computed by a neighboring thread in the same threadblock.

Memory Coalescing

To optimize performance, we must ensure coalesced data loads from the global memory. It is important to recall that threads in a warp should execute the same instruction at any given time, in order to avoid potential serial execution of threads. Furthermore, in CUDA, the shared memory consists of 16 memory banks. To achieve maximum memory performance, memory accesses at any one time are handled by half-warps (16 threads), each thread accessing a different bank. When the threads of a half-warp execute global loads, the loads are consolidated if they meet certain constraints necessary for the hardware to perform coalesced data loads. These constraints include (i) the threads in the half-warp must access consecutive global memory locations; and (ii) the number of threads matters only along the first dimension of the threadblock. We refer interested readers to [57] for

a more complete discussion of all the requirements. Regarding the GPU computation step for vectors $\mathbf{w}_i$, $i = 0, 1, 2, 3$, to ensure the data transfer coalescing, it is necessary to have the tile size in the s-direction, i.e. n_b, be 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. Assuming that n_b is a multiple of 16, and taking into account the data loading strategy for $\mathbf{u}^{m-1}$ that we adopted, the interior of the data tiles can be read from global memory into the shared memory in a coalesced way. The halos along the s-direction (i.e. North and South halos in Figure 4.3.2) can also be loaded in a coalesced fashion. However, halos along the r_d-direction (East and West halos) cannot be accessed via a coalesced pattern, since they do not belong to consecutive memory locations. As a result, 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.

4.3.2 First Phase - Steps a.2, a.3, a.4

Recall that, in Steps a.2, a.3 and a.4, we need to solve $\widehat{\mathbf{A}}_i^m \mathbf{v}_i = \widehat{\mathbf{v}}_i, i = 1, 2, 3$, where

$$\widehat{\mathbf{v}}_i = \mathbf{v}_{i-1} - \theta \mathbf{w}_i, \quad i = 1, 2, 3.$$

After Step a.1, the matrices $\widehat{\mathbf{A}}_i^m$, $i = 1, 2, 3$, and the vectors $\mathbf{w}_i$, $i = 1, 2, 3$ and $\mathbf{v}_0$ are stored in the device memory. Similarly, the data after Steps a.2, a.3, a.4 (i.e. $\mathbf{v}_i$, $i = 1, 2, 3$) are held in the device memory, and as a result, the elements of the right-side vectors $\widehat{\mathbf{v}}_i$, $i = 1, 2, 3$, can be easily computed in parallel. The parallel solution of the tridiagonal systems $\widehat{\mathbf{A}}_i^m \mathbf{v}_i = \widehat{\mathbf{v}}_i$, $i = 1, 2, 3$, can be achieved via a data partitioning different from the one described for Step a.1. The data partitioning for Steps a.2, a.3 and a.4 is motivated by the block structure of the tridiagonal matrices $\widehat{\mathbf{A}}_i^m$. For example, $\widehat{\mathbf{A}}_1^m$ has pq diagonal blocks, and each block is tridiagonal of size $n \times n$, while $\widehat{\mathbf{A}}_2^m$ has nq diagonal blocks, and each block is tridiagonal of size $p \times p$. Our approach for the solution of $\widehat{\mathbf{A}}_i^m \mathbf{v}_i = \widehat{\mathbf{v}}_i$, $i = 1, 2, 3$, is based on the parallelism arising from independent tridiagonal

solutions, rather than the parallelism within each one. To this end, each independent tridiagonal system is assigned to a different thread. Moreover, when we solve in one direction, the data are partitioned with respect to the other two. For example, the solution of $\widehat{\mathbf{A}}_1^m \mathbf{v}_1 = \widehat{\mathbf{v}}_1$ (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 pq threads. Similarly, the solution of $\widehat{\mathbf{A}}_2^m \mathbf{v}_2 = \widehat{\mathbf{v}}_2$ (Step a.3) is done via re-partitioning $\widehat{\mathbf{A}}_2^m$ and $\widehat{\mathbf{v}}_2$ into nq independent $p \times p$ tridiagonal systems, and then assigning each tridiagonal system to one of nq threads. Between Steps a.2 and a.3, the data of vector $\mathbf{v}_1$ are written back to the global memory, since, in Step a.3, a different partitioning of $\mathbf{v}_1$ is needed to compute the right-side vector $\widehat{\mathbf{v}}_2$ before $\widehat{\mathbf{A}}_2^m \mathbf{v}_2 = \widehat{\mathbf{v}}_2$ is solved.

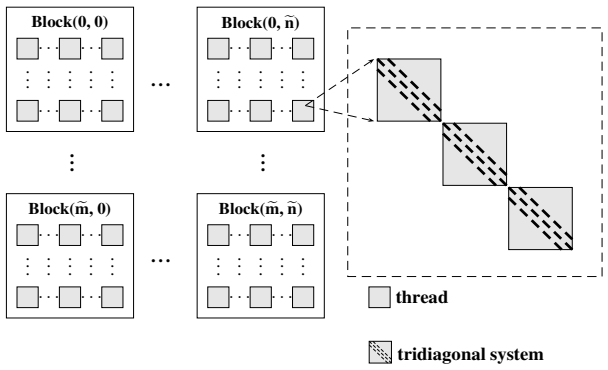


Figure 4.3.3: Thread assignment for the parallel solution of independent tridiagonal systems. Each thread handles one tridiagonal system.

To have one thread handle the solution of each tridiagonal system, we need to use sufficiently many threadblocks. Because the number of threads in a threadblock is limited to 512, and because we do not wish to have limitations on the grid sizes, we use many threadblocks for the solution of the independent tridiagonal systems. More specifically, at Steps a.2, a.3 and a.4, we have pq, nq and np total threads, respectively. In our implementation, each of the 2-D threadblocks has the 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 $\widehat{\mathbf{A}}_1^m \mathbf{v}_1 = \widehat{\mathbf{v}}_1$, a 2-D grid of threadblocks of size $\texttt{ceil}(\frac{p}{r_t}) \times \texttt{ceil}(\frac{q}{c_t})$ is invoked. An example of this

approach is illustrated in Figure 4.3.3, where the 2-D grid of threadblocks is of size $\texttt{ceil}(\frac{p}{r_t}) \times \texttt{ceil}(\frac{q}{c_t}) \equiv (\widetilde{m}+1) \times (\widetilde{n}+1)$.

Memory Coalescing

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 r_d-, then r_f-directions. As a result, the data partitionings for the tridiagonal solves in the r_d- and r_f-directions, 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.

More specifically, to illustrate the memory coalescing at Step a.2, a.3 and a.4, we consider a toy example of the computational grid of size $3 \times 3 \times 3$ presented in Figure 4.3.4. Note that all the points are number in the s-, then r_d-, then r_f-directions. In solving $\widehat{\mathbf{A}}_1^m \mathbf{v}_1 = \widehat{\mathbf{v}}_1$, threads in a warp do not access data in consecutive memory locations. For example, the data points with indices 0, 1 and 2, and the data points with indices 3, 4 and 5 are assigned to two consecutive threads of a warp. However, data points with indices 0 and 3 are not in consecutive memory locations, and neither are the data points with indices 1 and 4, and 2 and 5. On the other hand, in solving $\widehat{\mathbf{A}}_i^m \mathbf{v}_i = \widehat{\mathbf{v}}_i$, $i = 2, 3$, threads in a warp do access data in consecutive locations. For example, in solving $\widehat{\mathbf{A}}_2^m \mathbf{v}_2 = \widehat{\mathbf{v}}_2$, the data points with indices 0, 3 and 6, and the data points with indices 1, 4 and 7 are respectively in consecutive memory locations, and are assigned to two consecutive threads of a warp.

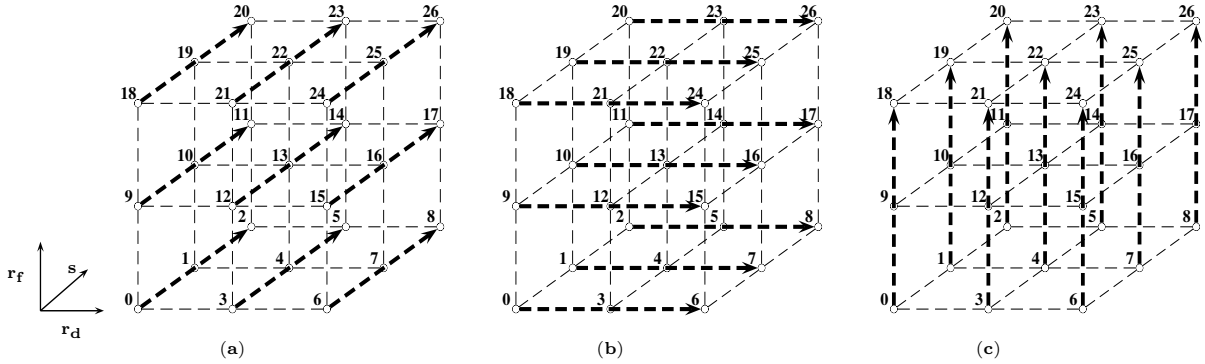


Figure 4.3.4: Tridiagonal solves along each spatial dimension in Steps (3.4b) and (3.4d): **(a)** along s, i.e. $\widehat{\mathbf{A}}_1^m \mathbf{v}_1 = \widehat{\mathbf{v}}_1$ (no memory coalescing), **(b)** along r_d, i.e. $\widehat{\mathbf{A}}_2^m \mathbf{v}_2 = \widehat{\mathbf{v}}_2$ (memory coalescing), and **(c)** along r_f, i.e. $\widehat{\mathbf{A}}_3^m \mathbf{v}_3 = \widehat{\mathbf{v}}_3$ (memory coalescing).

Note that it is possible to obtain full memory coalescence for the tridiagonal solves in all directions. See Remark 4.3.3 for details.

Remark 4.3.2. It is important to emphasize that, while the corrector steps, i.e. Steps a.2, a.3 and a.4, are inherently parallelizable, they require heavy communication (associated with copying to/from the global memory) between steps. More specifically, at the end of each corrector step, all data must be copied from the shared memory to the global memory. The data are then copied back to the shared memory during the next corrector step. On the other hand, between explicit steps, only halos need to be copied to/from the global memory. In other words, if the numerical method consisted of only explicit steps, only local communication would be needed between the steps, but, since the ADI method involves implicit (corrector) steps with solves along alternating directions, global communication is required. As a result, for the explicit steps, the communication cost is much less than the computational cost, whereas, for the corrector steps, the communication cost is of the same order as the computational cost.

4.3.3 Summary of the First Phase

In this subsection, we summarize the data loading and computation of the first phase of the ADI scheme (3.4), when stepping from time τ_{m-1} to τ_m.

We assume that, initially, the vector $\mathbf{u}^{m-1}$ is in the global memory and any needed constants (model parameters) are in the constant cache.

Step a.1:

A grid of $\texttt{ceil}(n/n_b) \times \texttt{ceil}(p/p_b)$ threadblocks is invoked, each of which consists of an $n_b \times p_b$ array of threads. Each threadblock does a q-iteration loop, processing an $n_b \times p_b$ tile at each iteration, and thus each thread does a q-iteration loop, processing one gridpoint at each iteration.

During one iteration of the q-iteration loop, each threadblock

1. loads from the global memory to its shared memory the components of $\mathbf{u}^{m-1}$ corresponding to a tile, and the associated halo values;

2. computes the rows of $\mathbf{A}_i^m, \widehat{\mathbf{A}}_i^m$, $i = 1, 2, 3$ and components of $\mathbf{w}_i$, $i = 0, 1, 2, 3$, and $\mathbf{v_0}$ corresponding to the tile;

3. copies the newly computed rows of $\mathbf{A}_i^m, \widehat{\mathbf{A}}_i^m$, $i = 1, 2, 3$ and components of $\mathbf{w}_i$, $i = 1, 2, 3$, and $\mathbf{v_0}$ from its shared memory to the global memory.

Step a.2:

A grid of $\texttt{ceil}(p/r_t) \times \texttt{ceil}(q/c_t)$ threadblocks is invoked, each of which consists of an $r_t \times c_t$ array of threads. Each threadblock is assigned a subgrid of $n \times r_t \times c_t$ points, and thus each thread is assigned n points along the s-direction.

Each threadblock

1. loads from the global memory to its shared memory its rows of $\widehat{\mathbf{A}}_1^m$ and its components of $\mathbf{v_0}$ and $\mathbf{w}_1$;

2. computes its components of $\widehat{\mathbf{v}}_1$;

3. solves $r_t c_t$ tridiagonal $n \times n$ systems (its part of $\widehat{\mathbf{A}}_1^m \mathbf{v_1} = \widehat{\mathbf{v}}_1$), with each thread solving one system;

4. copies its newly computed components of $\mathbf{v_1}$ from its shared memory to the global memory.

Step a.3:

A grid of $\texttt{ceil}(n/r_t) \times \texttt{ceil}(q/c_t)$ threadblocks is invoked, each of which consists of an $r_t \times c_t$ array of threads. Each threadblock is assigned a subgrid of $r_t \times p \times c_t$ points, and thus each thread is assigned p points along the r_d-direction.

Each threadblock

1. loads from the global memory to its shared memory its rows of $\widehat{\mathbf{A}}_2^m$ and its components of $\mathbf{v_1}$ and $\mathbf{w}_2$;

2. computes its components of $\widehat{\mathbf{v}}_2$;

3. solves $r_t c_t$ tridiagonal $p \times p$ systems (its part of $\widehat{\mathbf{A}}_2^m \mathbf{v_2} = \widehat{\mathbf{v}}_2$), with each thread solving one system;

4. copies its newly computed components of $\mathbf{v_2}$ from its shared memory to the global memory.

Step a.4:

A grid of $\texttt{ceil}(n/r_t) \times \texttt{ceil}(p/c_t)$ threadblocks is invoked, each of which consists of an $r_t \times c_t$ array of threads. Each threadblock is assigned a subgrid of $r_t \times c_t \times q$ points, and thus each thread is assigned q points along the r_f-direction.

Each threadblock

1. loads from the global memory to its shared memory its rows of $\widehat{\mathbf{A}}_3^m$ and its components of $\mathbf{v_2}$ and $\mathbf{w}_3$;

2. computes its components of $\widehat{\mathbf{v}}_3$;

3. solves $r_t c_t$ tridiagonal $q \times q$ systems (its part of $\widehat{\mathbf{A}}_3^m \mathbf{v_3} = \widehat{\mathbf{v}}_3$), with each thread solving one system;

4. copies its newly computed components of $\mathbf{v_3}$ from its shared memory to the global memory.

At the end of the first phase, the vector $\mathbf{v_3}$ is available in the global memory, and will be used in the second phase.

4.3.4 Second Phase

Several components required in the second phase of the HV scheme are available as results of the computations in the first phase, and hence are not recomputed. For example, the term $\mathbf{A}^{m-1}\mathbf{u}^{m-1}$, required in (3.4c), is needed to compute $\mathbf{v}_0$ in Step a.1 of the first phase. The computation of $\mathbf{A}^m\mathbf{v_3}$ in (3.4c) can be achieved in the same way as the computation of $\mathbf{A}^{m-1}\mathbf{u}^{m-1}$ in Step a.1. The solutions of the tridiagonal systems in (3.4d) can be implemented in essentially the same way as in Steps a.2, a.3, a.4, described above. Note that all the tridiagonal matrices $\mathbf{I} - \theta\Delta\tau\mathbf{A}_i^m$, $i = 1, 23$, are already computed in Step a.1 of the first phase.

4.3.5 Possible Improvements

Remark 4.3.3. It is possible to achieve memory coalescence for the tridiagonal solves in all three directions, if we renumber the gridpoints between Steps a.1, a.2, a.3 and a.4 appropriately. More specifically, if the data between Steps a.1 and a.2 are ordered in the r_f-direction first instead of the s-direction as in the current implementation, memory coalescence is fully achieved for Step a.2, i.e. for the tridiagonal solve in the s-direction. Full memory coalescence for Steps a.3 and a.4 can be obtained by ordering the data

between Steps a.2, a.3 and a.4 in the s-direction first as in the current implementation. That is, full memory coalescence for Steps a.2, a.3 and a.4 can be achieved via the following two different orderings of the data: (i) in the r_f-direction first for Step a.2 and (ii) in the s-direction first for Steps a.3 and a.4. However, such a renumbering will involve some overhead. In the future, we plan to investigate the trade-off between this overhead and achieving memory coalescence in two of the three directions only. The numerical experiments indicate that the current implementation is effective. Furthermore, between Step a.4, and Step a.1 of the next phase or timestep, the copying of all the data to/from the global memory can be avoided by combining the computation of Step a.4 and the following Step a.1 into one kernel. More specifically, the global communication used in the current implementation at the end of Step a.4 and in the beginning of Step a.1 can be substituted by local communication (copying of halos).

Remark 4.3.4. Another approach that can be employed for the solution of the tridiagonal systems arising in Steps a.2, a,3 and a.4 is to use a parallel cyclic reduction method [30]. In certain implementations [65], 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 the approach presented above.

Remark 4.3.5. Another possible approach that could be used for Steps a.2, a.3 and a.4 is based on partitioning the computational domain into 3-D subcubes of gridpoints, assigning one subcube to each thread, then employing a Schur complement domain decomposition method. To solve $\widehat{\mathbf{A}}_i^m \mathbf{v}_i = \widehat{\mathbf{v}}_i, i = 1, 2, 3$, each thread first solves a block of equations corresponding to its subcube of points in the x_i direction, $i = 1, 2, 3$, i.e. a subdomain problem. Each thread then corrects the solution of the subdomain problem by taking into account the equations on the interfaces between its subcube and neighbouring subcubes, i.e. equations corresponding to points on the boundaries of the subcube. (All

the subdomain problems are solved simultaneously in parallel, with each thread solving one problem.) In this approach, the communication cost (associated with copying from/to the global memory) is much less than the computational cost, assuming that the dimensions of the subcubes are chosen so that there are significantly more interior points within a subcube than interface points, whereas the communication cost of our approach is of the same order as the computational cost. Consequently, this alternative approach is expected to be more efficient and scalable than our approach. However, the implementation of this alternative approach on GPUs is far more complicated. In particular, how to handle memory coalescing is not obvious. Furthermore, to avoid any re-arrangement of data between Steps a.1 and a.2, the parallel implementation of Step a.1 needs to be adjusted to take into account the assignment of data according to subcubes. We plan to investigate this alternative approach in the future.

4.4 Implementation of Interpolation for FX-TARN PRDC swaps on a GPU cluster

The purpose of this section is to discuss issues related to the implementation on a GPU cluster of the interpolation scheme (3.18). Two important issues need to be addressed. The first issue is how each pricing process can acquire all the data it needs for interpolation. The second issue is the GPU-based parallel implementation of the interpolation, assuming all data required by the process are available. Regarding the first issue, the two main questions are: (i) how each process determines the ranks of those processes in the group whose data it needs (this is the "marking" phase for Stage I of the computations, see Remark 4.2.1 on page 67); and (ii) how the communication between processes can be implemented to guarantee correct data exchange (this is Stage I of the computations). We address questions (i) and (ii) of the first issue in Subsection 4.4.1. The implementa-

tion on the GPU of the interpolation scheme (3.18), i.e. the second issue, is discussed in Subsection 4.4.2.

As an illustrative example, we describe the procedures only for process i in the group, where $0 \leq i \leq w$. We denote by $\mathbf{u}_{y,\alpha^+}$ 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 results of the computations during the last time period $[T_{\alpha^+}, T_{(\alpha+1)^-}]$.

4.4.1 The "Marking" Phase and Communication via MPI

The "Marking" Phase

Recall that the "marking" phase for the current time period $[T_{(\alpha-1)^+}, T_{\alpha^-}]$ occurs at the end of the previous time period, i.e. at T_{α^+}. During this phase, process i uses its own vector of data $\mathbf{u}_{i,\alpha^+}$ to determine the ranks of those processes in the group whose data it needs, before writing the data to the host memory, i.e. when $\mathbf{u}_{i,\alpha^+}$ is still in the global memory of the associated host.

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. All gridpoints of a $n_b \times p_b$ 2-D block are assigned to one threadblock only, with one thread for each gridpoint. Figure 4.4.1 illustrates our partitioning approach.

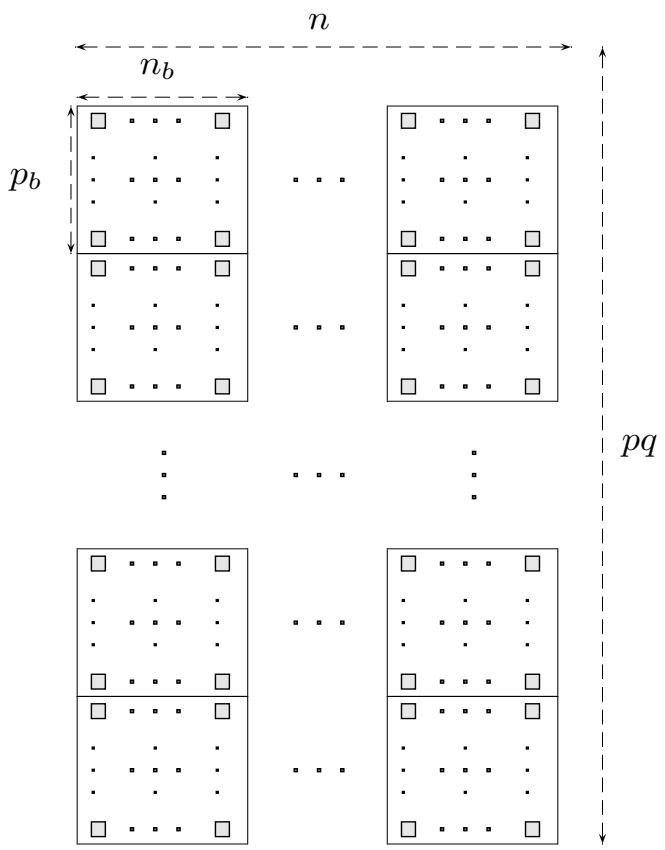


Figure 4.4.1: A partitioning approach of the computational grid into 2-D blocks.

For "marking" purposes, assume that we have an array of size $w+1$ in the global memory, referred to as the array $RECV\_FROM$. The $\bar{i}$th entry of the array $RECV\_FROM$ corresponds to the discrete value $a_{\bar{i}}$, $\bar{i} = 0, \ldots, w$, i.e. it corresponds to the process with rank $\bar{i}$ of the group. All the entries of the array are pre-set to a certain value. This array is copied from the host memory to the device memory before the kernel of the 'marking" phase is launched. Each thread of a threadblock of the kernel launched in this phase computes the quantity $\bar{a}_i$ associated with it via (3.17). If the quantity $\bar{a}_i$ satisfies

$$a_{\bar{i}} \leq \bar{a}_i \leq a_{\bar{i}+1}$$

for some $\bar{i} \in \{i, \ldots, w\}$, the thread then respectively changes the pre-set values of the $\bar{i}$

and $(\bar{i}+1)$st entries in the array $RECV\_FROM$. This procedure essentially marks the ranks of the two processes whose data are required by the process i. Note that no data loadings from the global memory are required for this procedure.

The approach adopted here suggests a $(w+1-i)$-iteration loop in the kernel, during each iteration, each threadblock works with a pair of $a_{\bar{i}}$ and $a_{\bar{i}+1}$. After the kernel has ended, the array $RECV\_FROM$ is copied back to the host memory for investigating. 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 occur. Although we do not know which one, it does not matter for the purpose of determining the ranks of those processes whose some data process y needs. Consequently, this approach suffices and works well.[3]

Inter-process Communication via MPI

At the end of the "marking" phase, process i has in its host memory the vector $\mathbf{u}_{i,\alpha^+}$ and the array $RECV\_FROM$. In order to guarantee that all processes have the data ready in the host memory for communication before the inter-process communication occurs, barrier synchronization among processes of the group is enforced via `MPI_Barrier(···)` before we start Stage I of the computations.

In Stage I, process i can obtain the vector of data from a "marked" process using the "marked" entries of the array $RECV\_FROM$. A simple approach is to loop through all the "marked" entries of the array $RECV\_FROM$ and use the single-sided MPI data transfer function `MPI_Get(···)` to fetch the data from the corresponding processes. While being convenient, the downside of this approach is that not all MPI implementations provide good support for one-sided communication functions, such as `MPI_Get(···)`. In addition, the performance of one-sided communication functions could vary significantly between

[3]The approach that we use here is very similar to the approach used to check the stopping criterion in pricing multi-asset American options discussed in Subsection 6.4.2.

different underlying computer systems. As a result, we also investigate an alternative to this approach, in which we use conventional MPI functions, such as `MPI_Send(···)` and `MPI_Recv(···)` for data exchange. We discuss below the alternative approach in detail.

At the beginning of Stage I, each process knows the ranks of all processes from which it receives data (stored in $RECV\_FROM$), but not the ranks of those processes which need its data. In order to use the functions `MPI_Send(···)` and `MPI_Recv(···)`, which are more efficient than `MPI_Get(···)`, each process needs to also know the ranks of those processes which need its data. To handle this issue, we have an array of size $w+1$ in the host memory, referred to as the array $SEND\_TO$. At the beginning of Stage I, we perform the following steps:

1. use `MPI_Gather(···)` to gather the arrays $RECV\_FROM$ in the root process (process 0); these collected arrays can be viewed as a $(w+1)\times(w+1)$ matrix, with the $\bar{i}$-th row corresponding to the array $RECV\_FROM$ of the process $\bar{i}$, $\bar{i}=0,\ldots,w$;

2. transpose this matrix on process 0;

3. use `MPI_Scatter(···)` to send each row of the transposed matrix to the array $SEND\_TO$ of the corresponding process.

Each process now can easily perform data exchange using `MPI_Send(···)` and `MPI_Recv(···)` by looping through all the "marked" entries of the arrays $RECV\_FROM$ and $SEND\_TO$.[4]

4.4.2 Interpolation

At the end of Stage I, process i has in its host memory all the vectors of data it needs to carry out the interpolation scheme (3.18). These vectors are then copied from the process'

[4]Note that, instead of the current approach, sophisticated parallel matrix transposition techniques using certain MPI functions, such as MPI_Alltoall(···), could be considered. However, since the matrix to be transposed is of size $(w+1)\times(w+1)$, where w is relatively small (about 40 in our experiments), we did not consider sophisticated techniques for the matrix transposition in these steps, as we believe the timing results would not be improved noticeably.

host memory to the global memory during the first phase of Stage II (see Figure 4.2.2). By the data exchange procedure described in Stage I, the vectors are stored in a buffer in an 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 i from other processes during Stage I. We denote the sorted list of k vectors, including the vector $\mathbf{u}_{i,\alpha^+}$, by

$$\{\mathbf{u}_{i_1,\alpha^+}, \ldots, \mathbf{u}_{i_k,\alpha^+}\},$$

where i_j, $j = 1, \ldots, k$, are in $\{i, \ldots, w\}$, with $i_1 = i$, and $i_1 < i_2 < \cdots < i_k$.

For a GPU-based implementation of the interpolation procedure, we adopt the same partitioning approach and assignment of gridpoints to threads as in the "marking" phase described in the previous subsection (see Figure 4.4.1). The interpolation can be achieved by a k-iteration loop in the kernel. Each thread in a threadblock first computes the quantity $\bar{a}_i$ associated with it using (3.17). During the jth iteration of the k-iteration loop in the kernel, each threadblock

1. first checks if $a_{i_j} \leq \bar{a}_i \leq a_{i_{j+1}}$,

2. if yes, it performs interpolation using the corresponding values in $\mathbf{u}_{i_j,\alpha^+}$ and $\mathbf{u}_{i_{j+1},\alpha^+}$.

Note that the partitioning approach and assignment of gridpoints to threads that we adopt here allow for coalesced loading of data in Step 2 above, i.e. the loading of entries of the vectors $\mathbf{u}_{i_j,\alpha^+}$ and $\mathbf{u}_{i_{j+1},\alpha^+}$ (see the discussion on memory coalescing towards the end of Subsection 4.3.1).

4.5 Other GPU-based Implementations

Other related GPU-based implementations include (i) enforcing the early exercise condition (3.8) in pricing the associated Bermudan swaption (see Figure 4.2.1), (ii) incor-

porating the PRDC coupons in pricing "vanilla" and FX-TARN PRDC swaps (see Figures 4.2.1 and 4.2.2), and (iii) incorporating the funding payment in pricing FX-TARN PRDC swaps (see Figure 4.2.2). The implementation in these cases is straightforward, since each thread of a threadblock can work independently from the others, i.e. no communication between threads is required. We use the same partitioning approach and assignment of gridpoints to threads employed in the previous section. As mentioned earlier, this approach allows for coalesced loadings of data from the global memory.

4.6 Single-GPU Implementation for FX-TARN PRDC Swaps

Although the primary focus of this section is on the development of a numerical methods for pricing Fx-TARn PRDC swaps that can be implemented effectively on clusters of GPUs, for comparison purposes, we also develop single-GPU based numerical methods for these derivatives. Over each time period $[T_{(\alpha-1)^+}, T_{\alpha^-}]$ of the swap's tenor structure, the computation can be divided into two phases: (i) the interpolation phase using data available from the previous time period, i.e. the vectors $\mathbf{u}_{y,\alpha^+}$ corresponding to a_y, $y = 0, \ldots, w$, and (ii) the PDE solution phase.

The interpolation for the time period $[T_{(\alpha-1)^+}, T_{\alpha^-}]$ on the GPU is carried out in the following steps. First, we copy all $w+1$ vectors to the device memory. For each of the vectors, we adopt the partitioning approach illustrated in Figure 4.4.1. To perform interpolation, we adopt a $(w+1)$-iteration loop, during the yth iteration of which, $y = 0, \ldots, w$, a kernel is launched for the interpolation of the vector $\mathbf{u}_{y,\alpha^+}$. This approach suggests a $(w+1-y)$-iteration loop in each of the kernels launched; during the $\bar{i}$th iteration, $\bar{i} = y, \ldots, w$, each thread of a threadblock works with the corresponding pairs of values in the vectors $\mathbf{u}_{\bar{i},\alpha^+}$ and $\mathbf{u}_{\bar{i}+1,\alpha^+}$.

In the second phase, a total of $w + 1$ PDEs are processed sequentially using data obtained from the interpolation phase, with each of the PDEs being solved using the GPU-based numerical methods discussed in earlier sections for a pricing subproblem of a FX-TARN PRDC swap.

Chapter 5

Numerical Results for PRDC Swaps

In this chapter, we first present selected numerical results to demonstrate the effectiveness of the GPU-based parallel pricing methods applied to PRDC swaps. We then discuss pricing results and modeling issues of PRDC swaps.

This chapter is organized as follows. In Section 5.1, we briefly describe common statistics collected from the experiments and present the computation and model parameters. The performance results illustrating the efficiency of the GPU-based parallel numerical methods are presented in Section 5.2. In Section 5.3, we discuss the pricing results and study the effects of the FX volatility skew on the prices of PRDC swaps.

5.1 Statistics Collected, Model and Computation Parameters

5.1.1 Statistics Collected

For all the experiments on GPUs, we used the CUDA 3.2 driver and toolkit version January 2011. The CPU code developed for experiments in the thesis is non-multithreaded, and only one CPU core was employed for the experiments with the CPU code.

The timing statistics collected for the experiments include the following quantities. The CPU and GPU computation times, respectively denoted by "CPU time" and "GPU time", measure the total computation times in seconds (s.), i.e. the entire end-to-end computation. If only CUDA is involved, these quantities are obtained using the CUDA timing functions `cutStartTimer(···)` and `cutStopTimer(···)`. In pricing FX-TARN PRDC swaps on the GPU cluster, the total computation time "GPU time", also reported in seconds (s.), is obtained using the MPI timing function `MPI_Wtime(···)`. In this case, the GPU times also include the total times required for both the data exchange between processes and interpolation. In any case, the GPU computation times include the overhead for memory transfers from the CPU to the device memory. The quantity "speedup" is defined as the ratio of the CPU time over the corresponding GPU time.

We report the quantity "value" which is the value of the financial instruments. In pricing PRDC swaps, this quantity is expressed as a percentage of the notional N_d. We also compute the quantity "$\log_\eta$ ratio" which provides an estimate of the convergence rate of the algorithm by measuring the true errors, referred to as "error", if analytical or very accurate numerical solutions are known, or by measuring the differences in prices on successively finer grids, referred to as "change", when an accurate reference solution is not available. More specifically, when an accurate reference solution is not available, this quantity is defined by

$$\log_\eta \text{ratio} = \log_\eta \Big(\frac{u_{approx}(\Delta x) - u_{approx}(\frac{\Delta x}{\eta})}{u_{approx}(\frac{\Delta x}{\eta}) - u_{approx}(\frac{\Delta x}{\eta^2})} \Big), \tag{5.1}$$

where $u_{approx}(\Delta x)$ is the approximate solution computed with discretization stepsize Δx. Otherwise, the formula

$$\log_\eta \text{ratio} = \log_\eta \Big(\frac{u_{exact} - u_{approx}(\Delta x)}{u_{exact} - u_{approx}(\frac{\Delta x}{\eta})} \Big)$$

can be used, where u_{exact} is the accurate reference solution. For second-order methods, such as those considered in this thesis, the quantity $\log_\eta$-ratio is expected to be about 2.

5.1.2 Model and Computation Parameters

As parameters to the model, we consider the same interest rates, correlation parameters, and the local volatility function as given in [58]. 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 $s(0) = 105.00$. The parameters $\xi(t)$ and $\varsigma(t)$ for the local volatility function are assumed to be piecewise constant and given in Table 5.1.1.

	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%

Table 5.1.1: The parameters $\xi(t)$ and $\varsigma(t)$ for the local volatility function (2.11). (Table C in [58].)

Note that the forward FX rate $F(0,t)$ defined by (2.5) and $\theta_i(t)$, $i = d, f$, in (2.10), and the domestic LIBOR rate (2.7) are fully determined by the above information [4, 8]. We consider the tenor structure (2.4) 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:

- Pay annual PRDC coupons and receive annual domestic LIBOR payments.

- Standard structure, i.e. $b_f = 0$, $b_c = +\infty$. The scaling factor $\{f_\alpha\}_{\alpha=1}^{\beta}$ is set to the forward FX rate $F(0, T_\alpha)$.

- The domestic and foreign coupons are chosen to provide three different levels of leverage: low ($c_d = 2.25\%, c_f = 4.50\%$), medium ($c_d = 4.36\%, c_f = 6.25\%$), high ($c_d = 8.1\%, c_f = 9.00\%$).

- All three exotic features, namely Bermudan cancelable, knockout and FX-TARN, are considered, details each of which are given below.

- The Bermudan cancelable feature allows the issuer to cancel the swap on any of the dates T_α, $\alpha = 1, \dots \beta$.

- The knockout feature is an up-and-out FX-linked barrier and the fixed barrier is set to $B_u = 110.25, 120.75$ and 131.25 for the low-, medium- and high-leverage levels, respectively. The knockout swap pre-maturely terminates on the first date T_α, $\alpha = 1, \dots \beta$, on which $s(T_\alpha) \geq B_u$.

- In the FX-TARN swaps, the values of the total coupon amount cap A_c is set to $A_c = 50\%$, 20%, and 10% of the notional for the low-, medium-, and high-leverage levels, respectively. The FX-TARN swap is terminated on the first date T_α, $\alpha = 1, \dots \beta$, when the accumulated PRDC coupon amount, including the coupon amount scheduled on that date, reaches the target cap A_c.

The truncated computational domain Ω_s is defined by setting $s_\infty = 3s(0) = 315$, $r_{d,\infty} = 3r_d(0) = 0.06$, and $r_{f,\infty} = 3r_f(0) = 0.15$. The grid sizes and the number of timesteps reported in the tables in this chapter are for each time period of the Table 5.1.1. All computations for PRDC swaps are carried out in single-precision.

With the above choice of the truncated computational domain and for all grid sizes considered, there is a gridpoint at the spot value in each spatial dimension, i.e. at $s(0)$, $r_d(0)$ and $r_f(0)$. Also, for all grid sizes considered for the knockout PRDC swaps, the fixed FX-linked barrier B_u is one of the midpoints of the grid in the spot FX rate direction. That is, we use the grid shifting strategy. This is desirable for the accuracy of the numerical methods (see Remark 3.5.1 on page 47).

5.2 Performance Results

In this section, we focus on the timing results to illustrate the effectiveness of our GPU-based numerical methods applied to pricing PRDC swaps. A detailed analysis of the pricing results is provided in the next section.

In Subsection 5.2.1, we focus on the GPU versus CPU performance comparison for Bermudan cancelable and knockout PRDC swaps. 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, as discussed in Section 4.6, for comparison purposes. The details of the GPU versus GPU cluster performance comparison in pricing FX-TARN PRDC swaps are presented in Subsection 5.2.2.

5.2.1 GPU versus CPU Performance Comparison

				underlying PRDC swap				cancelable PRDC swap			
l	n	p	q	value	CPU	GPU	speed	value	CPU	GPU	speed
(τ)	(s)	(r_d)	(r_f)	(%)	time (s.)	time(s.)	up	(%)	time (s.)	time (s.)	up
4	72	24	24	-11.098	5.3	1.7	3.1	11.313	10.7	1.7	6.2
8	144	48	48	-11.106	90.7	5.5	16.5	11.285	182.8	5.5	33.2
16	288	96	96	-11.107	1583.8	53.5	29.9	11.278	3168.3	53.5	58.9

Table 5.2.1: Computed prices and timing results for the underlying PRDC swap and Bermudan cancelable PRDC swap for the low-leverage case.

l (t)	n (s)	p (r_d)	q (r_f)	value (%)	CPU time (s.)	GPU time (s.)	speed up
2	30	12	12	0.825	0.4	0.4	1.0
6	90	36	36	1.304	27.9	2.1	13.1
18	270	108	108	1.349	2215.5	67.5	32.2

Table 5.2.2: Computed prices and timing results for the knockout PRDC swap for the low-leverage case using the grid shifting technique.

Tables 5.2.1 and 5.2.2 present some selected numerical and timing results for Bermudan cancelable and knockout PRDC swaps, respectively, for the low-leverage case under the FX skew model considered in this thesis. The timing results for the medium- and high-leverage cases are approximately the same, and hence omitted. The CPU and GPU computation times measure the total computation times in seconds (s.) required to solve the PDE over 29 periods of the tenor structure (also see Remark 2.2.1 on page 23).

For Bermudan cancelable PRDC swaps, the CPU times for pricing a Bermudan cancelable PRDC swap include the times needed for computing the underlying PRDC swap which is a "vanilla" 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 computed on two NVIDIA Tesla T10 GPUs simultaneously, one of which was used for pricing the underlying PRDC swap. The GPU times for pricing Bermudan 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 5.2.1, 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 Bermudan cancelable PRDC swaps, we obtained speedups that are approximately double those obtained when pricing the underlying alone, 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.

For the knockout PRDC swaps, the timing results behave more or less similar to those obtained when pricing the underlying on a single GPU, with the asymptotic speedup being about 32 for the largest grid we considered.

5.2.2 GPU Cluster versus Single-GPU Performance Comparison

In this subsection, we discuss the single-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 "GPU-MPI time" respectively denote the total computation times, in second (s.), on a single GPU and on the GPU cluster with specifications as in Subsection 4.1.2 using MPI. Note that, as discussed towards the end of Subsection 4.4.1, when pricing FX-TARN PRDC swaps on a GPU cluster, we consider two different schemes for the data exchange between processes: (i) the conventional functions `MPI_Send(···)`/`MPI_Recv(···)` and (ii) the one-sided communication function `MPI_Get(···)`. The resulting two implementations on a cluster are respectively referred to as "GPU-MPI-I" and "GPU-MPI-II".

The quantity "GPU-MPI speed up" is defined as the ratio of the "GPU time" over the respective "GPU-MPI time". The quantity "GPU-MPI efficiency" is defined as

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

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

Table 5.2.3 presents some selected numerical and timing results for FX-TARN PRDC swaps for the low-leverage case with the target cap $A_c = 50\%$. The timing results for the medium- and high-leverage cases 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 respective MPI functions. It is evident that both GPU-MPI implementations on the cluster are significantly more efficient than the single-GPU implementation, with the asymptotic speedups being about 31 and 28 for the GPU-MPI-I and GPU-MPI-II implementations, respectively, when using 40 GPUs (20 nodes) of the cluster. (Note that a total of $(w + 1)$ GPUs are used).

Comparing the GPU-MPI-I and the GPU-MPI-II implementations, it is obvious that the GPU-MPI-I implementation is more efficient than the GPU-MPI-II implementation. In particular, for the largest grid considered, the GPU-MPI-II implementation requires about 22% more time for the data exchange than does the GPU-MPI-I implementation (Table 5.2.3, 23.1 (s.) versus 16.3 (s.)).

l	n	p	q	w	value	GPU	GPU-MPI-I			GPU-MPI-II		
						time	time	speed-	effi-	time	speed-	effi-
(τ)	(s)	(r_d)	(r_f)	(a)	(%)	(s.)	(s.)	up	ciency	(s.)	up	ciency
4	72	24	24	39	-4.521	50.5	2.3 (0.5)	21.9	55%	2.4 (0.5)	21.0	53%
8	144	48	48	39	-4.440	227.5	8.4 (2.2)	27.1	68%	9.9 (3.2)	22.9	59%
16	288	96	96	39	-4.414	2339.3	72.8 (16.3)	32.3	81%	80.1 (23.1)	29.9	72%

Table 5.2.3: Computed prices and timing results for the FX-TARN PRDC swaps for the low-leverage case. The target cap is $A_c = 50\%$. The times in the brackets are those required for data exchange between processes using different MPI functions.

It is important to emphasize that, for both GPU-MPI implementations, the GPU-MPI efficiency increases with finer grid sizes (Table 5.2.3, from 55% to 81% for GPU-MPI-I, and from 53% to 72% for GPU-MPI-II). This is to be expected, since a fixed number of GPUs, i.e. 40 GPUs, is used for all the experiments, whereas the problem size is increasing, allowing the GPUs to be used more efficiently.

5.3 Analysis of Pricing Results and Effects of FX Volatility Skew

5.3.1 Analysis of Pricing Results

In this subsection, we discuss the computed prices for "vanilla", Bermudan cancelable, knockout and FX-TARN PRDC swaps. Recall that in Remark 2.2.2 on page 23, we mentioned that there is a settlement in the form of an initial fixed-rate coupon exchanged between the issuer and the investor at time T_0. This signed coupon is typically the value at time T_0 of the swap to the issuer. (Note that we price the deal from the perspective of the issuer.) More specifically, a negative value for the swap is the price that the investor has to pay to the coupon issuer to enter into a PRDC swap (a fund inflow for the issuer and a fund outflow for the investor). A positive value of the PRDC swap is the level of the initial fixed coupon that the issuer is willing to pay to the investor (a fund inflow for the investor and a fund outflow for the issuer).

Bermudan Cancelable PRDC swaps

In Table 5.3.1, we present pricing results for the underlying and Bermudan cancelable swaps. The numerical results indicate second-order convergence is achieved for the ADI scheme, as expected. With respect to the prices of the underlying PRDC swap, for the low-, medium- and high-leverage cases under the FX skew model, the investor should

leverage level	l (τ)	n (s)	p (r_d)	q (r_f)	underlying value (%)	underlying change	underlying $\log_2$ ratio	Bermudan cancelable value (%)	Bermudan cancelable change	Bermudan cancelable $\log_2$ ratio
low	2	36	12	12	-11.070			11.415		
	4	72	24	24	-11.098	2.8e-4		11.313	1.3e-3	
	8	144	48	48	-11.106	8.3e-5	1.8	11.285	2.8e-4	1.9
	16	288	96	96	-11.107	1.3e-5	2.5	11.278	6.8e-5	2.1
medium	2	36	12	12	-12.800			13.928		
	4	72	24	24	-12.715	8.6e-4		13.907	2.1e-04	
	8	144	48	48	-12.693	2.2e-4	2.0	13.903	4.6e-05	2.2
	16	288	96	96	-12.686	6.6e-5	1.8	13.902	9.8e-06	2.2
high	2	36	12	12	-11.363			19.546		
	4	72	24	24	-11.153	2.1e-3		19.617	7.1e-4	
	8	144	48	48	-11.102	5.2e-4	2.0	19.635	1.8e-4	2.0
	16	288	96	96	-11.087	1.5e-4	1.9	19.639	4.9e-5	1.9

Table 5.3.1: Values of the underlying PRDC swap and Bermudan cancelable PRDC swap for various leverage levels with FX skew model.

pay a net coupon of about 11.107%, 12.686% and 11.087%, respectively, of the notional to the issuer. Regarding the prices of the Bermudan cancelable PRDC swap, for the low-, medium- and high-leverage cases under the FX skew model, the issuer should pay a net coupon of about 11.278%, 13.902% and 19.639%, respectively, of the notional, to the investor. Of course, in these cases, the issuer would prefer to pay less and keep the difference as profit. Among the three leverage cases, the high-leverage case is more attractive to the investor, due to the high initial coupon paid by the issuer. (See the discussion on the economics of PRDC swaps with exotic features on page 26.)

Knockout PRDC swaps

In Table 5.3.2, we present pricing results for the knockout PRDC swap for various leverage levels. Note that, tripling the number of gridpoints ($\eta = 3$) of a coarser grid having the fixed FX-linked barrier B_u as a midpoint ensures that the resulting finer grid has the same property. We expect the quantity $\log_3$ ratio to be about 2 for a second-order discretization method as the grids are refined in this fashion. The numerical results indicate second-order convergence is achieved for the HV scheme, as expected.

leverage level	l (τ)	n (s)	p (r_d)	q (r_f)	value (%)	change	$\log_3$ ratio
low	2	30	12	12	0.825		
	6	90	36	36	1.304	4.8e-03	
	18	270	108	108	1.349	4.5e-04	2.1
medium	2	30	12	12	1.390		
	6	90	36	36	2.012	6.2e-03	
	18	270	108	108	2.093	8.0e-04	1.9
high	2	30	12	12	4.972		
	6	90	36	36	5.429	4.6e-03	
	18	270	108	108	5.495	6.6e-04	1.8

Table 5.3.2: Computed prices and convergence results for the knockout PRDC swap for various leverage levels under the FX skew model. The grid shifting technique is used.

To show the effect of the grid shifting technique on the convergence and accuracy of the numerical methods, we carried out experiments with different grids which do not have B_u as a midpoint, but rather as a gridpoint, in the spot FX rate direction. In these experiments, the coarser grids having the fixed FX-linked barrier B_u as a gridpoint are refined by doubling the number of gridpoints ($\eta = 2$).

The numerical results for these experiments are presented in Table 5.3.3. It is evident that, while the HV scheme has the expected quadratic convergence for smooth problems, only linear convergence is observed here, i.e. the observed $\log_2$ ratio is about 1 instead of 2. In addition, it is also evident that in this case the accuracy of the computed solutions deteriorates significantly as compared to that obtained with the grid shifting technique. This emphasizes the importance of handling appropriately the discontinuities in the terminal conditions on each date of the tenor structure of the knockout PRDC swaps, as discussed in Remark 3.5.1.

leverage level	l (τ)	n (s)	p (r_d)	q (r_f)	value (%)	change	$\log_2$ ratio
low	4	60	24	24	0.935		
	8	120	48	48	0.892	4.2e-04	
	16	240	96	96	0.871	2.0e-04	1.1
medium	4	60	24	24	1.995		
	8	120	48	48	1.930	6.5e-04	
	16	240	96	96	1.898	3.2e-04	1.1
high	4	60	24	24	5.121		
	8	120	48	48	5.082	3.9e-04	
	16	240	96	96	5.067	1.8e-04	1.0

Table 5.3.3: Computed prices and convergence results for the knockout PRDC swap for various leverage levels with the FX skew model without the grid shifting technique.

With respect to the computed prices, for the low-, medium- and high-leverage cases under the FX skew model, the issuer of a knockout PRDC swap should pay a net coupon of about 1.349%, 2.093% and 5.495% of the notional to the investor. Of course, the issuer would prefer to pay less and keep the difference as profit. Among the three leverage cases,

similar to the results of the Bermudan cancelable PRDC swap discussed earlier, the high-leverage case is more attractive to the investor, due to the high initial coupon paid by the issuer.

FX-TARN PRDC swaps

leverage level	A_c	l (τ)	n (s)	p (r_d)	q (r_f)	w (a)	value (%)	change	$\log_2$ ratio
low	50%	2	36	12	12	39	-4.787		
		4	72	24	24	39	-4.521	2.7e-03	
		8	144	48	48	39	-4.440	8.2e-04	1.7
		16	288	96	96	39	-4.414	2.5e-04	1.7
medium	20%	2	36	12	12	39	3.348		
		4	72	24	24	39	3.628	2.8e-03	
		8	144	48	48	39	3.710	8.2e-04	1.8
		16	288	96	96	39	3.736	2.5e-04	1.7
high	10%	2	36	12	12	39	18.067		
		4	72	24	24	39	18.458	3.9e-03	
		8	144	48	48	39	18.567	1.1e-03	1.9
		16	288	96	96	39	18.599	3.2e-04	1.8

Table 5.3.4: Values of the FX-TARN PRDC swap for various leverage levels under the FX skew model. The total coupon amount cap A_c is set to $A_c = 50\%$, 20%, and 10% of the notional for the low-, medium-, and high-leverage levels, respectively.

In Table 5.3.4, we present pricing results for the FX-TARN PRDC swaps for various levels of leverage and values of the target cap A_c. In all cases, the numerical results exhibit second-order convergence, as expected from the ADI timestepping methods and

the interpolation scheme. With respect to the prices of the FX-TARN PRDC swap, for the low-leverage level, the investor should pay a net coupon of about 4.414% of the notional to the issuer. (Note the negative values in this case.) However, for the medium- and high-leverage cases, the issuer should pay the investor a net coupon of about 3.735% and 18.599%, respectively, of the notional to the investor. Obviously, the high-leverage case is much more attractive to the investor, than all other leverage levels, due to the high initial coupon paid by the issuer. On the other hand, the low-leverage case with $A_c = 50\%$ is certainly not attractive to the investor, because the investor ends up paying the initial coupon.

In Table 5.3.5, we presents the prices of the FX-TARN PRDC swap for various values of the target caps A_c obtained using the finest mesh in Table 5.3.4.

leverage	A_c			
level	10%	20%	50%	80%
low	5.319	1.199	-4.414	-6.907
medium	8.702	3.735	-3.189	-6.412
high	18.599	14.821	8.931	5.899

Table 5.3.5: Values of the FX-TARN PRDC swap for various target cap levels A_c and various leverage levels for the FX skew model using the finest mesh in Table 5.3.4.

In Table 5.3.5, we observe that the price of the FX-TARN PRDC swap is a decreasing function of the target cap A_c. More specifically, a smaller value of the target cap A_c results in a larger (more positive) price of the FX-TARN PRDC swap, indicating that the issuer pays the investor the initial coupon (e.g. see the low-leverage case with $A_c = \{10\%, 20\%\}$). On the other hand, if the target cap A_c is large enough, the price could become negative, i.e. the investor pays the issuer the initial coupon (e.g. see the low-leverage case with $A_c = \{50\%, 80\%\}$). This behavior of the prices of the FX-TARN PRDC swaps are

expected, since, the smaller the target cap is, the higher the leverage of the swap is (from the perspective of the investor). With a larger value of the target cap A_c, the underlying PRDC swap is expected to be terminated later than with a smaller value of A_c. As a result, in this case, the FX-TARN PRDC swap tends to behave like a "vanilla" PRDC swap, which explains why the price of the FX-TARN PRDC swap is a decreasing function of the target cap A_c.

Another observation is that, for a fixed value of the target cap A_c, the the price of the FX-TARN PRDC swap is an increasing function of the leverage level. This observation is consistent with the results for PRDC swaps with Bermudan cancelable and knockout features presented earlier.

5.3.2 Effects of the FX Volatility Skew

To investigate the effects of the FX skew on PRDC swaps, we compare prices of the PRDC swap under the FX skew model with those obtained using the log-normal model, as suggested in [58]. In the log-normal model, the local volatility function is a deterministic function of the time variable t only, and not of the spot FX rate s. To this end, we used the parametrization as in (2.11) but independent of $s(t)$ for the log-normal local volatility function, and calibrated it to the same at-the-money FX option data that were used for the calibration of the FX skew model (Table A of [58]). The numerical results for the log-normal and the FX skew models using the finest mesh in Tables 5.3.1, 5.3.2 and 5.3.4 are presented in Table 5.3.6.

Before we discuss the effects of the FX volatility skew, we need to draw the reader's attention to important details presented in Remark 5.3.1 below.

Remark 5.3.1. Recall that the issuer pays PRDC coupons, the rates of which can be viewed as call options on the spot FX rate, as indicated by the coupon rate formula (2.9). It is important to emphasize that due to the interest rate differential between JPY and

USD, the forward FX curve is strongly downward sloping. As a result, the forward FX rate $F(0, T_\alpha)$, defined by formula (2.5), is considerably smaller than the spot FX rate $s(T_\alpha)$. For the low-, medium-, and high-leverage cases, the strike $e_\alpha = \frac{f_\alpha c_d}{c_f}$ is set to 50%, 70% and 90% of $f_\alpha \equiv F(0, T_\alpha)$, respectively, hence is significantly less than $s(T_\alpha)$. As a result, the PRDC coupon rates defined by (2.9) can be viewed as low-strike FX call options. Thus, the coupon issuer in a PRDC swap essentially shorts a collection of FX call options with low strikes. This observation is very important for understanding the effects of the FX volatility skew on the prices of PRDC swaps discussed below.

Underlying PRDC Swaps

First, let us study the effect of the FX skew on the underlying swaps. As shown in Table 5.3.6, all the underlying swaps have roughly the same value under the log-normal model across leverage levels, and the 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. This pushes down the value of the underlying swap for the issuer, due to the short position (also see Remark 5.3.1).

It is interesting, however, to note that the effect of the FX skew is not uniform across the leverage levels. The effect seems most pronounced for the medium-leverage PRDC swaps (a difference of -3.048 as opposed to -2.094 and -1.254 for low- and high-leverage swaps, respectively). An explanation for this observation is that the total effect of the FX skew on the prices of a "vanilla" PRDC swap 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

leverage		FX skew	log-normal	FX skew - log-normal
level	$\frac{c_d}{c_f}$	(%)	(%)	(%)
		underlying		
low	50%	-11.107	-9.012	-2.094
medium	70%	-12.686	-9.638	-3.048
high	90%	-11.087	-9.833	-1.254
		Bermudan cancelable		
low	50%	11.278	13.308	-2.030
medium	70%	13.902	16.899	-2.998
high	90%	19.639	22.938	-3.299
		knockout		
low	50%	1.349	3.836	-2.487
medium	70%	2.093	7.038	-4.945
high	90%	5.495	12.308	-6.813
		FX-TARN		
low	50%	-4.414	-2.733	-1.681
medium	70%	3.735	6.443	-2.708
high	90%	18.599	22.101	-3.502

Table 5.3.6: Computed prices for the underlying PRDC swap and Bermudan cancelable, knockout, and FX-TARN PRDC swaps for various leverage levels with the FX skew model ("FX skew") and the log-normal model ("log-normal") using the finest mesh in Tables 5.3.1, 5.3.2 and 5.3.4. For the knockout PRDC swap, the grid shifting technique is used.

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 [32]. 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.

Bermudan Cancelable

We now investigate the effect of the FX skew on the Bermudan cancelable swap. The difference between the FX skew model prices and the log-normal model prices is uniformly decreasing across the leverage levels (-2.030%, -2.998%, -3.299%). These differences are quite significant and they indicate that, under the FX skew model, the initial net coupons paid (by the issuer) to the investor are smaller than those paid under the log-normal model. These changes in values of the swap can be viewed as profits booked by the issuer.

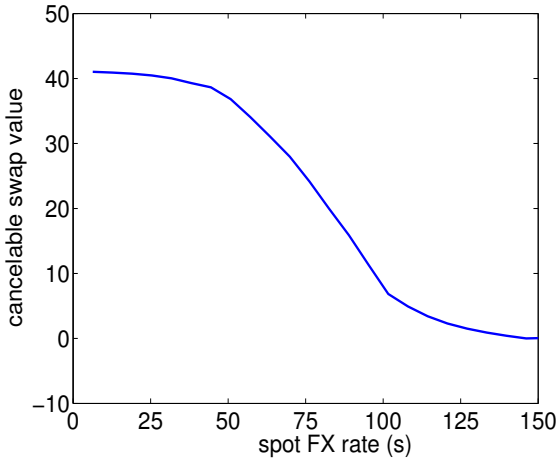


Figure 5.3.1: Values of the Bermudan cancelable PRDC swap, in percentage of N_d, as a function of the spot FX rate at time $T_\alpha = 5$ with high-leverage coupons.

To better understand the effects of the FX skew on the value of the Bermudan cancelable PRDC swap, we investigate the value of the Bermudan cancelable swap at an intermediate date of the tenor structure as a function of the spot FX rate on that date.

In Figure 5.3.1, a sample plot of such a function for the high-leverage case immediately after the exchange of fund flows scheduled at time $T_\alpha = 5$, i.e. at time T_{5^+}, is given. Note that the forward FX rate $F(0, T_\alpha)$ is about 90.3 on that date, as computed by the formula (2.5).

As shown in Figure 5.3.1, in the region where the spot FX rate s is less than the forward FX rate ($s < 90.3$), it is evident that the value function is positive and concave-down, i.e. it has negative gamma. This agrees with the interpretations that (i) the swap is not canceled due to low spot FX rates, and that (ii) the issuer has a short position in low-strike FX call options (see Remark 5.3.1).[1] However, if the spot FX rate is high enough, it would be optimal for the issuer to cancel, an observation that is reflected by the fact that the value function now becomes concave-up, i.e. it has positive gamma, due to the long position in high-strike FX call options (options to cancel). As a result, the profile of a Bermudan cancelable PRDC swap is similar to a bear spread created by call options: going short several low-strike call options and going long the same number of higher-strike options with the same maturity (see Figure 5.3.2), which is known to be very sensitive to the volatility skew. More specifically, since the FX volatility skew increases the volatility of the low-strike options, the value of a Bermudan cancelable PRDC swap in the concave-down part is pushed down for the issuer, due to the short position in low-strike FX call options. That is, the concave-down part is valued lower for the issuer under the FX skew model than under a log-normal model. On the other hand, due to lower volatility of the FX volatility skew for the high-strike options (options to cancel), the concave-up part is also valued lower for the issuer under the FX skew model than in a log-normal model, taking into account the fact that the issuer has a long position in this part.

Next, we investigate the effects of the different leverage levels on the values of Bermu-

[1] The gamma of a short position is always negative, while that of a long position is always positive.

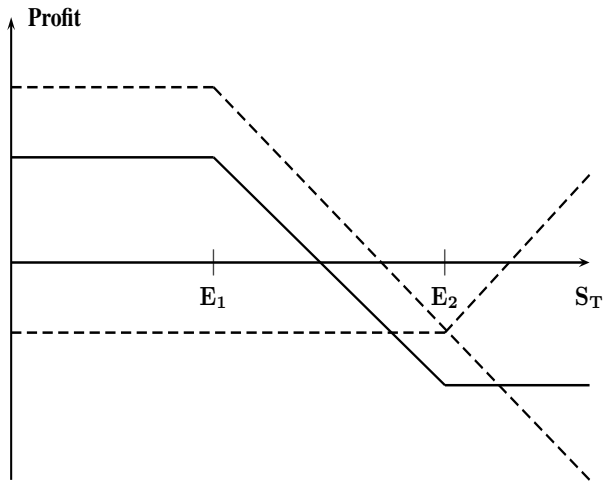


Figure 5.3.2: Example of a bear spread created using call options.

dan cancelable swaps. First, as evident from Table 5.3.6, under both the FX skew model and the log-normal model, the values of PRDC swaps increase with leverage levels: from 11.278% to 19.639% under the skew model and from 13.308% to 22.938% under the log-normal model. This behavior is expected, due to a positive correlation between the leverage level and the volatility level. In addition, due to the strong sensitivity of the prices of Bermudan cancelable PRDC swaps to the implied volatility, the differences between FX skew and log-normal prices also increase significantly in absolute terms with leverage levels. This is reflected through the changes in values by the amounts -2.030%, -2.998%, and -3.299% for low-, medium-, and high-leverage levels, respectively.

Knockout and FX-TARN PRDC Swaps

We now investigate the effect of the FX skew on the knockout and FX-TARN PRDC swaps. We first look at the knockout PRDC swap. In Figure 5.3.3, we plot values of the knockout PRDC swap immediately after the exchange of fund flows scheduled at time $T_\alpha = 3$, i.e. at time T_{3^+}, as a function of the spot FX rate of that date. The forward FX rate $F(0, T_\alpha)$ is about 95.6 on that date. When the spot FX rate is smaller

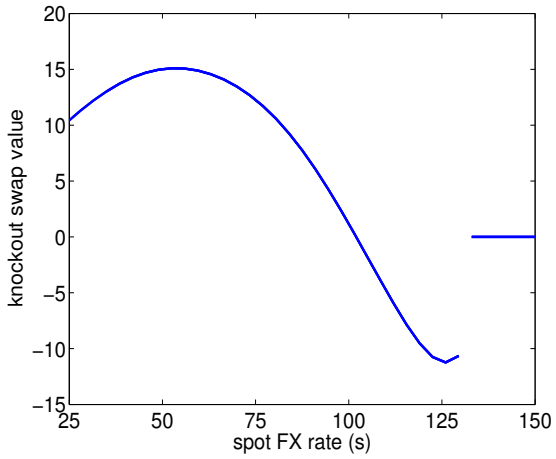


Figure 5.3.3: Values of the knockout PRDC swap, in percentage of N_d, as a function of the spot FX rate at time $T_\alpha = 3$ with high-leverage coupons. The barrier B_u is 131.25.

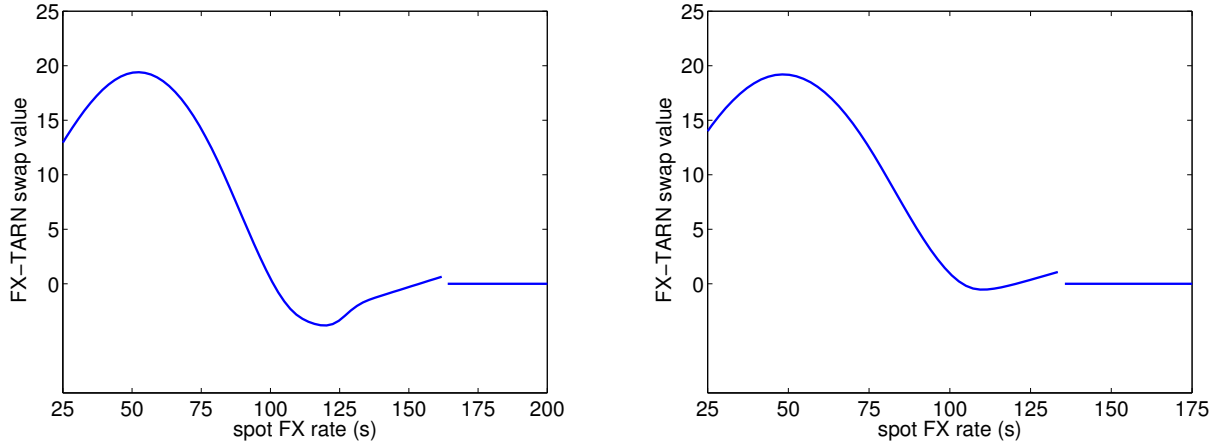
than the forward FX rate ($s < 95.6$), we observe that the value function is positive and concave down, which is consistent with the observations in the case of Bermudan cancelable swaps. However, in the region where the spot FX rate becomes larger than the forward FX rate and tends to the barrier, as evident from Figure 5.3.3, the value function becomes negative and the profile of the value function changes from being concave-down to being concave-up. The reason that the value function becomes negative in this region is high PRDC coupon rates/amounts which are fund outflows from the issuer's perspective. With respect to the change of concavity, the fact that the PRDC swap is knocked out when $s(T_\alpha) \geq B_u$ can be interpreted as the issuer having a long position in high-strike FX call options. As a result, the profile of the value function changes from concave-down to concave-up to reflect a change from a short position in low-strike FX call options to a long position in high-strike FX call options.

With respect to the FX-TARN PRDC swap, in Figure 5.3.4, we plot values of the FX-TARN PRDC swap immediately after the exchange of fund flows scheduled at time $T_\alpha = 3$, i.e. at time T_{3^+}, as a function of the spot FX rate of that. We consider two

cases in which the accumulated PRDC coupon amount immediately before the exchange of fund flows, i.e. at T_{3^-}, are

$$a_{\alpha^-} \equiv a_{3^-} = \{2.5\%, 5.0\%\}.$$

The profile of the value function in these two cases are quite similar to that of the value function of the knockout PRDC swap. More specifically, when $a_{\alpha^-} = 2.5\%$, the computed knockout barrier is 164.0625, whereas, when $a_{\alpha^-} = 5.0\%$, the computed knockout barrier is 137.8125. These values for the barrier are obtained using $l = 8$, $n = 144, p = q = 48$. These observations are expected, since a FX-TARN PRDC swap can be viewed as a knockout PRDC swap, the early termination of which depends on the total accumulated PRDC coupon amount. In addition, it appears that the value function knockout barrier at time T_α is a decreasing function of a_{α^-} (e.g. compare 164.0625 with $a_{\alpha^-} = 2.5\%$ to 137.8125 with $a_{\alpha^-} = 5.0\%$). This observation is consistent with the fact that the target cap A_c is fixed, and hence, the larger the value of a_{α^-} is, the smaller the value of the knockout barrier is.



(a) $a_{\alpha^-} \equiv a_{3^-} = 2.5\%$ (b) $a_{\alpha^-} \equiv a_{3^-} = 5.0\%$

Figure 5.3.4: Values of the FX-TARN PRDC swap, in percentage of N_d, as a function of the spot FX rate at time $T_\alpha = 3$ with high-leverage coupons.

For both knockout and FX-TARN PRDC swaps, due to the change of concavity from being concave-up to being concave-down in the value function as the spot FX rate increases, similar to the Bermudan cancelable swap, the impact of the FX volatility skew on the prices of these derivatives is quite substantial. As evident from Table 5.3.6, for the knockout PRDC swap, the differences between the FX skew model prices and the log-normal model prices are -2.487%, -4.945%, -6.813% for the low-, medium- and high-leverage swaps, respectively. For the FX-TARN PRDC swap, the respective differences are -1.681%, -2.708% and -3.502%. These differences are quite significant and can be viewed as profits to the issuer, if the market-observed FX skew is accurately approximated.

Chapter 6

Multi-asset Options

Options in general, and multi-asset options in particular, are an important and popular class of financial derivatives. These are the building blocks for many financial contracts, the purposes of which range from speculation to hedging. The abundance of these options support the claim that their efficient pricing is important not only to financial institutions, but also to speculators and investors. The purpose of this chapter is to discuss efficient pricing of multi-asset European and American options in the Black-Scholes-Merton framework via a PDE approach. Strong emphasis in this chapter is placed on American options, due to the challenges arising from the early exercise features embedded in these options. Note that this chapter is dedicated to options in the context of single-currency markets, not multi-currency ones as in the case of PRDC swaps.

The remainder of this chapter is organized as follows. Section 6.1 introduces a three-factor log-normal model and the corresponding 3-D Black-Scholes-Merton PDE. We present numerical methods for pricing multi-asset European options in Section 6.2. The pricing of multi-asset American options is discussed in Section 6.3. Section 6.4 presents a GPU-based parallel implementation of the numerical methods. Although we primarily focus on a three-factor model, many of the ideas and results in this chapter can be naturally extended to higher-dimensional applications with constraints.

6.1 The Model and the Black-Scholes-Merton PDE

Let $s_i(t)$, $i = 1, \ldots, 3$, denote the price of the ith underlying asset. Under the risk neutral measure, the underlying asset prices are assumed to follow the log-normal diffusion processes

$$\frac{ds_i(t)}{s_i(t)} = (r - d_i)dt + \sigma_i dW_i(t), \quad i = 1, 2, 3, \tag{6.1}$$

where $W_i(t)$, $i = 1, 2, 3$, are correlated Brownian motions with $dW_i(t)dW_j(t) = \rho_{ij}dt$. Here, r is the constant riskless interest rate; $d_i \geq 0$ is the constant asset dividend yield; σ_i is the constant volatility of the stochastic process for s_i; ρ_{ij} are the correlation factors between dW_i and dW_j satisfying $|\rho_{ij}| \leq 1$ for $i, j = 1, 2, 3$, and $\rho_{ii} = 1$ for $i = 1, 2, 3$. For simplicity, let $\mathbf{s} = (s_1, s_2, s_3)$. The corresponding 3-D Black-Scholes-Merton PDE, which governs the value $z \equiv z(\mathbf{s}, t)$ of an option, such as that of a European option, written on the three assets is given by

$$\frac{\partial z}{\partial t} + \mathcal{L}_b z \equiv \frac{\partial z}{\partial t} + \frac{1}{2}\sum_{i,j=1}^{3} \rho_{ij}\sigma_i\sigma_j s_i s_j \frac{\partial^2 z}{\partial s_i \partial s_j} + r\sum_{i=1}^{3} s_i \frac{\partial z}{\partial s_i} - rz = 0, \tag{6.2}$$

on $0 \leq s_i < \infty$, $i = 1, \ldots, 3$, and $t \in [0, \bar{T})$, where $\bar{T}$ is the maturity time of the option. A derivation of the multi-dimensional Black-Scholes-Merton PDE for an arbitrary number of underlying assets can be found in [43, 73].

Since we solve the PDE (6.2) backward in time from $\bar{T}$ to the current time, t, the change of variable $\tau = \bar{T} - t$ is used. Under this change of variable, the PDE (6.2) becomes

$$\frac{\partial z}{\partial \tau} = \mathcal{L}_b z, \tag{6.3}$$

and is solved forward in τ. To solve the PDE (6.3) numerically by finite difference methods, we must truncate the unbounded domain to finite-sized computational one

$$(s_1, s_2, s_3, \tau) \in [0, s_{1,\infty}] \times [0, s_{2,\infty}] \times [0, s_{3,\infty}] \times [0, \bar{T}] \equiv \Omega_b \times [0, \bar{T}],$$

for appropriately chosen $s_{i,\infty}$, $i = 1, 2, 3$ [71]. We denote by $\partial\Omega_b$ the boundary of the computational domain Ω_b.

For discretization purposes discussed later in the chapter, let the number of subintervals be $\bar{n}+1$, $\bar{p}+1$ and $\bar{q}+1$ in the s_1-, s_2- and s_3-directions, respectively. The uniform grid mesh widths in the respective direction are denoted by $\Delta s_1 = \frac{s_{1,\infty}}{\bar{n}+1}$, $\Delta s_2 = \frac{s_{2,\infty}}{\bar{p}+1}$, and $\Delta s_3 = \frac{s_{3,\infty}}{\bar{q}+1}$. Let the time interval $[0, \bar{T}]$, for a given number of timesteps $\bar{l}$, be partitioned via

$$0 = \tau_0 < \tau_1 < \cdots < \tau_{\bar{l}} = \bar{T}, \tag{6.4}$$

with

$$\Delta\tau_m = \tau_m - \tau_{m-1}, m = 1, 2, \ldots, \bar{l}; \quad c_m = \frac{\Delta\tau_m}{\Delta\tau_{m-1}}, m = 2, \ldots, \bar{l}. \tag{6.5}$$

Note that the number of timesteps $\bar{l}$ is fully determined by a timestep size selector (see (6.16)). Unless otherwise stated, assume that the mesh points are ordered in the s_1-, s_2-, then s_3- directions. For use later in the thesis, for basket options, we denote by $w_i > 0$, $i = 1, 2, 3$, the weight of the ith asset in the basket.

6.2 Multi-asset European Options

A European "vanilla" call/put option written on a single asset is a contract that gives the holder the right, but not an obligation, to buy/sell an underlying asset for a predetermined exercise price E, referred to as the strike price, at maturity time $\bar{T}$. A multi-asset call/put European option contract gives the holder the right, but not an obligation, to buy/sell at time $\bar{T}$ a specified basket of more than one underlying asset for a fixed exercise price E. Such options belong to the so-called class of exotic options. Since the price of a European option written on three assets satisfy the 3-D Black-Scholes-Merton PDE (6.3), it can be obtained by solving this PDE with appropriate initial and boundary conditions.

The remainder of this section is organized as follows. In Subsection 6.2.1, as examples, we introduce European rainbow options and European basket options written on three

assets, and their payoff functions. An appropriate choice for the boundary conditions for these options is presented in Subsection 6.2.2. The pricing of multi-asset European options is discussed in Subsection 6.2.3.

For use in this section, we denote by $z = z(\mathbf{s}, \tau)$ the value of a European option written on three assets, and by $z^*(\mathbf{s}) \equiv z(\mathbf{s}, 0)$ the initial condition (payoff function) for the option.

6.2.1 Rainbow and Basket Options

Rainbow Options

Rainbow options can take various forms, but their common characteristic is that their payoff function depends on the underlying assets sorted by their values at maturity. As an example, we consider a European *call-on-minimum* rainbow option, the initial condition (payoff function) of which is

$$z^*(\mathbf{s}) = \max(\min(s_1, s_2, s_3) - E, 0). \tag{6.6}$$

Closed-form solutions of rainbow call/put options on the maximum or minimum on several assets are provided in [40]. Thus, we can use rainbow options as a benchmark solution to test the accuracy of our numerical methods.

Basket Options

The payoff function of a European basket call/put option is typically based on the weighted sum of d assets $s_i, i = 1, \ldots, d$, in the basket. For $d = 3$, the initial condition (payoff function) of a call option is

$$z^*(\mathbf{s}) = \max\Big(\sum_{i=1}^{3} w_i s_i - E, 0\Big). \tag{6.7}$$

6.2.2 Linear Boundary Conditions

It can be shown that, for all the European options considered in the thesis, $z(\mathbf{s}, \tau)$ is approximately linear in s_i at the boundaries $s_i = 0$ and $s_i = s_{i,\infty}$ for $i = 1, 2, 3$. Consequently,

$$\frac{\partial^2 z}{\partial s_i^2}\Big|_{s_i=0} = \frac{\partial^2 z}{\partial s_i^2}\Big|_{s_i=s_{i,\infty}} = 0 \tag{6.8}$$

for $i = 1, 2, 3$. Note that, it can be verified easily from the Black-Scholes-Merton formula that "vanilla" European put and call options written on one asset approximately satisfy (6.8).

We can use (6.8) to derive the so-called *linear boundary conditions*, a commonly used boundary condition type in computational finance (see, for example, [17, 44, 66, 71]). To derive one such linear boundary condition, suppose that $s_1 = 0$ or $s_1 = s_{1,\infty}$, but that $s_2 \in (0, s_{2,\infty})$ and $s_3 \in (0, s_{3,\infty})$. Then, using (6.8), the PDE (6.3) reduces to

$$\frac{\partial z}{\partial \tau} = \sum_{i=2}^{3} (\sigma_i)^2 (s_i)^2 \frac{\partial^2 z}{\partial s_i^2} + \frac{1}{2} \sum_{\substack{i,j=1 \\ i \neq j}}^{3} \rho_{ij} \sigma_i \sigma_j s_i s_j \frac{\partial^2 z}{\partial s_i \partial s_j} + \sum_{i=1}^{3} (r - d_i) s_i \frac{\partial z}{\partial s_i} - rz \tag{6.9}$$

on the open faces of $\partial\Omega_b$ for which $s_1 = 0$ or $s_1 = s_{1,\infty}$, but $s_2 \in (0, s_{2,\infty})$ and $s_3 \in (0, s_{3,\infty})$. A similar linear boundary condition holds on the other open faces of $\partial\Omega_b$.

Similarly, using (6.8), the PDE (6.3) reduces to

$$\frac{\partial z}{\partial \tau} = (\sigma_3)^2 (s_3)^2 \frac{\partial^2 z}{\partial s_3^2} + \frac{1}{2} \sum_{\substack{i,j=1 \\ i \neq j}}^{3} \rho_{ij} \sigma_i \sigma_j s_i s_j \frac{\partial^2 z}{\partial s_i \partial s_j} + \sum_{i=1}^{3} (r - d_i) s_i \frac{\partial z}{\partial s_i} - rz \tag{6.10}$$

on the open edges of $\partial\Omega_b$ for which

1. either $s_1 = 0$ or $s_1 = s_{1,\infty}$,

2. either $s_2 = 0$ or $s_2 = s_{2,\infty}$, and

3. $s_3 \in (0, s_{3,\infty})$.

A similar linear boundary condition holds on the other open edges of $\partial\Omega_b$.

Finally, using (6.8), the PDE (6.3) reduces to

$$\frac{\partial z}{\partial \tau} = \frac{1}{2}\sum_{\substack{i,j=1\\ i\neq j}}^{3} \rho_{ij}\sigma_i\sigma_j s_i s_j \frac{\partial^2 z}{\partial s_i \partial s_j} + \sum_{i=1}^{3}(r-d_i)s_i\frac{\partial z}{\partial s_i} - rz \tag{6.11}$$

at the corners of $\partial\Omega_b$ at which

1. either $s_1 = 0$ or $s_1 = s_{1,\infty}$,

2. either $s_2 = 0$ or $s_2 = s_{2,\infty}$, and

3. either $s_3 = 0$ or $s_3 = s_{3,\infty}$.

Remark 6.2.1. A possible alternative approach to applying the linear boundary conditions on the lower boundaries of Ω_b, i.e. on $\partial\Omega_b$ where $s_i = 0$, $i = 1, 2, 3$, is to substitute $s_i = 0$ directly into the 3-D Black-Scholes-Merton PDE (6.3). This is referred to as *natural boundary condition* [44]. In this approach, each of the coefficients of spatial derivative terms, including the cross-derivative terms, with respect to s_i, $i = 1, 2, 3$, in (6.3) vanishes at $s_i = 0$. As a result, on each of the three open faces of the lower boundaries of Ω_b, only a 2-D Black-Scholes-Merton PDE with respect to the other two spatial variables remains, whereas, on each of the three open edges defined by $s_i = 0$ and $s_j = 0$, $i \neq j$, of the lower boundaries of Ω_b, the boundary condition is simply a 1-D Black-Scholes-Merton PDE with respect to the remaining spatial variable.

6.2.3 Discretization

Let the gridpoint values of a FD approximation to the solution z be denoted by

$$z^m_{i,j,k} \approx \quad z(s_{1i}, s_{2j}, s_{3k}, \tau_m) = \quad z(i\Delta s_1, j\Delta s_2, k\Delta s_3, \tau_m),$$

where $i = 0, \ldots, \bar{n}+1$, $j = 0, \ldots, \bar{p}+1$, $k = 0, \ldots, \bar{q}+1$, $m = 0, \ldots, \bar{l}$.

For the discretization of the space variables in the differential operator $\mathcal{L}_b$ in the interior of the rectangular domain Ω_b, we employ the standard second-order central differences on uniform grids, similar to the schemes (3.1) and (3.2). For brevity, we do not repeat these FD schemes here. (See Appendix C.1 for a derivation of these FD schemes). In addition, due to the linear boundary conditions, we also need to discretize certain PDEs, such as (6.9), (6.10) and (6.11), on the boundary $\partial\Omega_b$ of the computational domain Ω_b. In this case, in order to avoid introducing gridpoints outside Ω_b, we use one-sided FD approximations to derivatives discussed below. For an alternative discretization approach on the boundary, see Remark 6.2.2 on page 125.

One-sided FD Formulas

We use the one-sided forward and backward FD approximations

$$\left.\frac{\partial z}{\partial s_1}\right|_{s_1=0} \equiv \left.\frac{\partial z}{\partial s_1}\right|_{0,j,k}^{m} \approx \frac{z^m_{1,j,k} - z^m_{0,j,k}}{\Delta s_1}, \text{ or} \tag{6.12a}$$

$$\left.\frac{\partial z}{\partial s_1}\right|_{s_1=s_{1,\infty}} \equiv \left.\frac{\partial z}{\partial s_1}\right|_{\bar{n}+1,j,k}^{m} \approx \frac{z^m_{\bar{n}+1,j,k} - z^m_{\bar{n},j,k}}{\Delta s_1} \tag{6.12b}$$

at points on the boundary $\partial\Omega_b$ for which $s_1 = 0$ or $s_1 = s_{1,\infty}$, respectively. The cross derivatives $\dfrac{\partial^2 z}{\partial s_1 \partial s_j}$, $j = 2, 3$, in (6.9) can be discretized by successively applying the one-sided FD for the first derivative in the s_1-direction (scheme (6.12)) and the standard central FD scheme for the first derivative in the other direction (scheme (3.1a)). For instance, applying this discretization technique to $\dfrac{\partial^2 z}{\partial s_1 \partial s_2}$ at $s_1 = 0$ and $s_1 = s_{1,\infty}$ gives rise to the FD formulas

$$\left.\frac{\partial^2 z}{\partial s_1 \partial s_2}\right|_{s_1=0} \equiv \left.\frac{\partial^2 z}{\partial s_1 \partial s_2}\right|_{0,j,k}^{m} \approx \frac{z^m_{1,j+1,k} + z^m_{0,j-1,k} - z^m_{0,j+1,k} - z^m_{1,j-1,k}}{2\Delta s_1 \Delta s_2}, \tag{6.13a}$$

$$\left.\frac{\partial^2 z}{\partial s_1 \partial s_2}\right|_{s_1=s_{1,\infty}} \equiv \left.\frac{\partial^2 z}{\partial s_1 \partial s_2}\right|_{\bar{n}+1,j,k}^{m} \approx \frac{z^m_{\bar{n}+1,j+1,k} + z^m_{\bar{n},j-1,k} - z^m_{\bar{n},j+1,k} - z^m_{\bar{n}+1,j-1,k}}{2\Delta s_1 \Delta s_2}, \tag{6.13b}$$

respectively. All other derivatives in (6.9) can be evaluated using the central FD schemes (3.1a), (3.1b) and (3.2).

We introduce one more one-sided FD scheme for cross derivatives such as $\dfrac{\partial^2 z}{\partial s_1 \partial s_2}$ in (6.10) or (6.11) at boundary points for which

1. $s_1 = 0$ or $s_1 = s_{1,\infty}$, and

2. $s_2 = 0$ or $s_2 = s_{2,\infty}$.

To this end, we apply the one-sided FD scheme (6.12) in both the s_1- and s_2-directions. For example, at $s_1 = 0$ and $s_2 = 0$, we use

$$\left.\frac{\partial^2 z}{\partial s_1 \partial s_2}\right|_{s_1=0,s_2=0} \equiv \left.\frac{\partial^2 z}{\partial s_1 \partial s_2}\right|^m_{0,0,k} \approx \frac{z^m_{1,1,k} + z^m_{0,0,k} - z^m_{0,1,k} - z^m_{1,0,k}}{\Delta s_1 \Delta s_2}. \tag{6.14}$$

Other central FD schemes are replaced by one-sided FD schemes similar to those described above whenever using a central FD scheme in one of (6.9), (6.10) or (6.11) would require the use of a gridpoint outside Ω_b.

A derivation of (6.12), (6.13) and (6.14) is presented in Appendix C.1. As verified through Taylor expansions, each of (6.12), (6.13) and (6.14) is first-order, provided that the function z is sufficiently continuously differentiable. It is worth noting that, although (6.12), (6.13) and (6.14) are first-order approximations to the first and the cross derivatives, when the computational domain is properly truncated, the far-field effects of the boundaries on the spot price of the derivatives is insignificant. Furthermore, with z being almost linear at the boundary, the truncation error of the one-sided FD approximations to the derivatives is negligible. Thus, we still observe second-order convergence for the spot price of the derivatives in practice. Detailed numerical results are presented in Section 7.1.

The FD discretization of the spatial differential operator $\mathcal{L}_b$ of (6.3) on the spatial grid Ω_b is performed by replacing each spatial derivative appearing in the operator $\mathcal{L}_b$ by its corresponding FD scheme (as in (3.1), (3.2) and (6.12)–(6.14)).

Remark 6.2.2. A possible alternative approach to using the one-sided forward and backward FD approximations (6.12) is to introduce layers of "ghost" gridpoints outside the

computational domain Ω_b, and use the standard central FD schemes described above to discretize the PDE (6.3) and the boundary conditions (6.8) at gridpoints on the boundaries. Then, the "ghost" gridpoints can be eliminated using the equations arising from (6.3) and (6.8), assuming that these equations are linearly independent.

Time Discretization: ADI schemes

For use in this subsection, we denote by $\mathbf{I}$ the $(\bar{n}+2)(\bar{p}+2)(\bar{q}+2)\times(\bar{n}+2)(\bar{p}+2)(\bar{q}+2)$ identity matrix, and by $\mathbf{B}$ the matrix of the same size as $\mathbf{I}$ arising from the FD discretization of the differential operator $\mathcal{L}_b$ on the spatial grid Ω_b using FD schemes (3.1), (3.2) and (6.12)–(6.14), as described in the previous section. Note that, since the coefficients of the differential operator $\mathcal{L}_b$ are constant, the discretization matrix $\mathbf{B}$ is not time-dependent. Let $\mathbf{z}^m$ denote the vector of values at time τ_m on the mesh Ω_b that approximates the exact solution $z^m = z(\mathbf{s}, \tau_m)$ of a multi-asset European option written on three assets. Furthermore, denote by $\mathbf{z}^*$ the vector of the payoff values on Ω_b. Note that each of the vectors $\mathbf{z}^m$ and $\mathbf{z}^*$ is of size $(\bar{n}+2)(\bar{p}+2)(\bar{q}+2)$. Similar to the ADI splitting techniques presented earlier in Subsection 3.2.2, we decompose the matrix $\mathbf{B}$ into four submatrices:

$$\mathbf{B} = \mathbf{B}_0 + \mathbf{B}_1 + \mathbf{B}_2 + \mathbf{B}_3.$$

The matrices $\mathbf{B}_1$, $\mathbf{B}_2$ and $\mathbf{B}_3$ are the parts of $\mathbf{B}$ that correspond to the spatial derivatives in the s_1-, s_2- and s_3-directions, respectively, while the matrix $\mathbf{B}_0$ is the part of $\mathbf{B}$ that comes from the FD discretization of the cross derivative terms in the operator $\mathcal{L}_b$. The term rz in $\mathcal{L}_b$ is distributed evenly over $\mathbf{B}_1$, $\mathbf{B}_2$ and $\mathbf{B}_3$. Starting from an approximation $\mathbf{z}^{m-1}$ to the exact solution z^{m-1}, the Hundsdorfer and Verwer scheme [34] generates an

approximation $\mathbf{z}^m$ to the exact solution z^m, $m = 1, 2, \ldots, \bar{l}$, by

$$\begin{cases} \mathbf{v}_0 = \mathbf{v}^{m-1} + \Delta\tau_m \mathbf{B}\mathbf{z}^{m-1}, & (6.15a) \\ (\mathbf{I} - \theta\Delta\tau_m \mathbf{B}_i)\mathbf{v}_i = \mathbf{v}_{i-1} - \theta\Delta\tau_m \mathbf{B}_i \mathbf{z}^{m-1}, \quad i = 1, 2, 3, & (6.15b) \\ \widetilde{\mathbf{v}}_0 = \mathbf{v}_0 + \frac{1}{2}\Delta\tau_m(\mathbf{B}\mathbf{v}_3 - \mathbf{B}\mathbf{z}^{m-1}), & (6.15c) \\ (\mathbf{I} - \theta\Delta\tau_m \mathbf{B}_i)\widetilde{\mathbf{v}}_i = \widetilde{\mathbf{v}}_{i-1} - \theta\Delta\tau_m \mathbf{B}_i \mathbf{v}_3, \quad i = 1, 2, 3, & (6.15d) \\ \mathbf{z}^m = \widetilde{\mathbf{v}}_3. & (6.15e) \end{cases}$$

Note that all the matrices $\mathbf{B}_i$, $i = 1, 2, 3$, are block-diagonal with tridiagonal blocks.

Timestep Size Selector

We use a simple, but effective, timestep size selector presented in [18] that was shown to work well in the context of pricing options (e.g. see [9] and [18]). The idea underlying this scheme is to predict a suitable timestep size for the next timestep, using only information from the current and previous timesteps. We extend this timestep size selector for use with multi-asset options, including American options presented in the next section.

According to the formula in [18], given the current stepsize $\Delta\tau_m$, $m \geq 1$, the new stepsize $\Delta\tau_{m+1}$ is given by

$$\begin{cases} \Delta\tau_{m+1} &= \left(\min_{1 \leq \iota \leq npq} \left[\frac{\texttt{dnorm}}{\frac{|\mathbf{z}_\iota^m - \mathbf{z}_\iota^{m-1}|}{\max(N, |\mathbf{z}_\iota^m|, |\mathbf{z}_\iota^{m-1}|)}} \right] \right) \Delta\tau_m, \\ \Delta\tau_{m+1} &= \min\left\{ \Delta\tau_{m+1}, \bar{T} - \tau_m \right\}. \end{cases} \tag{6.16}$$

Here, `dnorm` is a user-defined target relative change, and the scalar N is chosen so that the method does not take an excessively large stepsize where the value of the option is small. Normally, for option values in dollars, $N = 1$ is used. In the experiments, we use the parameters $\Delta\tau_1 = 10^{-3}$ (the initial timestep size) and `dnorm` $= 0.4$ on the coarsest grids. The value of `dnorm` is reduced by a factor of two at each grid refinement, while $\Delta\tau_1$ is reduced by a factor of four.

A pricing algorithm for multi-asset European options is described in Algorithm 6.2.1.

Algorithm 6.2.1 Algorithm for pricing multi-asset European options.

1: initialize $\Delta\tau_1$;
2: set $\mathbf{z}^0 = \mathbf{z}^*$;
3: **for** $m = 1, \ldots, \bar{l}$ **do**
4: compute $\mathbf{z}^m$ using ADI scheme (6.15);
5: **if** $m < \bar{l}$ **then**
6: compute $\Delta\tau_{m+1}$ using (6.16);
7: **end if**
8: **end for**

6.3 Multi-asset American Options

6.3.1 Introduction

In contrast to European options, American options can be exercised at any time up to and including the maturity of the option. More specifically, a multi-asset American call/put option contract gives the holder the right, but not an obligation, to buy/sell a specified basket of more than one underlying asset for a fixed exercise price E at any time up to and including the maturity time $\bar{T}$. An American option, thus, offers more opportunity for the option holder to make a profit than its European counterpart.

The problem of pricing multi-asset American options is not only mathematically challenging but also very computationally intensive. The computational challenges come from the high-dimensionality of the problem, while the mathematical challenges arise from the early exercise feature of the option, which leads to an additional constraint that the value of an American option at any time $\tau \leq \bar{T}$ must be greater than or equal to

its payoff [66]. This constraint requires special treatment, a fact that makes an explicit closed form solution for an American option intractable for most cases. Consequently, numerical methods must be used.

This section discusses efficient PDE-based numerical methods for pricing multi-asset American options in the Black-Scholes-Merton framework. Using a PDE approach, the American option pricing problem can be formulated as a time-dependent LCP with the inequalities involving the Black-Scholes-Merton PDE and some additional constraints [69]. We adopt the penalty method of [18] to solve the LCP. In this approach, a penalty term is added to the discretized equations to enforce the early exercise constraint. The solution of the resulting discrete nonlinear equations at each timestep can be computed via a penalty iteration. An advantage of the penalty method of [18] is that it is readily extendible to handle multi-factor problems. In a high-dimensional application, such as American options written on three assets, the solution of the linear system arising at each penalty iteration presents a computational challenge. (See a relevant discussion on page 38 towards the beginning of Subsection 3.2.2 regarding the solution of the linear system arising from pricing PRDC swaps with CN timestepping method.) To handle this computational requirement, we develop an ADI-AF algorithm, based on an extension of the ADI splitting technique presented earlier, to efficiently solve these linear algebraic systems. Although variations of American options give rise to various payoff functions, we restrict our attention to American put options on the arithmetic average of three assets.

The remainder of this section is organized as follows. In Subsection 6.3.2, we present a PDE formulation of the pricing problem for a multi-asset American put option. The discretization methods are discussed in Subsection 6.3.3. A penalty iteration for the discretized American option and associated ADI-AF schemes are discussed in Subsection 6.3.4. Finally, in Subsection 6.3.5, we present a benchmark case which allows us to

obtain an accurate reference solution when pricing certain multi-asset American options.

6.3.2 Formulation

The early exercise constraint in an American option leads to the following time-dependent LCP for the value $v(\mathbf{s},\tau)$ of an American put option [66, 69]

$$\left\{ \begin{array}{r} \dfrac{\partial v}{\partial \tau} - \mathcal{L}_b v = 0 \\ v - v^* \geq 0 \end{array} \right\} \text{ or } \left\{ \begin{array}{r} \dfrac{\partial v}{\partial \tau} - \mathcal{L}_b v > 0 \\ v - v^* = 0 \end{array} \right\}, \tag{6.17}$$

$$\mathbf{s} \in (\Omega_b \setminus \partial\Omega_b), \tau \in (0, \bar{T}],$$

subject to the initial (payoff) condition

$$v^*(\mathbf{s}) \equiv v(\mathbf{s},0) = \max\Big(E - \sum_{i=1}^{3} w_i s_i, 0\Big) \text{ on } \Omega_b \times \{0\}, \tag{6.18}$$

and the boundary conditions [42]

$$v(\mathbf{s},\tau) = v^*(\mathbf{s}) \text{ on } \partial\Omega_b \times (0,\bar{T}]. \tag{6.19}$$

The optimal free boundary surface at each time τ, $0 \leq \tau < \bar{T}$, can then be determined a posteriori by finding where $v - v^*$ changes from being positive to being zero (within a certain tolerance). See Remark 6.3.5.

Following [18], we use a penalty parameter ζ', $\zeta' \to \infty$, and consider the non-linear PDE for the penalty formulation of the price $v(\mathbf{s},\tau)$ of an American put option written on three underlying assets

$$\frac{\partial v}{\partial \tau} - \mathcal{L}_b v = \zeta' \max(v^* - v, 0), \ \mathbf{s} \in (\Omega_b \setminus \partial\Omega_b), \tau \in (0,\bar{T}], \tag{6.20}$$

subject to the initial and boundary conditions (6.18) and (6.19), respectively. The penalty parameter ζ' effectively ensures that the solution satisfies $v - v^* \geq -\epsilon$ for some ϵ satisfying $0 < \epsilon \ll 1$. Essentially, in the region where $v \geq v^*$, the PDE (6.20) resembles the 3-D

Black-Scholes-Merton equation. On the other hand, when $-\epsilon \leq v - v^* < 0$, the 3-D Black-Scholes-Merton inequality $\dfrac{\partial v}{\partial \tau} - \mathcal{L}_b v > 0$ is satisfied and $v \approx v^*$.

Note that there are other approaches for choosing proper boundary conditions, such as those presented in [55]. These approaches typically involve solving the Black-Scholes-Merton PDEs with a smaller number of spatial dimensions that the option values approximately satisfy on the boundary of the computational domain. More specifically, in these approaches, for d underlying assets, a series of k-dimensional Black-Scholes-Merton PDEs, where $k = \{d-1, d-2, \ldots, 1\}$, are solved in order to determine approximate values of the option on the boundaries. For example, in our case with $d = 3$, a total of (i) six 2-D Black-Scholes-Merton PDEs defined on the six open faces of $\partial\Omega_b$, and (ii) twelve 1-D Black-Scholes-Merton PDEs defined on the twelve open edges of $\partial\Omega_b$ must be solved. However, by approximating the values of the option on the boundary by the payoff function, our choice for the approximate boundary conditions is much simpler, and has been shown to work well for the pricing problem for multi-asset American options. See the discussion and numerical results presented in Section 7.2.

6.3.3 Discretization

Space Discretization

Let the gridpoint values of a FD approximation to the solution v be denoted by

$$v^m_{i,j,k} \approx \quad v(s_{1i}, s_{2j}, s_{3k}, \tau_m) = \quad v(i\Delta s_1, j\Delta s_2, k\Delta s_3, \tau_m),$$

where $i = 0, \ldots, \bar{n}+1$, $j = 0, \ldots, \bar{p}+1$, $k = 0, \ldots, \bar{q}+1$, $m = 0, \ldots, \bar{l}$. Since the boundary condition (6.19) is of the Dirichlet-type, similar to the case of PRDC swaps, the FD discretization of the differential operator $\mathcal{L}_b$ in the interior of the rectangular domain Ω_b is performed by replacing each spatial derivative appearing in the operator $\mathcal{L}_b$ with its corresponding central FD scheme (as in (3.1) and (3.2)). We denote by $\mathcal{L}_b v^m_{i,j,k}$

the FD discretization of $\mathcal{L}_b$ at $(s_{1i}, s_{2j}, s_{3k}, \tau_m)$.

Time Discretization

We consider two second-order accurate time discretization schemes, namely the CN method and the two-level backward difference formula (BDF2), as well as the first-order accurate fully-implicit method, used primarily for smoothing.

Both the CN and the fully-implicit methods belong to the standard θ-timestepping discretization scheme, in which the time derivative is approximated by a first-order backward difference, while the discretized differential operator is treated as a θ-weighted average between the fully-implicit and the fully-explicit steps. More specifically, when proceeding from time τ_{m-1} to time τ_m, applying the standard θ-timestepping discretization scheme to (6.20) gives

$$(\mathcal{I} - \theta\Delta\tau_m\mathcal{L}_b)v^m_{i,j,k} = (\mathcal{I} + (1-\theta)\Delta\tau_m\mathcal{L}_b)v^{m-1}_{i,j,k} + \mathcal{P}v^m_{i,j,k}, \tag{6.21}$$

where $0 \leq \theta \leq 1$. Here, $\mathcal{I}$ and $\mathcal{P}$ denote the identity and penalty operators, respectively, where $\mathcal{P}$ is defined by

$$\mathcal{P}v^m_{i,j,k} = \zeta \max(v^*_{i,j,k} - v^m_{i,j,k}, 0),$$

with ζ being the penalty factor related to the desired tolerance. Essentially, we have the relation $\zeta' \sim \zeta/\Delta\tau_m$. The boundary conditions (6.19) are incorporated by setting $v^m_{i,j,k} = v^*_{i,j,k}$, if $i = \{0, \bar{n}+1\}$, or $j = \{0, \bar{p}+1\}$, or $k = \{0, \bar{q}+1\}$, with $v^*_{i,j,k}$ being the payoff value at the reference point $(s_{1i}, s_{2j}, s_{3k}, \cdot)$.

In (6.21), the values $\theta = 1/2$ and $\theta = 1$ give rise to the standard CN and the fully-implicit methods, respectively. It is known that, although the CN method is second-order accurate and unconditionally stable, i.e. no restriction on the timestep size is required for stability, this method is prone to producing spurious oscillations [69]. On the other hand, the fully-implicit method is first-order accurate, but is strongly stable (e.g. [59]). To maintain the accuracy of CN as well as smoothness of the solution, we use the Rannacher

smoothing technique [60], which applies the fully-implicit method for the first few (usually two) timesteps followed by the CN method on the remaining timesteps.

For the BDF2 scheme, the time derivative in (6.20) is approximated by the second-order difference formula[1]

$$\frac{\partial v}{\partial \tau} = \frac{1}{\Delta \tau_m}\Big(\frac{1+2c_m}{1+c_m}v^m - (1+c_m)v^{m-1} + \frac{c_m^2}{1+c_m}v^{m-2}\Big),$$

where c_m is the timestep size ratio defined in (6.5), while the discretized differential operator $\mathcal{L}_b$ is treated fully-implicitly. This gives rise to the scheme

$$\Big(\mathcal{I} - \frac{1+c_m}{1+2c_m}\Delta\tau_m\mathcal{L}_b\Big)v_{i,j,k}^m = \frac{(1+c_m)^2}{1+2c_m}v_{i,j,k}^{m-1} - \frac{c_m^2}{1+2c_m}v_{i,j,k}^{m-2} + \mathcal{P}v_{i,j,k}^m, \tag{6.22}$$

where the operators $\mathcal{I}$ and $\mathcal{P}$ are previously defined. The boundary conditions (6.19) are incorporated into (6.22) in the same fashion as in the θ-timestepping method.

It is important to note that in the case of the BDF2 method (6.22), the numerical solution for the first timestep, i.e. timestep $m = 1$, must be obtained using another method. The most natural choice for this is the fully-implicit method, which we use in our experiments. It is worth emphasizing that, since the BDF2 method is L-stable, a stronger property than the unconditional stability of the CN method [26], the BDF2 method has more favorable damping properties than the CN method does. Having good damping properties is particularly important in computing accurate hedging parameters, such as delta and gamma.

It is also important to emphasize that, for both the CN and BDF2 schemes, we also adopt another smoothing technique suggested in [59]. That is, the grids in the experiments are chosen so that there is a gridpoint at the strike E (the initial kink point) along each space dimension.

---

[1]For a uniform partition of the time interval with $\Delta\tau_m = \Delta\tau = \dfrac{\bar{T}}{l}$ and $c_m = 1$, we have the well-known BDF2 formula

$$\frac{\partial v}{\partial \tau} = \frac{3v^m - 4v^{m-1} + v^{m-2}}{2\Delta\tau}.$$

We adapt the penalty iteration algorithm in [18] to solve the set of discrete nonlinear penalized equations (6.21) and (6.22). In the next section, we present the penalty iteration algorithm and associated ADI-AF schemes.

6.3.4 Penalty Iteration and an Associated ADI-AF Technique

Let $\mathbf{v}^m$ denote the vector of values at time τ_m on the interior gridpoints of Ω_b that approximates the exact value $v^m = v(\mathbf{s}, \tau_m)$ of an American put option written on three assets. Furthermore, denote by $\mathbf{v}^*$ the vector of the payoff values on the interior gridpoints of Ω_b. Note that both vectors $\mathbf{v}^m$ and $\mathbf{v}^*$ are of size $\bar{n}\bar{p}\bar{q}$. Let κ, $\kappa \geq 0$, be the index of the penalty iteration. Let $\mathbf{v}^{m,(\kappa)}$ be the κth estimate of $\mathbf{v}^m$, and denote by $\Delta\mathbf{v}^{m,(\kappa)} = \mathbf{v}^{m,(\kappa+1)} - \mathbf{v}^{m,(\kappa)}$ the correction to the κth iterate of the penalty iteration at time τ_m.

At each penalty iteration, the θ-timestepping scheme (6.21) and the BDF2 scheme (6.22) must solve an $\bar{n}\bar{p}\bar{q} \times \bar{n}\bar{p}\bar{q}$ algebraic system of the form [18]

$$(\mathbf{I} + \theta\Delta\tau_m\mathbf{M} + \mathbf{P}^{m,(\kappa)})\mathbf{v}^{m,(\kappa+1)} = (\mathbf{I} - (1-\theta)\Delta\tau_m\mathbf{M})\mathbf{v}^{m-1} + \mathbf{P}^{m,(\kappa)}\mathbf{v}^* + \Delta\tau_m\mathbf{f}, \quad (6.23)$$

and

$$\Big(\mathbf{I} + \frac{1+c_m}{1+2c_m}\Delta\tau_m\mathbf{M} + \mathbf{P}^{m,(\kappa)}\Big)\mathbf{v}^{m,(\kappa+1)} = \frac{(1+c_m)^2}{1+2c_m}\mathbf{v}^{m-1} - \frac{c_m^2}{1+2c_m}\mathbf{v}^{m-2} + \mathbf{P}^{m,(\kappa)}\mathbf{v}^* + \frac{1+c_m}{1+2c_m}\Delta\tau_m\mathbf{f}, \quad (6.24)$$

respectively. Here, $\mathbf{I}$ denotes the identity matrix; $-\mathbf{M}$ is the matrix FD approximation to the differential operator $\mathcal{L}_b$; $\mathbf{P}^{m,(\kappa)}$ is the diagonal penalty matrix and $\mathbf{f}$ is a vector containing values arising from the boundary conditions. The explicit formula for $\mathbf{M}$ is given in Appendix C.2. The penalty matrix $\mathbf{P}^{m,(\kappa)}$ is defined by

$$\big(\mathbf{P}^{m,(\kappa)}\big)_{ij} \equiv \begin{cases} \zeta & \text{if} \quad \mathbf{v}_i^{m,(\kappa)} < \mathbf{v}_i^* \quad \text{and} \quad i = j, \\ 0 & \text{otherwise.} \end{cases} \quad (6.25)$$

In general, if we want to solve (6.20) with a relative precision *tol*, we should have $\zeta \simeq \frac{1}{tol}$ [9]. For future reference, we decompose the matrix $\mathbf{M}$ into four submatrices $\mathbf{M} = \mathbf{M}_0 + \mathbf{M}_1 + \mathbf{M}_2 + \mathbf{M}_3$. The matrices $\mathbf{M}_1$, $\mathbf{M}_2$ and $\mathbf{M}_3$ are the parts of $\mathbf{M}$ that correspond to the spatial derivatives in the s_1-, s_2- and s_3-directions, respectively, while the matrix $\mathbf{M}_0$ is the part of $\mathbf{M}$ that comes from the FD discretization of the cross derivative terms in the operator $\mathcal{L}_b$. The term rv in $\mathcal{L}_b$ is distributed evenly over $\mathbf{M}_1$, $\mathbf{M}_2$ and $\mathbf{M}_3$. For simplicity, let $\mathbf{D}^{m,(\kappa)} = \mathbf{I} + \mathbf{P}^{m,(\kappa)}$.

We adapt the ADI-AF approach discussed in [72] to solve (6.23) and (6.24). For brevity, we present only the derivation of the ADI-AF scheme for (6.23). It is straightforward to apply a similar technique to (6.24). We first write an ADI-AF scheme for (6.23) in the form

$$\begin{aligned}\big(\mathbf{D}^{m,(\kappa)}+\theta\Delta\tau_m\mathbf{M}_1\big)\big(\mathbf{D}^{m,(\kappa)}\big)^{-1}\big(\mathbf{D}^{m,(\kappa)}+\theta\Delta\tau_m\mathbf{M}_2\big)\big(\mathbf{D}^{m,(\kappa)}\big)^{-1}\big(\mathbf{D}^{m,(\kappa)}+\theta\Delta\tau_m\mathbf{M}_3\big)\mathbf{v}^{m,(\kappa+1)} = \\ (\mathbf{I}-(1-\theta)\Delta\tau_m\mathbf{M})\mathbf{v}^{m-1} + \mathbf{P}^{m,(\kappa)}\mathbf{v}^* + \Delta\tau_m\mathbf{f} + \big(\mathbf{D}^{m,(\kappa)}\big)^{-2}(\theta\Delta\tau_m)^3\mathbf{M}_1\mathbf{M}_2\mathbf{M}_3\mathbf{v}^{m,(\kappa)} \\ + \big(\mathbf{D}^{m,(\kappa)}\big)^{-1}(\theta\Delta\tau_m)^2\big(\mathbf{M}_1\mathbf{M}_2+\mathbf{M}_1\mathbf{M}_3+\mathbf{M}_2\mathbf{M}_3\big)\mathbf{v}^{m,(\kappa)} - \theta\Delta\tau_m\mathbf{M}_0\mathbf{v}^{m,(\kappa)}. \quad (6.26)\end{aligned}$$

We then subtract

$$\big(\mathbf{D}^{m,(\kappa)}+\theta\Delta\tau_m\mathbf{M}_1\big)\big(\mathbf{D}^{m,(\kappa)}\big)^{-1}\big(\mathbf{D}^{m,(\kappa)}+\theta\Delta\tau_m\mathbf{M}_2\big)\big(\mathbf{D}^{m,(\kappa)}\big)^{-1}\big(\mathbf{D}^{m,(\kappa)}+\theta\Delta\tau_m\mathbf{M}_3\big)\mathbf{v}^{m,(\kappa)}$$

from both sides of (6.26). The resulting ADI-AF scheme, referred to as *ADI-AF-CN*, for the correction $\Delta\mathbf{v}^{m,(\kappa)}$ is given by

$$\begin{aligned}\big(\mathbf{D}^{m,(\kappa)}+\theta\Delta\tau_m\mathbf{M}_1\big)\big(\mathbf{D}^{m,(\kappa)}\big)^{-1}\big(\mathbf{D}^{m,(\kappa)}+\theta\Delta\tau_m\mathbf{M}_2\big)\big(\mathbf{D}^{m,(\kappa)}\big)^{-1}\big(\mathbf{D}^{m,(\kappa)}+\theta\Delta\tau_m\mathbf{M}_3\big)\Delta\mathbf{v}^{m,(\kappa)} = \\ -(\mathbf{I}+\theta\Delta\tau_m\mathbf{M})\mathbf{v}^{m,(\kappa)} + (\mathbf{I}-(1-\theta)\Delta\tau_m\mathbf{M})\mathbf{v}^{m-1} + \mathbf{P}^{m,(\kappa)}\big(\mathbf{v}^* - \mathbf{v}^{m,(\kappa)}\big) + \Delta\tau_m\mathbf{f}, \quad (6.27)\end{aligned}$$

which can be rewritten equivalently as

$$\begin{cases} \big(\mathbf{D}^{m,(\kappa)} + \widetilde{\Delta\tau}_m \mathbf{M}_1\big)\big(\Delta\mathbf{v}^{m,(\kappa)}\big)^{(1)} = \mathbf{b}^{m,(\kappa)}, & \text{(6.28a)} \\ \big(\mathbf{D}^{m,(\kappa)} + \widetilde{\Delta\tau}_m \mathbf{M}_2\big)\big(\Delta\mathbf{v}^{m,(\kappa)}\big)^{(2)} = \mathbf{D}^{m,(\kappa)}\big(\Delta\mathbf{v}^{m,(\kappa)}\big)^{(1)}, & \text{(6.28b)} \\ \big(\mathbf{D}^{m,(\kappa)} + \widetilde{\Delta\tau}_m \mathbf{M}_3\big)\big(\Delta\mathbf{v}^{m,(\kappa)}\big)^{(3)} = \mathbf{D}^{m,(\kappa)}\big(\Delta\mathbf{v}^{m,(\kappa)}\big)^{(2)}, & \text{(6.28c)} \\ \Delta\mathbf{v}^{m,(\kappa)} = \big(\Delta\mathbf{v}^{m,(\kappa)}\big)^{(3)}, & \text{(6.28d)} \end{cases}$$

with $\widetilde{\Delta\tau}_m = \theta\Delta\tau_m$, and $\mathbf{b}^{m,(\kappa)}$ being the right-hand-side of (6.27).

Similarly, the ADI-AF scheme for the correction $\Delta\mathbf{v}^{m,(\kappa)}$ for (6.24), referred to as *ADI-AF-BDF2*, is given by relations (6.28a)–(6.28d), with $\widetilde{\Delta\tau}_m = \Big(\frac{1+c_m}{1+2c_m}\Big)\Delta\tau_m$ and the vector $\widetilde{\mathbf{b}}^{m,(\kappa)}$ associated with the right-hand-side of (6.28a) defined as

$$\widetilde{\mathbf{b}}^{m,(\kappa)} = -(\mathbf{I}+\widetilde{\Delta\tau}_m\mathbf{M})\mathbf{v}^{m,(\kappa)} + \frac{(1+c_m)^2}{1+2c_m}\mathbf{v}^{m-1} - \frac{c_m^2}{1+2c_m}\mathbf{v}^{m-2} + \mathbf{P}^{m,(\kappa)}\big(\mathbf{v}^* - \mathbf{v}^{m,(\kappa)}\big) + \widetilde{\Delta\tau}_m\mathbf{f}.$$

Thus, both the ADI-AF-CN and ADI-AF-BDF2 schemes require the solution of the linear systems in (6.28a)–(6.28c). The corresponding ADI-AF FD penalty algorithm based on the ADI-AF-CN or the ADI-AF-BDF2 scheme is presented in Algorithm 6.3.1. A pricing algorithm for multi-asset American put options is described in Algorithm 6.3.2.

Algorithm 6.3.2 Algorithm for pricing multi-asset American options.

```
initialize Δτ_1;
set v^0 = v*;
for m = 1, ..., l̄ do
    compute v^m using Algorithm 6.3.1;
    if m < l̄ then
        compute Δτ_{m+1} using (6.16);
    end if
end for
```

Remark 6.3.1. For both the ADI-AF-CN and ADI-AF-BDF2 schemes, we use as the initial guess $\mathbf{v}^{m,(0)}$ for the penalty iteration the linear two-level extrapolation of the

Algorithm 6.3.1 ADI-AF FD penalty iteration for American options

initialize $\mathbf{v}^{m,(0)}$;

construct $\mathbf{P}^{m,(0)}$ using (6.25);

for $\kappa = 0, \ldots,$ until convergence **do**

  carry out (6.28) to obtain $\Delta\mathbf{v}^{m,(\kappa)}$;

  set $\mathbf{v}^{m,(\kappa+1)} = \mathbf{v}^{m,(\kappa)} + \Delta\mathbf{v}^{m,(\kappa)}$;

  construct $\mathbf{P}^{m,(\kappa+1)}$ using (6.25);

  if $\left[\max\limits_{1\le i\le \bar{n}\bar{p}\bar{q}} \left\{ \dfrac{|\mathbf{v}_i^{m,(\kappa+1)} - \mathbf{v}_i^{m,(\kappa)}|}{\max(1, |\mathbf{v}_i^{m,(\kappa+1)}|)} \right\} < tol \right]$ or $\left[\mathbf{P}^{m,(\kappa)} = \mathbf{P}^{m,(\kappa+1)} \right]$ **then**

    break;

  end if

end for

$\mathbf{v}^{m+1} = \mathbf{v}^{m,(\kappa+1)}$;

numerical solution from the two previous timesteps, i.e. $\mathbf{v}^{m,(0)} = (1+c_m)\mathbf{v}^{m-1} - c_m\mathbf{v}^{m-2}$, except on the first timestep, where $\mathbf{v}^{1,(0)} = \mathbf{v}^0 \equiv \mathbf{v}^*$. In addition to conceptually fitting the ADI-AF-BDF2 scheme, this initial guess also (i) gives rise to more efficient ADI-AF-CN schemes than those using $\mathbf{v}^{m,(0)} = \mathbf{v}^{m-1}$, and (ii) enables consistent and fair efficiency comparisons between the two ADI-AF schemes. Detailed numerical results are presented in Tables 7.2.1–7.2.2, with a discussion in Section 7.2.

Remark 6.3.2. Due to the similarities between the aforementioned ADI-AF schemes and an ADI method, such as the Douglas and Rachford scheme [36], the fact that the cross derivatives are treated solely explicitly in these ADI-AF schemes might lead one to expect that second-order convergence of the numerical methods would be lost. This is a typical problem for ADI methods (e.g. see [36]). However, as our numerical results indicate, the ADI-AF schemes presented in this thesis exhibit second-order convergence. This does not contradict the aforementioned problem for ADI methods, since these ADI methods are used in a non-iterative context, whereas, in our case, the ADI-AF schemes are applied

iteratively. While the first iterate $\mathbf{v}^{m,(1)}$ could be a first-order accurate approximate solution, it seems that, with further penalty iterations, $\mathbf{v}^{m,(\kappa)}$ converges to a second-order accurate approximate solution at each timestep.

A heuristic explanation for this observation is as follows. The total error in the iterative solution at each penalty iteration can be viewed as arising from two sources: (1) the iteration error, which includes the first-order approximate factorization error arising from the AF schemes, and (2) the second-order truncation error. In the first iteration, the dominant source of error is the iteration error, but this error is reduced with further penalty iterations. If enough penalty iterations are performed, the major source of error is the truncation error, rather than the iteration error. Hence, we are able to observe the second-order accuracy of the numerical solution at each timestep. Detailed results are given in Tables 7.2.1–7.2.2, with a discussion in Section 7.2.

Remark 6.3.3. A possible extension of the ADI-AF schemes presented in this thesis is to modify them so that they maintain second-order accuracy for $\mathbf{v}^{m,(\kappa)}$ at each penalty iteration. To this end, after carrying out the steps in (6.28), a special correction to the cross-derivative terms, similar to those suggested in [13] in the context of ADI timestepping methods, could be added to the right-side vector of (6.28a), followed by solving an additional tridiagonal linear system along each spatial dimension, similar to (6.28a)–(6.28c). However, the computational cost per penalty iteration of an ADI-AF scheme based on this approach is approximately double that of the ADI-AF schemes considered in this thesis.

Following [13], we carried out experiments with correction terms of the form $\widetilde{\Delta\tau}_m \mathbf{M}_0 \Delta\mathbf{v}^{m,(\kappa)}$ with $\widetilde{\Delta\tau}_m$ set accordingly for ADI-AF-CN and ADI-AF-BDF2. Although the resulting ADI-AF methods did reduce the total number of iterations required for convergence compared to the ADI-AF schemes presented in this thesis, the reduction was less than 50%. As a result, it seems that a straightforward extension of the ADI-AF-CN/BDF2 schemes

based on this correction term is not cost effective. Possibly, a more productive correction term specifically designed for ADI-AF schemes could be developed.

Remark 6.3.4. For the FD discretization for the spatial variables described in (3.1), if the gridpoints are ordered appropriately, all the linear systems in (6.28) are block-diagonal with tridiagonal blocks. As a result, the number of floating-point operations per penalty iteration is directly proportional to $\bar{n}\bar{p}\bar{q}$, which yields a significant reduction in computational cost compared to the application of a direct method. Moreover, the block diagonal structure of these matrices gives rise to a simple, yet efficient, parallelization for the solution of the linear systems in (6.28), as discussed in Section 6.4.

Remark 6.3.5. The approximate location of the early exercise surface at each time τ_{m+1}, $m = 0, \ldots, \bar{l}$, can be extracted from the vector $\mathbf{v}^{m+1}$ by finding where $v^{m+1} - v^*$ changes from being positive to being zero (within the tolerance *tol*). It would be interesting to study the shape and properties, such as convexity, of the early exercise surface, as well as its evolution with time. We plan to investigate this issue further in the future.

Remark 6.3.6. We now determine the complexity of the ADI-AF penalty algorithm. We assume that all the matrices stored are in sparse format, and that variable timestep sizes are used. The cost for determining these timestep sizes is about $3\,\bar{n}\bar{p}\bar{q}$ flops[2] per timestep, which results in an approximate total of $3(\bar{l}-1)\bar{n}\bar{p}\bar{q}$ flops for all timesteps. Each of the penalty iterations requires

(i) about $43\,\bar{n}\bar{p}\bar{q}$ flops for the matrix-vector multiplications $-(\mathbf{I} + \theta\Delta\tau_m\mathbf{M})\mathbf{v}^{m,(\kappa)}$ and $\mathbf{P}^{m,(\kappa)}(\mathbf{v}^* - \mathbf{v}^{m,(\kappa)})$, and the addition involving the vector $\mathbf{f}$, assuming that the PDE coefficients are available (see Step (x) below);[3]

[2] A flop is one addition, or one subtraction, or one multiplication, or one division of two floating-point numbers.

[3] Since the matrix $\mathbf{M}$ has about 19 nonzero entries per row, the matrix-vector product $\mathbf{M}\mathbf{v}^{m,(\kappa)}$ requires about $38\,\bar{n}\bar{p}\bar{q}$ flops.

(ii) about $2\,\bar{n}\bar{p}\bar{q}$ flops for updating the two right-side vectors of (6.28b)–(6.28c).

(iii) about $15\,\bar{n}\bar{p}\bar{q}$ flops for updating the three tridiagonal matrices in (6.28a)–(6.28c), assuming $\mathbf{M}_1, \mathbf{M}_2$, and $\mathbf{M}_3$ are available (see Step (xi) below);

(iv) about $12\,\bar{n}\bar{p}\bar{q}$ flops for the solutions of the three tridiagonal systems in (6.28a)–(6.28c);[4]

(v) about $\bar{n}\bar{p}\bar{q}$ flops for updating the vector $\mathbf{v}^{m,(\kappa+1)}$;

(vi) about $\bar{n}\bar{p}\bar{q}$ flops for checking the stopping criterion.

In addition to the above costs, at each timestep, the ADI-AF-CN and ADI-AF-BDF2 schemes require

(vii) about $3\,\bar{n}\bar{p}\bar{q}$ flops for the initial guess;

(viii) about $40\,\bar{n}\bar{p}\bar{q}$ flops for the matrix-vector multiplication involving $\mathbf{v}^{m-1}$ (ADI-AF-CN) and about $3\,\bar{n}\bar{p}\bar{q}$ for the vector-vector addition of $\mathbf{v}^{m-1}$ and $\mathbf{v}^{m-2}$ (ADI-AF-BDF2).

Moreover, we also need to include

(ix) about $16(\bar{n}\bar{p}+\bar{n}\bar{q}+\bar{p}\bar{q})$ flops for computing values arising from the boundary conditions, assuming that the coefficients of the PDE are available (see Step (x));

(x) about $57\,\bar{n}\bar{p}\bar{q}$ flops for computing the coefficients of the PDE; and

(xi) $21\,\bar{n}\bar{p}\bar{q}$ flops for assembling the three tridiagonal matrices $\mathbf{M}_1, \mathbf{M}_2$, and $\mathbf{M}_3$, assuming that the coefficients of the PDE are available (see Step (x)).

Since the values arising from the boundary conditions are not time-dependent and are relatively cheap to store, in our implementation, we compute the values in Step (ix)

[4] About $4\,\bar{n}\bar{p}\bar{q}$ flops are needed for the solution of each of these tridiagonal systems.

only once at the first penalty iteration of the first timestep, and store them for use in subsequent penalty iterations. On the other hand, while the values computed in Steps (x)–(xi) could be stored for use at each subsequent timestep and/or penalty iteration, in our implementation, we recompute these values at each penalty iteration. This extra computation is often cheaper than the memory bandwidth to retrieve pre-computed values, not to mention a significant reduction in the memory footprint.

Let $\bar{\kappa}$ denote the total number of penalty iterations over all timesteps required by the penalty method. Thus, the approximate total number of flops required by our implementation of the ADI-AF-CN and ADI-AF-BDF2 penalty algorithms are

$$152\,\bar{n}\bar{p}\bar{q}\bar{\kappa} + 43\,\bar{n}\bar{p}\bar{q}\bar{l} + 16(\bar{n}\bar{p} + \bar{n}\bar{q} + \bar{p}\bar{q})$$

and

$$152\,\bar{n}\bar{p}\bar{q}\bar{\kappa} + 6\,\bar{n}\bar{p}\bar{q}\bar{l} + 16(\bar{n}\bar{p} + \bar{n}\bar{q} + \bar{p}\bar{q}),$$

respectively.

Remark 6.3.7. The total number of flops required for the timestep size selector (6.16) is about $3\,\bar{n}\bar{p}\bar{q}$ per timestep, which results in an approximate total of $3(\bar{l}-1)\bar{n}\bar{p}\bar{q}$ flops for all timesteps.

6.3.5 Benchmark Case: Geometric Average American Options

Although most basket options are written on arithmetic averages, using geometric averages instead allows us to compute an accurate benchmark solution using a dimension reduction approach. Below, we discuss geometric average American put options and an associated dimension reduction approach.

We first consider the geometric average process $g(t)$ defined by

$$g(t) = \Big(\prod_{i=1}^{3} s_i(t)\Big)^{\frac{1}{3}}, \quad t \geq 0. \tag{6.29}$$

It can be shown that the dynamics of $g(t)$ are given by

$$\frac{dg(t)}{g(t)} = (r - d_g)dt + \sigma_g dB_g(t), \tag{6.30}$$

where

$$\sigma_g = \Big(\frac{1}{3^2}\sum_{i,j=1}^{3} \rho_{ij}\sigma_i\sigma_j\Big)^{\frac{1}{2}}, \tag{6.31}$$

and

$$d_g = \frac{1}{3}\sum_{i=1}^{3}\big(d_i + \frac{1}{2}(\sigma_i)^2\big) - \frac{1}{2}(\sigma_g)^2. \tag{6.32}$$

Here, $B_g(t)$ is a standard Brownian motion defined by

$$B_g(t) = \frac{1}{3\sigma_g}\sum_{i=1}^{3}\sigma_i W_i(t). \tag{6.33}$$

The derivation of the dynamics of the geometric average process $g(t)$, i.e. relations (6.30)–(6.33), is presented in Appendix E.2. We then consider a geometric average American put option written on three assets with the payoff function

$$v^*(\mathbf{s}) = \max(E - g(t), 0), \tag{6.34}$$

where the dynamics of $g(t)$ are given by (6.30)–(6.33). This option is essentially equivalent to an American put option written on one asset with starting value $g(0) = \Big(\prod_{i=1}^{3} s_i(0)\Big)^{\frac{1}{3}}$, strike E, volatility σ_g defined by (6.31), and risk-neutral drift $r_g = r - d_g$, where d_g is defined by (6.32). Since an American option written on one asset can be solved very efficiently and accurately using a wide range of numerical methods, such as the adaptive and high-order pricing methods developed in [9], we can obtain a very accurate benchmark solution for a geometric average American put option written on three assets by solving the equivalent pricing problem for an American put option on one asset. A detailed discussion and numerical results are presented in Subsection 7.2.1.

6.4 GPU Implementation

The parallelization of the ADI timestepping scheme (6.15) for a multi-asset European option is essentially the same as the one discussed in Chapter 4, and hence is not repeated here. The focus of this section is a parallelization of the timestep size selector (6.16) and a GPU-based parallel algorithm for each penalty iteration of the Algorithm 6.3.1. As an example, we focus on describing the parallel implementation of the ADI-AF-CN scheme and the stopping criterion (Line 6) of the penalty algorithm. The implementation the ADI-AF-BDF2 scheme is essentially the same, and hence omitted.

6.4.1 Timestep Size Selector

The key to an efficient implementation of the timestep size selector (6.16) on a GPU is quickly finding the minimum element of a large array of real numbers. To this end, we adapt the parallel reduction technique discussed in [27, 28]. In this approach, there are two levels of reduction involved in the process: the local and the global reductions. The local reduction refers to the reduction within each threadblock, the purpose of which is to find the minimum value within a threadblock. On the other hand, the global reduction aims at processing the intermediate results from all of the threadblocks. Both levels of reduction are built-upon on tree-based parallel techniques. We explain these two phases in detail below. Although, in this section, we focus on describing parallel reduction techniques in the context of finding the minimum element of a large array, it is straightforward to extent the techniques to other reduction operations, such as computing the sum of an array.

Local Reduction

Assume that we have a 1-D array of s_t elements in the global memory, referred to as the array B, and this array is assigned to a 1-D threadblock of the same size. In this

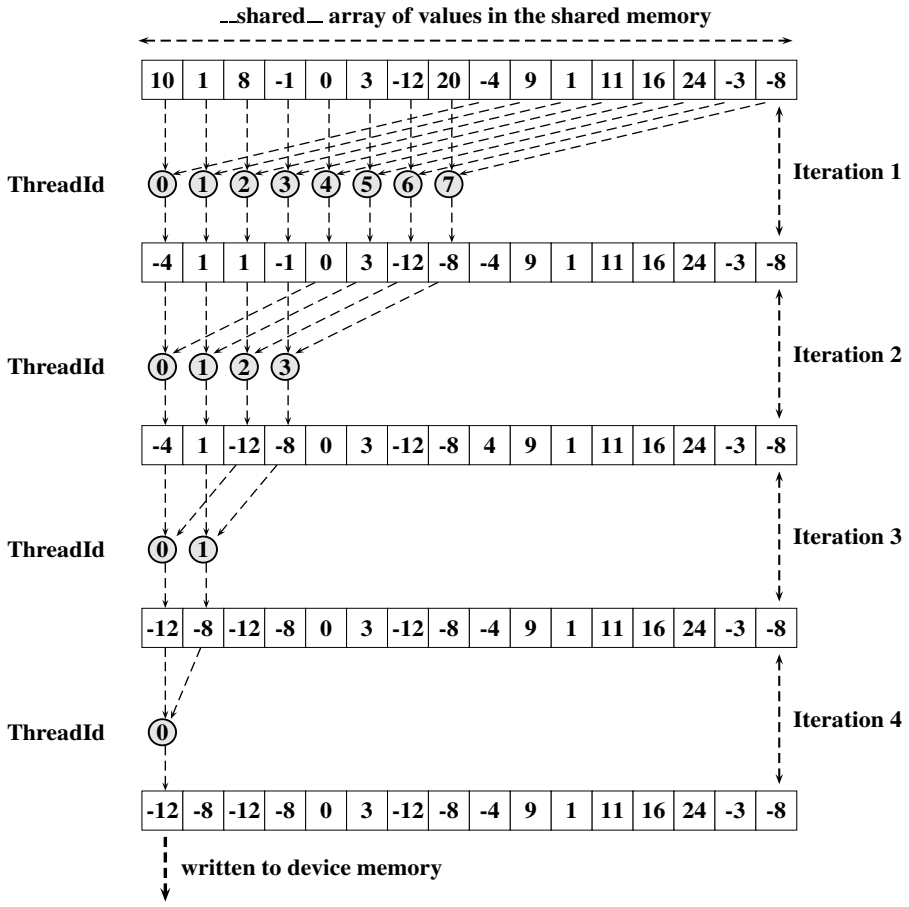


Figure 6.4.1: An example of a tree-based algorithm for finding the minimum element of a 1D array. The second half of the array are compared pairwise with the first half of the array by the set of leading threads of the threadblock.

phase of reduction, the minimum value of the array B is obtained using a tree-based parallel technique. More specifically, each of the $\frac{s_t}{2}$ threads is assigned a pair of numbers, compares them, and writes the result (the smaller of the pair) to its location in the original array. After this step, we have $\frac{s_t}{2}$ partial minimum values stored at specific locations in the original array. We repeat the procedure, via a loop in the kernel, until we obtain the minimum value of the array B. This minimum value is then written to a location specifically assigned to the threadblock in the global memory.

Several considerations need to be taken into account when designing an efficient GPU algorithm for this tree-based approach. First, the result produced by a thread during the current iteration will be accessed by another thread in the next iteration when pairs of numbers are compared. As a result, the array B should really be cached. Since the GPU which we used for our numerical experiments does not have a cache, we manually cache the entries of the array B in the shared memory by using a `__shared__` array in the kernel; this array is used in each iteration. It is also important to emphasize that for the loading phase from the global memory to the `__shared__` array, barrier synchronization among threads in the same threadblock must be enforced, and this can be achieved by using the function `__syncthreads()`. Second, not all threads in a threadblock are busy during each iteration. More specifically, $\frac{s_t}{2}$ threads are busy in the first iteration, while only $\frac{s_t}{4}$ threads are busy in the second, and so on. For efficiency, it is desirable to ensure that each warp is either fully active or fully inactive, as much as possible. Therefore, in our implementation, during each iteration, the elements of the second half of the (sub-)array being processed are compared pairwise with those of the first half of the (sub-)array by the set of leading threads of the threadblock. A pictorial presentation of the aforementioned tree-based algorithm is given in Figure 6.4.1.

Global Reduction

Since threads from different threadblocks cannot communicate effectively with each other, a mechanism is needed to process the partial results, i.e. intermediate minimum values, produced by different threadblocks. In the local reduction approach described above, after a kernel has executed, the minimum value of each threadblock is written to a different entry in a global array of partial results. The purpose of the global reduction is to use the kernel launch as a global synchronization point to process partial results; this is inexpensive because a kernel launch has negligible overhead.

More specifically, assume that initially we have a large array, called G, of size N_s, in the global memory, and we want to find the minimum value of G. The general idea is to partition the initial array G into $\texttt{ceil}(N_s/s_t)$ sub-arrays of size s_t, each of which is assigned to a 1-D threadblock of the same size. Note that each of these sub-arrays plays the role of the array B in the local reduction level described earlier. During the first kernel launch, i.e. the first level of global reduction, each threadblock first carries out the reduction operation via the tree-based approach described in the previous subsection, to find the minimum of the corresponding sub-array, then writes the intermediate result to its exclusively assigned 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, i.e. the second level of global reduction, but now the number of threadblocks is divided by s_t. 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, i.e. the last level of global reduction, after which the minimum element of the array G is found.

Figure 6.4.2 illustrates an example of this global reduction approach applied to an array of $N_s = \bar{n}\bar{p}\bar{q}$ elements. In the first level of global reduction, i.e. Level 1, $\bar{n}\bar{p}\bar{q}$ elements are partitioned into $\texttt{ceil}\left(\frac{\bar{n}\bar{p}\bar{q}}{s_t}\right)$ 1-D sub-arrays of data, each of which has size s_t, and is assigned to a 1-D threadblock of the same size. After Level 1, we have an array of size $\texttt{ceil}\left(\frac{\bar{n}\bar{p}\bar{q}}{s_t}\right)$ containing the intermediate results in the global memory to work with in the second level of global reduction. In general, in the kth level of global reduction, we have an array of size $\texttt{ceil}\left(\frac{\bar{n}\bar{p}\bar{q}}{(s_t)^{k-1}}\right)$ containing intermediate results to work with, and this array is partitioned into $\texttt{ceil}\left(\frac{\bar{n}\bar{p}\bar{q}}{(s_t)^{k}}\right)$ sub-arrays, each of which is of size s_t, and is processed by a 1-D threadblock of the same size.

Note that, although the number of threadblocks decreases after each level of global reduction, the number of threads in a threadblock remains the same at each level of reduction. As a result, the kernel code for all levels of global reduction is virtually the

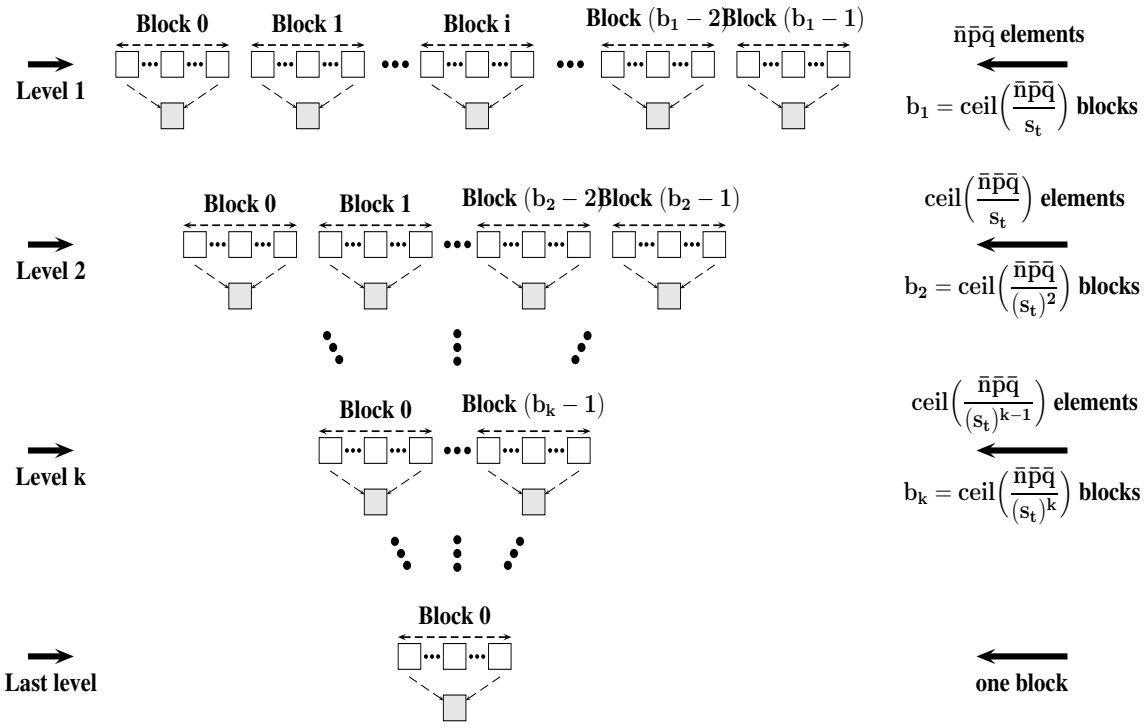


Figure 6.4.2: Global reduction levels and associated numbers of elements and threadblocks.

same, except that in the first kernel call, each thread of a threadblock needs to load, via a coalesced pattern, its components of the vectors $\mathbf{z}^m$ and $\mathbf{z}^{m-1}$ (or $\mathbf{v}^m$ and $\mathbf{v}^{m-1}$ in the case of American options) and computes the respective quantity $\dfrac{|\mathbf{z}_\iota^m - \mathbf{z}_\iota^{m-1}|}{\max(N, |\mathbf{z}_\iota^m|, |\mathbf{z}_\iota^{m-1}|)}$, before performing local reduction to find the minimum element of the threadblock.

Although several advanced optimization techniques, such as unrolling the last warp, loop unrolling, and processing multiple elements per thread, can be used to improve the performance of the GPU-based timestep size selector, we decided not to implement any of these more sophisticated techniques, since the current implementation of the timestep size selector is sufficiently fast and its computation times occupy a very small fraction, about 1%, of the total computation times (see, for example, Table 7.2.2). Therefore, there would be little benefit to reducing the runtime of the timestep size selector further.

6.4.2 ADI-AF schemes

For presentation purposes, let

$$\begin{aligned}
\mathbf{w}^{m-1} &= (1-\theta)\Delta\tau_m \mathbf{M}\mathbf{v}^{m-1}, \\
\mathbf{w}^{(\kappa)} &= \theta\Delta\tau_m \mathbf{M}\mathbf{v}^{m,(\kappa)}, \\
\widehat{\mathbf{M}}_i^{m,(\kappa)} &= \mathbf{D}^{m,(\kappa)} + \theta\Delta\tau_m \mathbf{M}_i, \quad i = 1,2,3, \\
\widehat{\Delta\mathbf{v}}^{(\kappa),i} &= \mathbf{D}^{m,(\kappa)}\big(\Delta\mathbf{v}^{m,(\kappa)}\big)^{(i-1)}, \quad i = 2,3,
\end{aligned}$$

and notice that

$$\mathbf{b}^{m,(\kappa)} = \mathbf{v}^{m-1} - \mathbf{v}^{m,(\kappa)} - (\mathbf{w}^{m-1} + \mathbf{w}^{(\kappa)}) + \mathbf{P}^{m,(\kappa)}\big(\mathbf{v}^* - \mathbf{v}^{m,(\kappa)}\big) + \Delta\tau_m \mathbf{f}.$$

Here, to simplify the notation, we have dropped the superscript, m, for the timestep index from the vectors $\mathbf{w}^{(\kappa)}$ and $\widehat{\Delta\mathbf{v}}^{(\kappa),i}$, $i = 2,3$. The computation of the ADI-AF-CN scheme (6.28) and the checking of the stopping criterion of Algorithm 6.3.1 consist of the following steps:

(i) Step a.1: Compute the matrices $\mathbf{D}^{m,(\kappa)}$, $\mathbf{M}_i$ and $\widehat{\mathbf{M}}_i^{m,(\kappa)}$, $i = 1,2,3$, and the vectors $\mathbf{w}^{m-1}$, $\mathbf{w}^{(\kappa)}$ and $\mathbf{b}^{m,(\kappa)}$;

(ii) Step a.2: Solve $\widehat{\mathbf{M}}_1^{m,(\kappa)}\big(\Delta\mathbf{v}^{m,(\kappa)}\big)^{(1)} = \mathbf{b}^{m,(\kappa)}$;

(iii) Step a.3: Compute $\widehat{\Delta\mathbf{v}}^{(\kappa),2}$ and solve $\widehat{\mathbf{M}}_2^{m,(\kappa)}\big(\Delta\mathbf{v}^{m,(\kappa)}\big)^{(2)} = \widehat{\Delta\mathbf{v}}^{(\kappa),2}$;

(iv) Step a.4: Compute $\widehat{\Delta\mathbf{v}}^{(\kappa),3}$ and solve $\widehat{\mathbf{M}}_3^{m,(\kappa)}\big(\Delta\mathbf{v}^{m,(\kappa)}\big)^{(3)} = \widehat{\Delta\mathbf{v}}^{(\kappa),3}$;

(v) Step a.5: Check the stopping criterion.

We observe similarities between the computation of the ADI-AF schemes and that of the ADI timestepping method (3.4) (and also of (6.15)). More specifically, the computation of the vector $\mathbf{w}^{m-1}$ in Step a.1 resembles the explicit Euler predictor step, while Steps a.2–a.4 are essentially the same as the three implicit corrector steps in (3.4), each

of which involves solving a block-diagonal system with tridiagonal blocks along a spatial dimension. As a result, the GPU-based parallelization of the ADI-AF scheme considered in this section can be viewed as a natural extension of the parallelization of the ADI timestepping method presented in Section 4.3 For brevity, we only outline the main steps of the parallel algorithm for the ADI-AF-CN scheme. A detailed discussion of each of the parallelization steps can be found in Section 4.3.

Step a.1

We assume that, initially, the vectors $\mathbf{v}^{m-1}$, $\mathbf{v}^{m,(\kappa)}$ and $\mathbf{v}^*$ are in the global memory, and any needed constants (model parameters) are in the constant cache. Note that the data copying from the host memory to the device memory occurs on the first timestep only, for the initial condition (payoff) data and the model constants. Data for the subsequent timesteps and steps of the ADI-AF schemes are stored in the global memory.

We apply the same partitioning approach as discussed in Subsection 4.3.1. That is, we partition the $\bar{n} \times \bar{p} \times \bar{q}$ computational grid into 3-D blocks of size $n_b \times p_b \times q$, i.e. a total of $\bar{q}$ $n_b \times p_b$ tiles. All gridpoints of a $n_b \times p_b \times \bar{q}$ 3-D block are assigned to one $n_b \times p_b$ threadblock only, with one thread for each "stack" of q gridpoints in the s_3 direction. This partitioning approach and data assignment result in a $\bar{q}$-iteration loop in the kernel (also see Figure 4.3.1).

The computation required in Step a.1 includes construction of matrices, addition of matrices, multiplication of vectors and matrices by scalars, as well as matrix-vector multiplication. With the approach of data partitioning and assignment to threads/threadblocks described above, all the above computations, except possibly the matrix-vector multiplication, can be executed in parallel in a natural way, without the need for communication between threads in different threadblocks. However, due to the FD schemes (3.1) and (3.2), the computation of matrix-vector multiplications embedded in $\mathbf{w}^{m-1}$ and $\mathbf{w}^{(\kappa)}$ re-

quires communication between threads in different threadblocks. More specifically, at each instance of the q-iteration loop, a threadblock carrying the computation of a tile needs the halo values of neighbouring gridpoints from adjacent tiles in the s_1 and s_2 directions. These adjacent tiles belong to different threadblocks. In our approach, this type of communication is realized via copies to/from the global memory, which, although they are slow, they involve small amounts of data transfer compared to the amount of computation. Furthermore, because 16KB of shared memory available per multiprocessor are not sufficient to store many data tiles, we adopt a three-plane strategy, in which, to process a tile of a 3-D block, each threadblock works with three data tiles of size $n_b \times p_b$ and their halo values during each iteration of the loop. As we proceed in the s_3-direction, at each iteration, the next tile data are loaded, the current tile data are being computed and the previous tile data are then being discarded. For issues regarding memory coalescing at this step, the reader is referred to the discussion on page 78 in Subsection 4.3.1.

Remark 6.4.1. It is worth emphasizing that the vector $\mathbf{w}^{m-1}$ is computed only in the first penalty iteration of the mth timestep. This vector is then loaded in a coalesced fashion from the global to the shared memory for use in subsequent penalty iterations of that timestep.

Steps a.2, a.3, a.4

Similar to the parallelization approach presented in Subsection 4.3.2 for the computation of the three implicit corrector steps of the ADI timestepping methods, 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 $\widehat{\mathbf{M}}_i^{m,(\kappa)}$, $i = 1, 2, 3$, respectively. For example, $\widehat{\mathbf{M}}_1^{m,(\kappa)}$ has $\bar{p}\bar{q}$ diagonal blocks, each block being $\bar{n} \times \bar{n}$ tridiagonal, thus the solution of $\widehat{\mathbf{M}}_1^{m,(\kappa)}\left(\Delta \mathbf{v}^{m,(\kappa)}\right)^{(1)} = \mathbf{b}^{m,(\kappa)}$ (Step a.2) is computed by first partitioning $\widehat{\mathbf{M}}_1^{m,(\kappa)}$ and

$\mathbf{b}^{m,(\kappa)}$ into $\bar{p}\bar{q}$ independent $\bar{n} \times \bar{n}$ tridiagonal systems, and then assigning each tridiagonal system to one of the $\bar{p}\bar{q}$ threads generated, i.e. each thread is assigned $\bar{n}$ gridpoints along the s_1-direction.

In our implementation, each of the 2-D threadblocks used in Steps a.2, a.3 and a.4 has the 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 issues regarding memory coalescing at this step, the reader is referred to the discussion on page 81 in Subsection 4.3.2.

Step a.5

In the current implementation, checking the stopping criterion is done during the kernel generated in Step a.4. More specifically, each threadblock of the kernel launched in Step a.4, after computing its component of the vector $\widehat{\Delta \mathbf{v}}^{(\kappa),3}$ corresponding to the reference point $(s_{1i}, s_{2j}, s_{3k}, \cdot)$, computes the quantity

$$\frac{|v_{i,j,k}^{m,(\kappa+1)} - v_{i,j,k}^{m,(\kappa)}|}{\max(1, |v_{i,j,k}^{m,(\kappa+1)}|)} \tag{6.35}$$

and the corresponding row of the penalty matrix $\mathbf{P}^{m,(\kappa+1)}$ (one entry). (Note that the loading of components of the vectors $\mathbf{v}^{m,(\kappa)}$ and $\mathbf{v}^*$ used for computing $\mathbf{P}^{m,(\kappa+1)}$ is fully coalesced.) If the quantity (6.35) is greater than or equal to the tolerance *tol* or if two corresponding rows of the matrices $\mathbf{P}^{m,(\kappa)}$ (obtained from the matrix $\mathbf{D}^{m,(\kappa)}$) and $\mathbf{P}^{m,(\kappa+1)}$ are different, the thread then respectively changes the pre-set values of two different flag variables stored in a global memory location. Note that the two pre-set flag variables are copied from the host memory to the device memory before the kernel of Step a.4 is launched. After the kernel has ended, the values of the two flag variables are copied back to the host memory to be checked. (These host-device copies are cheap.) The stopping criterion is satisfied if the two pre-set values were not altered during the kernel. Although it may happen that multiple threads try to write to the same memory location of a flag

variable at the same time, it is guaranteed that one of the writes will occur. Although we do not know which one, this does not matter for the purpose of checking the stopping criterion. Consequently, this approach suffices and works well.

Chapter 7

Numerical Results for Multi-asset Options

In this chapter, we present selected numerical results to demonstrate the effectiveness of the GPU-based parallel methods applied to the pricing of multi-asset options. In addition to the statistics mentioned in Subsection 5.1.1, additional statistics collected in this chapter include:

- "value": the spot value of the option;
- "iter. #": the total number of penalty iterations over all timesteps required by the penalty method when pricing American options;
- "work": the approximate total flops required by a ADI-AF method, and is computed as described in Remark 6.3.6 (page 139) and Remark 6.3.7 (page 141).

Note that, if the timestep size selector (6.16) is used, the "CPU time" and "GPU time" also include the total computation times for all timesteps required by the procedure. When reported, the total CPU and GPU times, expressed in milliseconds (ms.), required by the timestep size selector (6.16) for all timesteps, respectively denoted by "CPU (6.16)" and "GPU (6.16)".

We truncate the unbounded domain into a finite-sized computational one

$$\{(s_1, s_2, s_3, \tau) \in [0, s_{1,\infty}] \times [0, s_{2,\infty}] \times [0, s_{3,\infty}] \times [0, \bar{T}]\} \equiv \Omega_b \times [0, \bar{T}],$$

where $s_{1,\infty} = s_{2,\infty} = s_{3,\infty} = 3E$. The spot prices are chosen to be $s_1(0) = s_2(0) = s_3(0) = E$. For basket options, we consider the weights of the assets to be $w_1 = w_2 = w_3 = \frac{1}{3}$, so that we have $\sum_{i=1}^{3} w_i s_i(0) = \frac{1}{3} \sum_{i=1}^{3} s_i(0) = E$. Note that, with this choice of the truncated computational domain and for all grid sizes considered, there is gridpoint at E (the initial kink point) in each asset price grid.

All the GPU experiments for multi-asset option pricing were conducted on a NVIDIA Tesla T10 GPU of a server node. (See the description of the GPU cluster in Subsection 4.1.2.) Similar to pricing PRDC swaps, for both the GPU-based ADI timestepping methods (for European options) and the GPU-based ADI-AF-CN/BDF2 methods (for American options), 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 $r_t \times c_t \equiv 32 \times 4$. The size for each threadblock used in the parallelization of the timestep size selector (6.16) is $s_t = 128$.

7.1 Multi-asset European Options

We consider the ADI methods with variable timestep sizes automatically chosen by the timestep size selector (6.16) (variable-timestep-size ADI). We use the set of parameters for three assets taken from [44]: $E = 100, r = 0.04, \bar{T} = 1$, $d_1 = d_2 = d_3 = 0$, $\sigma_1 = 0.3, \sigma_2 = 0.35, \sigma_3 = 0.4$, $\rho_{12} = \rho_{13} = \rho_{23} = 0.5$. The spot prices are chosen to be $s_1(0) = s_2(0) = s_3(0) = E$. The analytic solution for the European call-on-minimum rainbow option with this set of parameter is computed to be 4.4450, using formulas in [40]. The reference solution for the basket option is 13.2449 obtained using an accurate numerical FFT-based pricing technique [44]. We refer to these two values as accurate

reference solutions for the rainbow and basket options, respectively.

Tables 7.1.1 presents selected numerical results for multi-asset European options written on three assets obtained with variable timestep size ADI methods using double-precision. The total number of timesteps, $\bar{l}$, for the variable-timestep-size methods is automatically determined by the timestep size selector (6.16). The computed prices for the rainbow and basket options on the CPU and GPU are identical and exhibit second-order convergence for the HV scheme, as expected. Regarding the timing results, the GPU is significantly faster than the CPU for any size of the discretized problem and has a speedup ratio of about 18 for the largest grid we considered.

$\bar{n}$ (s_1)	$\bar{p}$ (s_2)	$\bar{q}$ (s_3)	$\bar{l}$ (τ)	value	error	$\log_2$ ratio	CPU time (s.)	GPU time (s.)	speed up
				Rainbow option					
45	45	45	22	4.4162	2.9e-02		1.2	0.3	5.9
90	90	90	44	4.4404	4.6e-03	2.6	17.8	1.4	12.8
180	180	180	88	4.4445	5.4e-04	2.5	311.9	17.8	17.6
				Basket option					
45	45	45	24	13.2375	7.4e-03		1.3	0.3	5.1
90	90	90	45	13.2438	1.1e-03	2.5	18.1	1.5	12.8
180	180	180	87	13.2446	3.0e-04	1.9	311.7	17.8	17.6

Table 7.1.1: Spot values and performance results for pricing an at-the-money European call-on-minimum rainbow option and a basket call option, each written on three assets. Variable-timestep-size ADI methods are used.

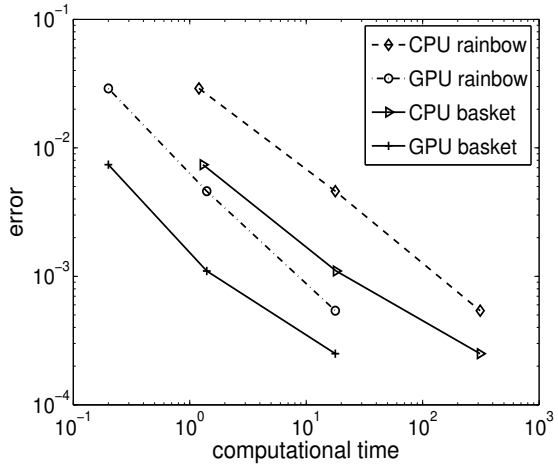


Figure 7.1.1: Efficiency comparison of the sequential CPU-based and parallel GPU-based methods with double-precision applied to the European rainbow and European basket option pricing problems.

In Figure 7.1.1, we plot errors versus computation times required by each method. It is evident that GPU-based parallel methods are significantly more efficient than standard sequential CPU-based methods in pricing these multi-asset options.

To see the effects of the single- and double-precisions on the performance on the GPU, in Table 7.1.2, we present selected numerical results of the GPU-based variable-timestep-size ADI methods applied to pricing European call-on-minimum rainbow option written on three assets. It is obvious that the numerical results obtained in the two cases are almost the same, except that, for the largest grid size we considered, the option price obtained with single-precision is less accurate than with double-precision. However, single-precision gives better timing results.

It is evident from Tables 7.1.1 and Figure 7.1.1 that the numerical methods are more efficient when applied to pricing a basket option than when applied to a rainbow option. For the same grid sizes, the errors for the basket option are about 2 to 4 times smaller than the errors for the rainbow option. To investigate this further, we look at the payoff

			Double-precision				Single-precision			
$\bar{n}$	$\bar{p}$	$\bar{q}$	$\bar{l}$	value	error	GPU	$\bar{l}$	value	error	GPU
(s_1)	(s_2)	(s_3)	(τ)			time (s.)	(τ)			time (s.)
45	45	45	22	4.4162	2.9e-02	0.2	22	4.4162	2.9e-02	0.2
90	90	90	44	4.4404	4.6e-03	1.4	44	4.4404	4.6e-03	1.0
180	180	180	88	4.4445	5.4e-04	17.8	88	4.4442	7.4e-04	13.5

Table 7.1.2: Spot values and timing results for pricing an at-the-money European call-on-minimum rainbow option written on three assets. Variable-timestep-size ADI methods are used.

functions of the two options. To simplify the analysis, we first consider a two-stock counterpart of (6.6) and (6.7), i.e. the European call-on-minimum rainbow option and the European basket option written on two assets with the payoff functions specified by

$$u(s_1, s_2, T) = \max(\min(s_1, s_2) - E, 0) \quad \text{and} \quad u(s_1, s_2, T) = \max\Big(\sum_{i=1}^{2} w_i s_i - E, 0\Big),$$

respectively. For the basket option on two stocks, the "kink region" for the payoff, i.e. the region where the first derivative of the payoff with respect to the space variables is not continuous, is just the line segment that connects the two points $(\frac{E}{w_1}, 0)$ and $(0, \frac{E}{w_2})$ in the s_1–s_2 plane. However, for the call-on-minimum rainbow option on two stocks, the kink region consists of two half lines, starting from the point (E, E) and running parallel to the stock value axes. Plots of the kink regions of the two payoff functions are given in Figure 7.1.2.

Similarly, for three assets, the kink region for the payoff of the basket option is a plane segment determined by $(\frac{E}{w_1}, 0, 0)$, $(0, \frac{E}{w_2}, 0)$, and $(0, 0, \frac{E}{w_3})$. However, the kink region for the payoff of the call-on-minimum rainbow option consists of three half planes, starting

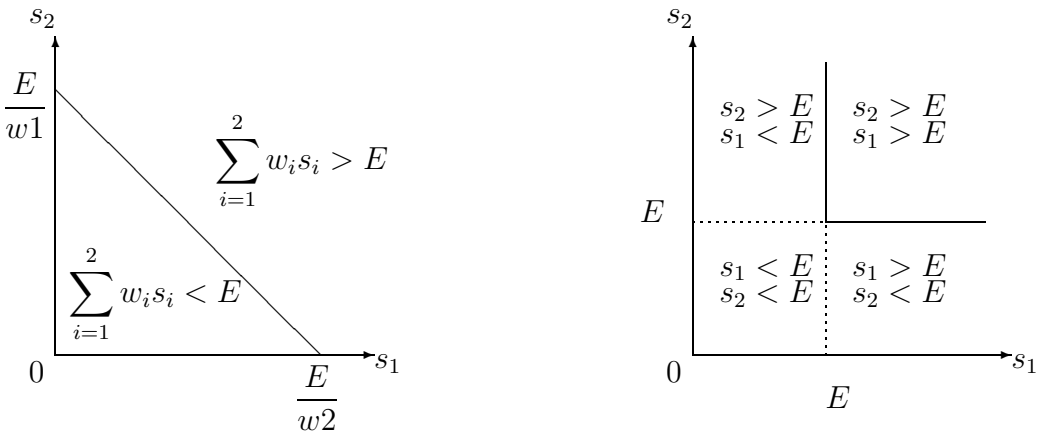


Figure 7.1.2: The kink region for the payoff function of a basket option (left) and of a call-on-minimum rainbow option (right).

from the point with coordinates $(s_1, s_2, s_3) = (E, E, E)$ and running parallel to the three faces of the rectangular spatial domain. Thus, in a sense, the topology of the payoff function of a European rainbow option is more complex and "harder" to handle than that of a European basket option. This may explain the observed loss of efficiency of the numerical methods applied to pricing a rainbow option.

7.2 Multi-asset American Options

Several methods are considered, namely, the ADI-AF-CN and ADI-AF-BDF2 methods with uniform timestep sizes (uniform-timestep-size ADI-AF-CN and uniform-timestep-size ADI-AF-BDF2, respectively), and the ADI-AF-CN and ADI-AF-BDF2 methods with variable timestep sizes automatically chosen by (6.16) (variable-timestep-size ADI-AF-CN and variable-timestep-size ADI-AF-BDF2, respectively).

We use the set of parameters for three assets taken from [42]: $E = 100, r = 0.03, \bar{T} = 0.25$, $\sigma_1 = \sigma_2 = \sigma_3 = 0.2$, $d_1 = d_2 = d_3 = 0$, $\rho_{12} = \rho_{13} = \rho_{23} = 0.5$. The penalty factor $\zeta = 10^7$ is used. Note that all computations for multi-asset American options are carried out in double precision.

$\bar{n}$ $\bar{p}$ $\bar{q}$	$\bar{l}$	value	error	$\log_2$	iter	work	$\bar{l}$	value	error	$\log_2$	iter.	work
(s_1) (s_2) (s_3)	(τ)			ratio	#	(flops)	(τ)			ratio	#	(flops)
	uniform-timestep-size ADI-AF-CN						variable-timestep-size ADI-AF-CN					
45 45 45	20	2.9569	4.8e-2		53	8.1×10^{8}	10	2.9619	4.3e-2		54	7.8×10^{8}
90 90 90	40	2.9931	1.1e-2	2.0	125	1.5×10^{10}	18	2.9948	1.0e-2	2.1	112	1.3×10^{10}
180 180 180	80	3.0015	2.9e-3	2.0	330	3.1×10^{11}	34	3.0022	2.3e-3	2.1	280	2.6×10^{11}
	uniform-timestep-size ADI-AF-BDF2						variable-timestep-size ADI-AF-BDF2					
45 45 45	20	2.9571	4.7e-2		63	8.8×10^{8}	10	2.9748	2.9e-2		53	7.4×10^{8}
90 90 90	40	2.9931	1.1e-2	2.0	134	1.5×10^{10}	18	2.9990	5.5e-3	2.4	118	1.3×10^{10}
180 180 180	80	3.0016	2.8e-3	2.0	304	2.7×10^{11}	34	3.0034	1.1e-3	2.3	292	2.6×10^{11}

Table 7.2.1: Observed errors for an at-the-money American put option on the geometric average of three assets and respective orders of convergence for various methods. The benchmark value is 3.00448.

7.2.1 Geometric Averages

With the set of parameters used, we have $g(0) = 100$, $\sigma_g = 0.1633$, and $r_g = 0.03 - 0.0067$. The benchmark solution is 3.00448 obtained using an accurate, adaptive, high-order pricing method developed in [9] for pricing American put options written on one asset.

Table 7.2.1 presents selected numerical results for the at-the-money American put option on the geometric average of three assets described above obtained with various methods. The total number of timesteps, $\bar{l}$, for the variable-timestep-size methods is automatically determined by the timestep size selector (6.16). The computed option prices enjoy second-order convergence, in all cases, a favorable behavior due to the iterative application of the ADI-AF schemes; see also Remark 6.3.2 on page 137. We next compile an efficiency comparison between various methods for solving the American put option on the geometric average of three assets. In Figure 7.2.1, we plot errors ("error") versus the

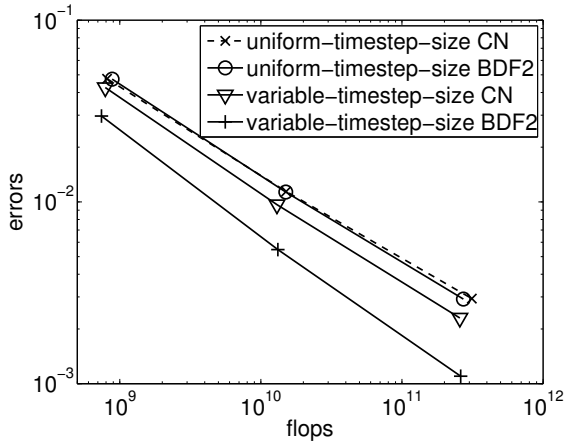


Figure 7.2.1: Efficiency comparison of various methods applied to the geometric average American put option pricing problem.

approximate total number of flops required by each of the methods ("work"). It is evident that the variable-timestep-size methods significantly outperform the uniform-timestep-size methods, with the variable-timestep-size ADI-AF-BDF2 being the most efficient, followed by the variable-timestep-size ADI-AF-CN. Between the uniform-timestep-size ADI-AF-CN and ADI-AF-BDF2 methods, the ADI-AF-BDF2 is only marginally more efficient. It seems that the benefit of the timestep size selector (6.16) is more pronounced with the ADI-AF-BDF2 scheme than with the ADI-AF-CN scheme.

Interestingly, we can also see from Table 7.2.1 that the average number of iterations per timestep required by a variable-timestep-size ADI-AF method is significantly larger than that required by its uniform-timestep-size counterpart. A possible explanation for this observation is as follows. For variable-timestep-size ADI-AF methods, although a few initial timestep sizes are usually quite small (due to small initial timestep size $\Delta\tau_1$), subsequent timestep sizes increase rapidly. However, a larger timestep size also gives rise to a larger error in the initial guess for the solution v^m of (6.21) or (6.22) as well as a larger error in an AF scheme (see (6.26)). Consequently, more penalty iterations may be required for the convergence of the penalty iteration at that timestep (also see

$\bar{n}$ (s_1)	$\bar{p}$ (s_2)	$\bar{q}$ (s_3)	$\bar{l}$ (τ)	value	change	ratio	iter. #	CPU time (s.)	GPU time (s.)	speed up	CPU (6.16) (ms.)	GPU (6.16) (ms.)	speed up
			variable-timestep-size ADI-AF-CN										
45	45	45	11	2.8924			22	0.5	0.3	1.8	6.2	1.6	3.9
90	90	90	20	2.9309	3.9e-2		54	12.7	1.0	12.8	115.7	14.3	8.0
180	180	180	37	2.9408	9.9e-3	3.9	181	289.5	17.1	16.9	1677.2	171.5	9.6
			variable-timestep-size ADI-AF-BDF2										
45	45	45	11	2.9059			26	0.7	0.3	2.3	6.1	1.6	3.9
90	90	90	20	2.9348	2.9e-2		66	15.3	1.2	11.9	115.2	14.3	8.0
180	180	180	37	2.9419	7.1e-3	4.1	217	336.3	19.8	17.1	1678.2	172.3	9.6

Table 7.2.2: Observed spot prices and performance results for an at-the-money American put option on the arithmetic average of three assets obtained using variable-timestep-size ADI-AF-CN and ADI-AF-BDF2 methods. The reference price is 2.94454 [42].

Remark 6.3.2 on page 137).

7.2.2 Arithmetic Averages

Table 7.2.2 presents selected numerical results for the American put option on the arithmetic average of three assets, described above, obtained using the two most efficient methods, namely the variable-timestep-size ADI-AF-CN and ADI-AF-BDF2 methods. In the last three columns of this table, we also present the total CPU and GPU times, expressed in milliseconds (ms.), required by the timestep size selector (6.16) for all timesteps, respectively denoted by "CPU (6.16)" and "GPU (6.16)", and the respective speedups. Note that these times are included in the total CPU and GPU times ("CPU time" and "GPU time"). Other than the computation times and the speedups, our experiments on

the CPU and on the GPU give similar results.

Table 7.2.2 shows a second-order rate of convergence for the computed "value". Moreover, these values are consistent with the reference price 2.94454 quoted in [42]. Table 7.2.2 also shows that the GPU implementation of each method is significantly faster than the corresponding CPU implementation for any size of the discretized problem. In particular, for the largest grid considered, we observe a speedup ratio of about 17 for the total computation times, while the GPU-based timestep size selector is about 10 times faster than its CPU-based counterpart.

Estimated overall rates of computation, reported in units of Giga-Floating Point Operations per second (GFLOP/s), for the two GPU-based variable-timestep-size ADI-AF methods and for the timestep size selector (6.16) are presented in Table 7.2.3.

$\bar{n}$	$\bar{p}$	$\bar{q}$	$\bar{l}$	ADI-AF-CN	ADI-AF-BDF2	(6.16)
(s_1)	(s_2)	(s_3)	(τ)	(GFLOP/s)	(GFLOP/s.)	(GFLOP/s.)
45	45	45	11	1.16	1.23	1.70
90	90	90	20	6.65	6.21	3.04
180	180	180	37	9.10	8.93	3.71

Table 7.2.3: Estimated performance results in GFLOP/s for the GPU-based variable-timestep-size ADI-AF methods and the timestep size selector (6.16).

Note that these performance results may actually be underestimated, since the flops used to compute these performance results do not include those that are duplicated on different threads during the computations. An investigation of this topic, however, is beyond the scope of this thesis; we plan to address it in the future.

Chapter 8

Summary and Future Work

8.1 Summary of Research

The thesis develops highly-efficient PDE-based modeling frameworks for multi-factor financial derivatives, with strong emphasis on three-factor models. Two important classes of financial derivatives, namely long-dated cross-currency/FX interest rate hybrids and multi-asset options, are investigated. The thesis focuses in particular on (i) PRDC swaps with popular exotic features, namely Bermudan cancelability, knockout and FX-TARN, under log-normal/FX skew models, and (ii) multi-asset American options under the Black-Scholes-Merton framework. The following summarizes the major features of the frameworks developed in this thesis and relevant research findings.

8.1.1 Cross-currency/FX Interest Rate Hybrid Derivatives

The frameworks developed for cross-currency/FX interest rate hybrids in general, and PRDC swaps in particular, can efficiently handle the early exercise features of Bermudan cancelability, as well as knockout provisions and strong path-dependency of the FX-TARN feature. In particular, when pricing these derivatives with Bermudan can-

celable and FX-TARN features via a PDE approach, substantial extra computational requirements arise.

The general pricing approach developed in this thesis for cross-currency/FX interest rate hybrids with these exotic features is based on partitioning the pricing problem into multiple independent pricing subproblems over each time period of the swap's tenor structure, each of which requires a solution of the model-dependent PDE. The modeling of PRDC swaps using one-factor Gaussian models for the domestic and foreign interest short rates, and a one-factor skew model for the spot FX rate, such as the one considered in this thesis, results in a time-dependent parabolic PDE in three space dimensions with all cross derivatives, due to the correlation between stochastic processes in the pricing model. Each of these subproblems can be solved efficiently on a GPU via an efficient parallelization of the ADI timestepping technique that is employed for the time discretization of the model PDE.

More specifically, 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, with communication at the end of the period. On a multiple-GPU platform, we can efficiently utilize two GPUs to linearly scale the speedup when pricing the underlying PRDC swap, with a speedup of about 59 for the largest grid size considered when comparing the parallel GPU to the optimized sequential CPU computing times using single-precision. In the case of FX-TARN PRDC swaps, from the pricing logic point-of-view, due to the path-dependency of the exotic feature, forward pricing algorithms, such as Monte Carlo simulations, are the natural choice for the pricing of these derivatives. By introducing an auxiliary state variable to keep track of the total accumulated PRDC coupon to date, which stays constant between dates of the tenor structure and is updated on each date of the tenor structure by a PRDC coupon amount known on that date, we develop a PDE-based pricing algorithm for FX-TARN PRDC swaps which computes

backward in time. This requires us to solve a set of independent model PDEs for each of the discretized values of the auxiliary state variable over each period of the swap's tenor structures, with communication at the end of the period. To attain even higher efficiency by solving each of the independent PDEs on a separate GPU, we extend the pricing procedure to work on a cluster of GPUs. Using a total of 40 GPUs of the cluster, we obtain a speedup of about 32 for the largest grid size considered when comparing the implementation of the pricing procedure on the cluster to a single-GPU implementation using single-precision.

An important modeling aspect of long-dated FX interest rate hybrids in general, and of PRDC swaps in particular, is the strong sensitivity of the computed price of these derivatives to the skew of the FX volatility smiles. Our numerical results show that the FX skew model, which assumes that the local volatility function depends not only on time, but also on the spot FX rate, gives rise to significantly lower values for the PRDC swap with exotic features than the plain log-normal model. These differences, which are more pronounced for the high-leverage levels than for the low- and medium-leverage levels, are seen as profit from the point-of-view of the issuer (bank). In general, PRDC swaps are very sensitive to the skew of the volatility smile, which highlights the importance of having a proper FX skew model for pricing and risk managing PRDC swaps.

8.1.2 Multi-asset Options

The focus of the thesis in this area is on European and American options written on three assets under the Black-Scholes-Merton framework. The price of a multi-asset European option satisfies the multi-dimensional Black-Scholes-Merton PDE, the solution of which is itself a computational challenge, due to the high-dimensionality of the PDE. In this case, one can apply the GPU-based parallel ADI timestepping techniques to solve this PDE ef-

ficiently. However, pricing a multi-asset American options is much more challenging, due to the mathematical complexity associated with the early exercise features, in addition to the substantial computational requirements arising from the high-dimensionality.

Using a PDE approach, the American option pricing problem can be formulated as a LCP with the inequalities involving the Black-Scholes-Merton PDE and some additional constraints. The algorithm developed incorporates the penalty approach for handling the LCP and parallel ADI-AF methods on GPUs for the solution of the linear algebraic system arising at each penalty iteration. More specifically, in this penalty approach, a penalty term is added to the discretized equations to enforce the early exercise constraint. The solution of the resulting discrete nonlinear equations at each timestep can be computed via a penalty iteration, the computation of which is intensive, due to the high-dimensionality of the problem. We develop the ADI-AF techniques, which can be viewed as being based on the splitting techniques of the ADI timestepping methods, but at the matrix level, for efficiently solving the linear system at each penalty iteration. More specifically, in our approach, the matrix associated with the linear system at each penalty iteration is approximately factored into a product of three tridiagonal matrices. Hence, the computational cost is directly proportional to the number of gridpoints. We then develop an efficient GPU-based parallelization of the ADI-AF techniques based on the parallelization of the ADI timestepping method. A GPU-based timestep-size-selector is employed to further increase the performance of the pricing methods. Numerical results indicate that the proposed algorithm is very effective for pricing such derivatives, with an observed speedup of about 18 when comparing the parallel GPU to the optimized sequential CPU computing times using double-precision.

8.2 Future Work

Several possible extensions of the research work presented in this thesis are mentioned below.

8.2.1 Numerical Methods

From a numerical methods perspective, support for non-uniform grids with more points concentrated in the regions of interest, such as around the strike of the options, or in areas where the problem is difficult, such as around the barriers of the knockout PRDC swap, could be added to the current implementation to further increase the accuracy and efficiency of the methods. To achieve even a higher efficiency, one can consider incorporating adaptive techniques, such as those developed in [9], which dynamically adjust the location of the gridpoints to control the error in the approximate solution. However, for such adaptive methods, the stability of the ADI timestepping and ADI-AF methods on a non-uniform mesh needs to be studied.

In the context of ADI-AF methods, it would be desirable to have a theoretical analysis of the second-order convergence of the ADI-AF techniques observed in the experiments. In addition, it would be interesting to investigate the damping properties of the two ADI-AF timestepping methods and their effects on the Greeks delta and gamma of the options. It would also be interesting to investigate other possible extensions of the ADI-AF schemes, such as those discussed in Remark 6.3.3, as well as to study the early exercise surface and its evolution with time as mentioned in Remark 6.3.5. Moreover, it would be desirable to develop efficient ADI-AF schemes for multi-dimensional PDEs with time-dependent coefficients. These PDEs arise very frequently in financial applications, where the local volatility functions and/or time-dependent correlations between stochastic processes in the model are present.

From a parallelization perspective, it would be desirable to investigate other possible

extensions of the GPU-based ADI timestepping methods, such as those discussed in Remarks 4.3.3, 4.3.4 and 4.3.5 in Subsection 4.3.5. Extending the current implementation of the ADI timestepping or ADI-AF schemes to a multi-GPU platform should increase the performance of the GPU algorithm presented here. It would be interesting to investigate the effects of the inter-GPU communication on the total speedup in a context of multi-GPUs. Certainly, one may combine the high-order and adaptive methods with a multi-GPU implementation for an even more efficient pricing framework.

We conclude this subsection by noting that in many practical financial applications, the pricing models may have significantly more than three stochastic factors. Examples include cross-currency models with multi-factor Gaussian interest rate short models and stochastic volatility (see a relevant discussion in the following section), or basket or index options on dozens of stocks. As a result, standard PDE-based pricing approaches, such as those based on the ADI methods presented in this thesis, may become infeasible, due to the curse of dimensionality associated with high-dimensional PDEs. In such cases, advanced computational techniques for high-dimensional PDEs, such as sparse grids or high-dimensional approximations [61], could be employed.

8.2.2 Modeling of Multi-factor Derivatives

Other Types of Financial Derivatives and Exotic Features

The highly efficient parallel pricing frameworks presented in this thesis, albeit concentrated on long-dated FX interest rate hybrids, can, after straightforward modifications, be used for many other similar long-dated financial derivative hybrids, notably equities and commodities.

The proposed efficient pricing methods for cross-currency interest rate derivatives can be extended to price cross-currency interest rate derivatives with combined exotic features, such as knockout Bermudan cancelable swaps/PRDC swaps, or knockout Bermu-

dan swaptions. Recently, new exotic features, such as "snowball", have become increasingly popular. More specifically, snowball is a generic term for an entire family of structured products which have direct path-dependence on the coupon amounts, i.e. each coupon amount is dependent on the coupon amount immediately preceding it. Cross-currency interest rate derivatives with such a path-dependent exotic feature can also be efficiently handled by the PDE-based pricing framework developed in this thesis for the FX-TARN features. However, most of the interest rate derivatives with snowball features are also Bermudan cancelable. As a result, a combination of the pricing frameworks developed in this thesis for PRDC swaps with Bermudan cancelable and FX-TARN features could be used to price such derivatives.

Enriched Pricing Models

Below, we first discuss deficiencies of pricing models adopted in this thesis, then outline possible enrichments to these models, and discuss how the numerical methods developed in this thesis can be adapted for use in such cases.

Long-dated FX Interest Rate Hybrids

As shown earlier, the price of an FX interest rate hybrid in general, and of a PRDC swap with an exotic feature in particular, is very sensitive to the FX volatility skew. As a result, it is desirable to have a mechanism that could accurately approximate the observed FX volatility skew so that these derivatives can be more precisely priced. Regarding this aspect, a local volatility function for the spot FX rate, such as the one used in the standard model considered in this thesis, while providing better modeling for the skewness of the FX rate than a log-normal one, cannot accurately approximate the market-observed FX volatility smiles.

One approach to address the aforementioned deficiency is to model the variance of the spot FX rate using a stochastic process, such as the Heston model [29], so that the market-

observed FX volatility smiles are more precisely captured. This enrichment to the current model leads to a time-dependent PDE in four state variables – the spot FX rate, domestic and foreign interest short rates, and volatility. In such an application, a GPU-based pricing method, compared to a similar CPU-based pricing method, is expected to deliver even larger speedups and better performance than we observed in this thesis. Another possible approach is to retain the current three-factor model, and instead of having a local volatility function, use a regime switching model for the stochastic volatility of the spot FX rate. Although regime switching models are quite common in option pricing (see, for example, [37, 53]), an extension of such models to cross-currency swaps, especially long-dated FX interest rate hybrids, such as PRDC swaps, has not yet been developed in the open literature. In this case, although the number of stochastic factors remains the same as in this paper, this approach leads to a coupled system of three-dimensional PDEs, with each PDE corresponding to a state of the regime. As a result, the GPU-based parallel ADI scheme presented in this paper needs to be extended to handle systems of PDEs.

In addition to the strong sensitivity to the skew of the FX volatility smiles, another challenge in modeling long-dated FX interest rate hybrids is their very long maturities. Regarding this aspect, another deficiency of the standard model is that one-factor interest rate models cannot provide realistic evolutions of the term structures over a significantly long period of time [8]. To address this deficiency, multi-factor Gaussian interest rate models, such as two- or three-factor Hull-White models, can be used.

In general, as an enriched model for long-dated FX interest rate hybrids may have significantly more than three stochastic factors,[1] direct application of the PDE-based pricing approach developed in this thesis is limited by the curse of dimensionality associated with high-dimensional PDEs. While a MC pricing approach is the popular choice

[1] A cross-currency model with multi-factor Gaussian interest rate short models and stochastic volatility has at least six stochastic factors.

for such an enriched model, the main challenge is to find an effective variance reduction technique. In this case, hybrid pricing methods combining the MC and PDE approaches could be developed. More specifically, a highly accurate numerical solution obtained from the standard model, such as the three-factor model considered in this thesis, via a PDE approach could be used as a control variate to accelerate the convergence of numerical solutions obtained from an enriched model using MC simulations. This phase of the hybrid pricing approach could be built upon the highly efficient PDE-based pricing methods for long-dated FX interest rate hybrids in the standard model developed in this thesis.

We conclude this discussion by noting that, from the perspective of the issuer (a bank), it would be of great interest to investigate and compare the enriched and standard models and their associated assumptions on the volatility and the interest rates. More specifically, for accurate and efficient pricing and hedging purposes of FX interest rate hybrids, it would be desirable (i) to quantify the exposure of long-dated FX interest rate hybrids to the FX volatility skew and to the evolution of the interest rates over a long period of time, and (ii) to study and compare the improvements among the enriched and standard models in these aspects, as well as the associated increases in the computational costs.

Multi-asset Options

Although the Black-Scholes-Merton model is commonly used to model asset prices in a large array of derivative markets, mostly due to its mathematical tractability, the constant volatility of the model cannot account for the volatility smile which is observed in market prices for derivatives. In this regard, multi-asset European and American option pricing with the Heston model for stochastic volatility could be considered. Alternatively, regime switching models [37, 53] for the stochastic volatility could also be considered. The GPU-based parallel ADI timestepping or ADI-AF schemes presented in this thesis could, after

modifications, be used for the pricing of multi-asset options written on two or three assets in these models.

Other enrichments to the standard Black-Scholes-Merton framework could involve a deterministic model (log-normal or local volatility) for the volatility combined with a jump model for the stocks [3, 68]. A more general model would have stochastic volatility combined with a jump model either in the stock prices or in both stock prices and volatility [16]. The price of a European option in such an enriched model is governed by a Partial Integro-Differential Equation (PIDE), while American options lead to a LCP with the same differential operator. In pricing multi-asset options under these enriched models, in addition to high-dimensionality of the differential operator, the treatment of the integral terms arising from the jumps in the models gives rise to substantial extra computational requirements, since their discretization leads to full matrices. In these cases, the GPU-based PDE pricing frameworks developed for multi-asset options in this thesis can be extended for use in combination with certain GPU-based techniques for efficient treatment of the integral terms.

Appendix A

Abbreviations and Notation

A.1 Abbreviations

- **ADI**: Alternating Direction Implicit; **p.2**
- **AF**: Approximate Factorization; . . **p.9**
- **AUD**: Australian Dollar; **p.26**
- **BDF2**: Two-level Backward Difference Formula; **p.132**
- **CN**: Crank-Nicolson; **p.10**
- **CUDA**: Compute Unified Device Architecture; **p.7**
- **FD**: Finite Difference **p.2**
- **FFT**: Fast Fourier Transform; . . . **p.38**
- **FST**: Fast Sine Transform; **p.197**
- **FX**: Foreign Exchange; **p.4**
- **GFLOP**: Giga-Floating Point Operations; **p.162**
- **GMRES**: Generalized Minimal Residual Method; **p.38**
- **GPU**: Graphics Processing Unit; . **p.3**
- **JPY**: Japanese Yen; **p.26**
- **LCP**: Linear Complementarity Problem; **p.9**
- **LIBOR**: London Interbank Offered Rate **p.16**
- **MC**: Monte-Carlo; **p.2**
- **OTC**: Over-the-Counter; **p.1**

- **PDE**: Partial Differential Equation; **p.2**
- **PRDC**: Power Reverse Dual Currency; **p.4**
- **SIMD**: Single Instruction Multiple Data; **p.60**
- **TARN**: Target-Redemption; **p.6**
- **USD**: United States Dollar; **p.26**

A.2 Statistics Collected

- "CPU time": the total CPU computational times in seconds (s.); **p.96**
- "GPU time": the total CPU computational times in seconds (s.); **p.96**
- "CPU (6.16)": the total CPU computational times in milliseconds (ms.) required by the timestep size selector (6.16) for all timesteps; **p.153**
- "GPU (6.16)": the total GPU computational times in milliseconds (ms.) required by the timestep size selector (6.16) for all timesteps; **p.153**
- "speedup": the ratio of the CPU time over the corresponding GPU time; **p.96**
- "value": (PRDC swaps) the spot value of the swap, expressed as a percentage of the domestic notional N_d; **p.96**
- "value": (multi-asset options) the computed spot value of the option; . **p.153**
- "error": difference between the numerical solution and analytical or very accurate numerical solution; **p.96**
- "change": the difference in values between a coarser grid to a finer one; **p.96**
- "$\log_\eta$ ratio": an estimate of the convergence rate; **p.96**

A.3 GPU Parameters

- n_b: the size along the first dimension for each of the 2-D threadblocks used in Step a.1 (set to 32); **p.73**
- p_b: the size along the second dimension for each of the 2-D threadblocks used in Step a.1 (set to 4); **p.73**
- r_t: the size along the first dimension for each of the 2-D threadblocks used in Steps a.2, a.3 and a.4 (set to 32); **p.80**

- c_t: the size along the second dimension for each of the 2-D threadblocks used in Steps a.2, a.3 and a.4 (set to 4); . **p.80**

- s_t: the size for each 1-D threadblock used in the parallelization of the timestep size selector (6.16) (set to 128); **p.143**

A.4 PRDC Swap Pricing Notation

- t: the forward time variable;

- $B(t)$: the value of a bank account at time $t > 0$; **p.15**

- $r(t)$: the instantaneous spot interest rate, or short rate, at time t; **p.15**

- $P(t,T)$: the price at time $t \geq 0$ of a zero-coupon with maturity $T \geq t$; **p.15**

- $L(t,T)$: the simply compounded spot interest rate prevailing at time $t \geq 0$ for the maturity $T \geq t$, referred to as LIBOR; **p.16**

- $\nu(t,T)$: the year fraction between t and $T \geq t$; **p.15**

- $s(t)$: the spot foreign exchange rate prevailing at time t; **p.17**

- $F(t,T)$: the instantaneous forward rate at time t for the maturity T; **p.17**

- $\{T_0 < T_1 < \cdots < T_\beta < T_{\beta+1}\}$: the swap's tenor structure; **p.16**

- ν_α, $\alpha = 1, \ldots, \beta+1$: the year fraction between $T_{\alpha-1}$ and T_α; **p.16**

- $[T_{\alpha-1}, T_\alpha]$, $\alpha = 1, \ldots, \beta+1$: a period of the tenor structure; **p.16**

- T_{α^+} (T_{α^-}), $\alpha = 1, \ldots, \beta+1$: the instant of time just after (before) the date T_α; **p.17**

- C_α, $\alpha = 1, \ldots, \beta$: the PRDC coupon rate at time T_α; **p.23**

- f_α, $\alpha = 1, \ldots, \beta$: the scaling factor of the PRDC coupon rate at time T_α; **p.23**

- $_d$: (subscript "d") associated with " domestic" **p.17**

- $_f$: (subscript "f") associated with " foreign" **p.17**

- $B_d(t)$: the value in domestic currency of the domestic bank account at time $t > 0$; **p.32**
- $W_d(t)$: the Brownian motion for the domestic short rate process; **p.28**
- $W_f(t)$: the Brownian motion for the foreign short rate process; **p.28**
- $W_s(t)$: the Brownian motion for the FX spot rate process; **p.28**
- ρ_{ij}, $i, j = \{d, f, s\}$: the correlation factor between dW_i and dW_j; **p.28**
- N_d: the PRDC swap's notional in the domestic currency; **p.22**
- $P_d(T_{\alpha-1}, T_\alpha)$: the price at time $T_{\alpha-1}$ in domestic currency of the domestic zero coupon bond with maturity T_α; .. **p.17**
- $P_d(T_\alpha)$: a notation for $P_d(T_{\alpha-1}, T_\alpha)$; **p.17**
- $L_d(T_{\alpha-1}, T_\alpha)$: the domestic LIBOR rate prevailing at time $T_{\alpha-1}$ for the maturity T_α; **p.23**
- $L_d(T_\alpha)$: a notation for $L_d(T_{\alpha-1}, T_\alpha)$; **p.23**
- c_i, $i = d, f$: the domestic (d) and foreign (f) coupon rates; **p.23**
- b_i, $i = f, c$: the floor (f) and cap (c) of the PRDC coupon rate; **p.23**
- $r_i(t)$, $i = d, f$: the instantaneous domestic (d) or foreign (f) spot interest rates prevailing at time t; **p.28**
- κ_i, $i = d, f$: the domestic (d) or foreign (f) mean reversion rates of the respective short rates; **p.28**
- σ_i, $i = d, f$: the domestic (d) or foreign (f) volatility functions of the respective short rates; **p.28**
- $\gamma(t, s(t))$: the local volatility function for the spot FX rate; **p.29**
- u: the value of a multi-currency interest rate derivative ($u \equiv u(s, r_d, r_f, t)$); **p.31**
- $u^c_\alpha(t)$: the value at time t of all PRDC coupon amounts scheduled on or after $T_{\alpha+1}$; **p.41**
- $u^f_\alpha(t)$: the value at time t of all funding payments in a PRDC swap scheduled on or after $T_{\alpha+1}$; **p.41**

- $u^e_\alpha(t)$: the value at time t of all fund flows in the offsetting swap scheduled on or after $T_{\alpha+1}$ ($u^e_\alpha(t) = -(u^f_\alpha(t) - u^c_\alpha(t))$); **p.44**

- $u^h_\alpha(t)$: the value at time t of the offsetting Bermudan swaption that has only the dates $\{T_{\alpha+1}, \ldots, T_\beta\}$ as exercise opportunities; **p.44**

- $u^k_\alpha(t)$: the value at time t of the a knockout PRDC swap that has only the dates $\{T_{\alpha+1}, \ldots, T_\beta\}$ as knockout opportunities; **p.46**

- A_c: the target cap of the accumulated PRDC coupon amount; **p.49**

- $a(t)$: the accumulated PRDC coupon amount at time t; **p.50**

- $a_{\alpha^+}(a_{\alpha^-})$: a notation for $a(T_{\alpha^+})$ $(a(T_{\alpha^-}))$; **p.51**

- $u_\alpha(t; a)$: 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, and
  (ii) the total accumulated PRDC coupon amount, including the coupon amount scheduled on T_α, equal to a; **p.53**

- s_∞: the truncated upper boundary in the s-direction; **p.32**

- $r_{d,\infty}$: the truncated upper boundary in the r_d-direction; **p.32**

- $r_{f,\infty}$: the truncated upper boundary in the r_f-direction; **p.32**

- Ω_s: finite-sized truncated computational domain for PRDC swaps; .. **p.32**

- τ: time to maturity; **p.32**

- l: the number of timesteps for each time period; **p.35**

- $n+1$: the number of subintervals in the s-direction; **p.35**

- Δs: the uniform grid mesh width in the s-direction ($\Delta s = \dfrac{s_\infty}{n+1}$); **p.35**

- $p+1$: the number of subintervals in the r_d-direction; **p.35**

- Δr_d: the uniform grid mesh width in the r_d-direction ($\Delta r_d = \dfrac{r_{d,\infty}}{p+1}$); **p.35**

- $q+1$: number of subintervals in the r_f-direction; **p.35**

- Δr_f: uniform grid mesh width in the r_f-direction ($\Delta r_f = \frac{r_{f,\infty}}{q+1}$); **p.35**

A.5 Option Pricing Notation

- t: the forward time variable;
- s_i, $i = 1, 2, 3$: the ith asset price; **p.119**
- $\mathbf{s}$: (s_1, s_2, s_3); **p.119**
- $\bar{T}$: the maturity time of the option; **p.119**
- E: the exercise price (strike) of the option; **p.120**
- r: the constant risk-free interest rate; **p.119**
- d_i, $i = 1, 2, 3$: the constant asset dividend yield of the ith asset ($d_i \geq 0$); **p.119**
- W_i, $i = 1, 2, 3$: the Brownian motion of the ith asset process; **p.119**
- ρ_{ij}, $i, j = 1, 2, 3$: the correlation factors between dW_i and dW_j ($|\rho_{ij}| \leq 1$); **p.119**
- w_i, $i = 1, 2, 3$: the weight of the ith asset in the basket ($w_i > 0$); **p.120**
- z: the value of a European option written on three assets ($z \equiv z(\mathbf{s}, t)$) ; **p.121**
- $z^*(\mathbf{s})$: the payoff function of a European option written on three assets; .. **p.121**
- v: the value of an American option written on three assets $v \equiv v(\mathbf{s}, t)$; .. **p.130**
- $v^*(\mathbf{s})$: the payoff function of an American option written on three assets; **p.121**
- $s_{i,\infty}$, $i = 1, 2, 3$: the truncated upper boundary of the ith asset; **p.119**
- Ω_b: finite-sized truncated computational domain for option pricing; **p.119**
- $\partial\Omega_b$: the boundary of $\partial\Omega_b$; **p.119**
- τ: time to maturity ($\tau = \bar{T} - t$); **p.119**
- $\bar{l}$: number of subintervals in the time direction; **p.120**
- $\Delta\tau_m$: the mth timestep size, not necessarily uniform ($\Delta\tau_m = \tau_m - \tau_{m-1}$); **p.120**

- $\bar{n}+1$: the number of subintervals in the s_1-direction; **p.120**
- $\bar{p}+1$: the number of subintervals in the s_2-direction; **p.120**
- $\bar{q}+1$: the number of subintervals in the s_3-direction; **p.120**
- Δs_i, $i = 1, 2, 3$: the uniform grid mesh width in the s_i-direction; **p.120**
- ζ': the penalty parameter ($\zeta' \to \infty$); **p.130**
- ζ: the penalty factor ($\zeta \sim \zeta' \Delta\tau_m$); **p.132**
- κ: the index of the penalty iteration; **p.134**
- $\mathbf{v}^m$: the vector of approximate values to $v(\mathbf{s}, \tau_m)$; **p.134**
- $\mathbf{v}^*$: the vector of payoff values; . **p.134**
- $\mathbf{P}^{m,(\kappa)}$: the penalty matrix at the κth penalty iteration of timestep τ_m; **p.134**

Appendix B

Glossary of Terms

B.1 Relevant Finance Terms

- **American Option**: an option that can be exercised only once any time during its life; **p.128**
- **Bermudan Swaption**: an option to enter into a swap, and can be exercised only once on a specified set of dates; **p.20**
- **Bermudan Cancelable PRDC Swap**: a PRDC swap that can be canceled by the issuer of the PRDC coupons on a pre-specified set of dates; ... **p.25**
- **Calibration**: methods of implying a model's parameters from the market prices of actively traded options; . **p.29**
- **Cross-currency Interest Rate Derivatives**: financial contracts whose values are contingent on the evolution of the two interest rates, namely the domestic and foreign interest rates, and the spot FX rate that links the two currencies; **p.4**
- **Delta**: the rate of change in the price of a financial derivative with respect to changes in the price of the underlying asset;
- **Derivatives**: Financial contracts whose price depends on, or is derived

from, the evolution of the underlying assets. **p.1**

- **European Option**: an option that can be exercised only at its maturity; **p.120**

- **European Swaption**: an option to enter into a swap, and can be exercised only at its maturity; **p.20**

- **Fixed-for-Floating Swap**: an agreement to exchange a stream of fixed rate of interest on a certain notional principal for a stream of floating rate of interest on the same notional principal, also known as fixed-for-floating interest rate swap; **p.18**

- **Fixed-Rate Coupon**: the initial fund settlement between the two parties of a swap;

- **Fixed-Rate Coupon (PRDC swap)**: the initial fund settlement in a PRDC swap between the issuer and the investor at time T_0, with a positive value indicating a fund inflow for the investor, i.e. the issuer pays the investor, and a negative value indicating otherwise; **p.23**

- **Forward Foreign Exchange (FX) Rate**: the rate at which the foreign currency can be exchanged for the domestic currency on a future date; **p.17**

- **Funding Payment**: the investor's domestic LIBOR payment in a PRDC swap; **p.22**

- **FX-TARN PRDC Swap**: a PRDC swap that pre-maturely terminates on the first date of the swap's tenor structure on which the accumulated PRDC coupon amount, including the coupon amount scheduled on that date, reaches a pre-determined target cap; **p.25**

- **Gamma**: the rate of change in the delta of a financial derivative with respect to changes in the price of the underlying asset; **p.113**

- **Geometric Average**: the nth root of the product of n numbers; **p.141**

- **Implied Volatility**: volatility implied from an option price using the Black-Scholes or a similar model; **p.110**

- **Investor**: the party paying the stream

of domestic floating rate (LIBOR) payments in a PRDC swap; **p.22**

- **Issuer**: the party paying the stream of PRDC coupon amounts in a PRDC swap; **p.22**

- **Knockout PRDC Swap**: a PRDC swap that pre-maturely terminates on the first date of the swap's tenor structure on which the spot FX rate exceeds a specified level; **p.25**

- **Local Volatility Model**: model that treats the volatility as a deterministic function of the current underlying asset level and of time;

- **Local Volatility Function**: a deterministic function of the current underlying asset level and of time for modeling volatility; also see "Local Volatility Model"; **p.5**

- **Log-normal**: a variable has a log-normal distribution when the logarithm of the variable has a normal distribution; **p.119**

- **Long Position**: a position involves the purchase of an asset; **p.44**

- **Martingale**: a zero-drift stochastic process; **p.31**

- **Mean Reversion**: the tendency of a market variable, such as an interest rate, to revert back to a certain long-run average level; **p.28**

- **Numeraire**: defines the units in which security prices are measured; **p.28**

- **Offsetting/Opposite Swap**: a swap with reversed fund flows of a "vanilla" PRDC swap; **p.42**

- **PRDC Coupon**: coupon linked to the spot FX rate prevailing when the coupon rate is set via a certain formula; **p.22**

- **PRDC Swap**: an agreement to exchange a stream of PRDC coupon amounts for a stream of domestic floating rate (domestic LIBOR) payments; **p.22**

- **Short Position**: a position that involves selling an asset; **p.113**

- **Spot Foreign-Exchange Rate**: the number of units of domestic currency per one unit of foreign currency prevailing at a certain time; **p.17**

- **Swap**: an agreement to exchange fund flows in the future according to a prearranged formula; **p.18**

- **Swaption**: an option to enter into a swap, such as a fixed-for-floating interest rate swap; **p.20**

- **Tenor Structure**: a set of fixed dates on which fund flows of a swap are exchanged; **p.16**

- **Term Structure**: relationship between interest rates and their maturities; **p.19**

- **Term Structure of Interest Rate**: see "Term Structure";

- **Vega**: the rate of change in the price of a financial derivative with respect to changes in the volatility; **p.110**

- **Volatility**: a measure of the uncertainty of the return realized on an asset; **p.4**

- **Volatility Skew**: a term used to describe non-symmetrical volatility smile; **p.4**

- **Volatility Smile**: the variation of implied volatility with strike price;

- **Zero-Coupon Bond**: a bond that provides no coupons; **p.15**

B.2 Relevant GPU Terms

- **Barrier Synchronization**: a mechanism to ensure that no thread or warp is allowed to proceed beyond a certain point until the rest have reached it; attained via `__syncthreads()`; **p.78**

- **BlockId**: an unique coordinate assigned to a threadblock during a kernel launch; **p.60**

- **Constant Cache**: a small part of the device memory dedicated to storing constants; **p.59**

- **Device**: the GPU; **p.60**

- **Device Memory**: largest, yet slowest, off GPU-chip memory; accessible by all threads; **p.59**
- **Global Memory**: see device memory;
- **Grid**: an arrangement of the threadblocks generated by a kernel invocation into a 1-D or 2-D array; **p.60**
- **Halo**: the value of a neighbouring gridpoint in an adjacent tile; **p.75**
- **Half-warp**: a group of 16 threads; **p.61**
- **Host**: the CPU; **p.59**
- **Kernel**: a function that runs on the device; **p.60**
- **Memory Coalescing**: coalesced data copies from the global memory to the device memory; **p.62**
- **Shared Memory**: the main memory on a GPU multiprocessor; accessible to all threads in a threadblock; **p.59**
- **Thread**: a copy of the kernel **p.60**
- **Threadblock**: a grouping of all the GPU threads generated by a kernel invocation into 1-, 2-, or 3-D arrays; threads within the same threadblock can communicate and cooperate efficiently; **p.60**
- **ThreadId**: an unique coordinate assigned to a thread during a kernel launch; **p.60**
- **Tile**: a group of data points assigned to a threadblock; **p.73**
- **Warp**: a group of 32 threads; ... **p.61**

Appendix C

Finite Difference and Matrices Formulas

In this appendix, we derive the FD schemes and present explicit formulas for the matrices used in Chapters 3 and 6. For convenience in presentation, we work with a model PDE of the form

$$\frac{\partial u}{\partial \tau} = \mathcal{L}u \equiv \sum_{\substack{i,j=1 \\ i \le j}}^{3} c_{ij} \frac{\partial^2 u}{\partial x_i \partial x_j} + \sum_{i=1}^{3} c_i \frac{\partial u}{\partial x_i} + c_0 u, \tag{C.1}$$

which has the general form of the pricing PDEs used in Chapters 3 and 6. Here, $c_{ij}, i, j = 1, \ldots, 3$, $i \le j$ $c_i, i = 0, \ldots, 3$, are given functions of x_1, x_2, x_3 and/or τ.

We assume that the truncated computational domain is $\Omega = [L_{x_1}, U_{x_1}] \times [L_{x_2}, U_{x_2}] \times [L_{x_3}, U_{x_3}] \subset \mathbb{R}^3$ is a finite rectangular spatial domain. Let the number of subintervals be $n+1$, $p+1$, $q+1$ and l in the x_1-, x_2-, x_3- and τ-directions, respectively. The uniform grid mesh widths in the respective spatial directions are denoted by $\Delta x_1 = \dfrac{U_{x_1} - L_{x_1}}{n+1}$, $\Delta x_2 = \dfrac{U_{x_2} - L_{x_2}}{p+1}$ and $\Delta x_3 = \dfrac{U_{x_3} - L_{x_3}}{q+1}$. Let the gridpoint values of a FD approximation to the solution u are denoted by

$$u_{i,j,k}^{m} \approx u(x_{1i}, x_{2j}, x_{3k}, \tau_m),$$

where $i = 0, \ldots, n+1$, $j = 0, \ldots, p+1$, $k = 0, \ldots, q+1$, $m = 0, \ldots, l$.

C.1 Finite Difference Formulas

In Subsection C.1.1, we derive the two- and three-point standard central FD schemes for the first and second partial derivatives of the space variables, i.e. schemes (3.1a) and (3.1b), respectively, as well as a second-order four-point FD stencil for the cross derivatives, i.e. scheme (3.2). In the case of Dirichlet boundary conditions, such as those arising in the pricing PRDC swaps and multi-asset American options, only these FD schemes are employed. In the case of multi-asset European options, certain one-sided FD schemes, i.e. schemes (6.12), (6.13) and (6.14), are used on the boundary points. These are derived in Subsection C.1.2. We assume that the function u is sufficiently continuously differentiable.

C.1.1 Central FD Formulas

To derive the two- and three-point central FD schemes (3.1a) and (3.1b), we consider the following Taylor expansions about the reference point $(x_{1i}, x_{2j}, x_{3k}, \tau_m)$:

$$u^m_{i+1,j,k} \approx u^m_{i,j,k} + \Delta x_1 \frac{\partial u}{\partial x_1}\Big|^m_{i,j,k} + \frac{1}{2!}(\Delta x_1)^2 \frac{\partial^2 u}{\partial (x_1)^2}\Big|^m_{i,j,k} + \frac{1}{3!}(\Delta x_1)^3 \frac{\partial^3 u}{\partial (x_1)^3}\Big|^m_{i,j,k} + \frac{1}{4!}(\Delta x_1)^4 \frac{\partial^4 u}{\partial (x_1)^4}\Big|^m_{i,j,k} + \mathcal{O}((\Delta x_1)^5), \quad \text{(C.2a)}$$

$$u^m_{i-1,j,k} \approx u^m_{i,j,k} - \Delta x_1 \frac{\partial u}{\partial x_1}\Big|^m_{i,j,k} + \frac{1}{2!}(\Delta x_1)^2 \frac{\partial^2 u}{\partial (x_1)^2}\Big|^m_{i,j,k} - \frac{1}{3!}(\Delta x_1)^3 \frac{\partial^3 u}{\partial (x_1)^3}\Big|^m_{i,j,k} + \frac{1}{4!}(\Delta x_1)^4 \frac{\partial^4 u}{\partial (x_1)^4}\Big|^m_{i,j,k} + \mathcal{O}((\Delta x_1)^5). \quad \text{(C.2b)}$$

We first subtract (C.2b) from (C.2a), and, by rearranging the terms of the resulting expression, we then obtain (3.1a)

$$\frac{\partial u}{\partial x_1}\Big|^m_{i,j,k} \approx \frac{u^m_{i+1,j,k} - u^m_{i-1,j,k}}{2\Delta x_1} + \mathcal{O}((\Delta x_1)^2).$$

We first add (C.2a) and (C.2b), and then, by rearranging the terms of the resulting expression, we then obtain (3.1b)

$$\frac{\partial^2 u}{\partial (x_1)^2}\Big|_{i,j,k}^m \approx \frac{u_{i+1,j,k}^m - 2u_{i,j,k}^m + u_{i-1,j,k}^m}{(\Delta x_1)^2} + \mathcal{O}((\Delta x_1)^2).$$

To derive the second-order four-point FD scheme for the cross derivatives (3.2), we consider the following Taylor expansions about the reference point $(x_{1i}, x_{2j}, x_{3k}, \tau_m)$:

$$\begin{aligned}
u_{i+1,j+1,k}^m \approx\ & u_{i,j,k}^m + \Delta x_1 \frac{\partial u}{\partial x_1}\Big|_{i,j,k}^m + \Delta x_2 \frac{\partial u}{\partial x_2}\Big|_{i,j,k}^m \\
& + \frac{1}{2!}\Big((\Delta x_1)^2 \frac{\partial^2 u}{\partial x_1^2}\Big|_{i,j,k}^m + 2\Delta x_1 \Delta x_2 \frac{\partial^2 u}{\partial x_1 \partial x_2}\Big|_{i,j,k}^m + (\Delta x_2)^2 \frac{\partial^2 u}{\partial x_2^2}\Big|_{i,j,k}^m\Big) \\
& + \frac{1}{3!}\Big((\Delta x_1)^3 \frac{\partial^3 u}{\partial x_1^3}\Big|_{i,j,k}^m + 3(\Delta x_1)^2 \Delta x_2 \frac{\partial^3 u}{\partial x_1^2 \partial x_2}\Big|_{i,j,k}^m + 3\Delta x_1 (\Delta x_2)^2 \frac{\partial^3 u}{\partial x_1 \partial x_2^2}\Big|_{i,j,k}^m \\
& \qquad + (\Delta x_2)^3 \frac{\partial^3 u}{\partial x_2^3}\Big|_{i,j,k}^m\Big) \\
& + \frac{1}{4!}\Big((\Delta x_1)^4 \frac{\partial^4 u}{\partial x_1^4}\Big|_{i,j,k}^m + 4(\Delta x_1)^3 \Delta x_2 \frac{\partial^4 u}{\partial x_1^3 \partial x_2}\Big|_{i,j,k}^m + 6(\Delta x_1)^2 (\Delta x_2)^2 \frac{\partial^4 u}{\partial x_1^2 \partial x_2^2}\Big|_{i,j,k}^m \\
& \qquad + 4\Delta x_1 (\Delta x_2)^3 \frac{\partial^4 u}{\partial x_1 \partial x_2^3}\Big|_{i,j,k}^m + (\Delta x_2)^4 \frac{\partial^4 u}{\partial x_2^4}\Big|_{i,j,k}^m\Big) \\
& + \mathcal{O}\Big((\Delta x_1)^5, (\Delta x_2)^5, (\Delta x_1)(\Delta x_2)^4, (\Delta x_1)^2(\Delta x_2)^3, (\Delta x_1)^3(\Delta x_2)^2, (\Delta x_1)^4(\Delta x_2)\Big),
\end{aligned} \tag{C.3a}$$

$$\begin{aligned}
u_{i-1,j-1,k}^m \approx\ & u_{i,j,k}^m - \Delta x_1 \frac{\partial u}{\partial x_1}\Big|_{i,j,k}^m - \Delta x_2 \frac{\partial u}{\partial x_2}\Big|_{i,j,k}^m \\
& + \frac{1}{2!}\Big((\Delta x_1)^2 \frac{\partial^2 u}{\partial x_1^2}\Big|_{i,j,k}^m + 2\Delta x_1 \Delta x_2 \frac{\partial^2 u}{\partial x_1 \partial x_2}\Big|_{i,j,k}^m + (\Delta x_2)^2 \frac{\partial^2 u}{\partial x_2^2}\Big|_{i,j,k}^m\Big) \\
& - \frac{1}{3!}\Big((\Delta x_1)^3 \frac{\partial^3 u}{\partial x_1^3}\Big|_{i,j,k}^m + 3(\Delta x_1)^2 \Delta x_2 \frac{\partial^3 u}{\partial x_1^2 \partial x_2}\Big|_{i,j,k}^m + 3\Delta x_1 (\Delta x_2)^2 \frac{\partial^3 u}{\partial x_1 \partial x_2^2}\Big|_{i,j,k}^m \\
& \qquad + (\Delta x_2)^3 \frac{\partial^3 u}{\partial x_2^3}\Big|_{i,j,k}^m\Big) \\
& + \frac{1}{4!}\Big((\Delta x_1)^4 \frac{\partial^4 u}{\partial x_1^4}\Big|_{i,j,k}^m + 4(\Delta x_1)^3 \Delta x_2 \frac{\partial^4 u}{\partial x_1^3 \partial x_2}\Big|_{i,j,k}^m + 6(\Delta x_1)^2 (\Delta x_2)^2 \frac{\partial^4 u}{\partial x_1^2 \partial x_2^2}\Big|_{i,j,k}^m \\
& \qquad + 4\Delta x_1 (\Delta x_2)^3 \frac{\partial^4 u}{\partial x_1 \partial x_2^3}\Big|_{i,j,k}^m + (\Delta x_2)^4 \frac{\partial^4 u}{\partial x_2^4}\Big|_{i,j,k}^m\Big) \\
& + \mathcal{O}\Big((\Delta x_1)^5, (\Delta x_2)^5, (\Delta x_1)(\Delta x_2)^4, (\Delta x_1)^2(\Delta x_2)^3, (\Delta x_1)^3(\Delta x_2)^2, (\Delta x_1)^4(\Delta x_2)\Big),
\end{aligned} \tag{C.3b}$$

$$
\begin{aligned}
u^m_{i-1,j+1,k} \approx\ & u^m_{i,j,k} - \Delta x_1 \frac{\partial u}{\partial x_1}\Big|^m_{i,j,k} + \Delta x_2 \frac{\partial u}{\partial x_2}\Big|^m_{i,j,k} \\
& + \frac{1}{2!}\Big((\Delta x_1)^2 \frac{\partial^2 u}{\partial x_1^2}\Big|^m_{i,j,k} - 2\Delta x_1 \Delta x_2 \frac{\partial^2 u}{\partial x_1 \partial x_2} + (\Delta x_2)^2 \frac{\partial^2 u}{\partial x_2^2}\Big|^m_{i,j,k} \Big) \\
& + \frac{1}{3!}\Big(-(\Delta x_1)^3 \frac{\partial^3 u}{\partial x_1^3}\Big|^m_{i,j,k} + 3(\Delta x_1)^2 \Delta x_2 \frac{\partial^3 u}{\partial x_1^2 \partial x_2}\Big|^m_{i,j,k} - 3\Delta x_1 (\Delta x_2)^2 \frac{\partial^3 u}{\partial x_1 \partial x_2^2}\Big|^m_{i,j,k} \\
& \qquad\qquad + (\Delta x_2)^3 \frac{\partial^3 u}{\partial x_2^3}\Big|^m_{i,j,k} \Big) \\
& + \frac{1}{4!}\Big((\Delta x_1)^4 \frac{\partial^4 u}{\partial x_1^4}\Big|^m_{i,j,k} - 4(\Delta x_1)^3 \Delta x_2 \frac{\partial^4 u}{\partial x_1^3 \partial x_2}\Big|^m_{i,j,k} + 6(\Delta x_1)^2 (\Delta x_2)^2 \frac{\partial^4 u}{\partial x_1^2 \partial x_2^2}\Big|^m_{i,j,k} \\
& \qquad - 4\Delta x_1 (\Delta x_2)^3 \frac{\partial^4 u}{\partial x_1 \partial x_2^3}\Big|^m_{i,j,k} + (\Delta x_2)^4 \frac{\partial^4 u}{\partial x_2^4}\Big|^m_{i,j,k} \Big) \\
& + \mathcal{O}\Big((\Delta x_1)^5, (\Delta x_2)^5, (\Delta x_1)(\Delta x_2)^4, (\Delta x_1)^2(\Delta x_2)^3, (\Delta x_1)^3(\Delta x_2)^2, (\Delta x_1)^4(\Delta x_2) \Big),
\end{aligned}
\tag{C.3c}
$$

$$
\begin{aligned}
u^m_{i+1,j-1,k} \approx\ & u^m_{i,j,k} + \Delta x_1 \frac{\partial u}{\partial x_1}\Big|^m_{i,j,k} - \Delta x_2 \frac{\partial u}{\partial x_2}\Big|^m_{i,j,k} \\
& + \frac{1}{2!}\Big((\Delta x_1)^2 \frac{\partial^2 u}{\partial x_1^2}\Big|^m_{i,j,k} - 2\Delta x_1 \Delta x_2 \frac{\partial^2 u}{\partial x_1 \partial x_2} + (\Delta x_2)^2 \frac{\partial^2 u}{\partial x_2^2}\Big|^m_{i,j,k} \Big) \\
& + \frac{1}{3!}\Big((\Delta x_1)^3 \frac{\partial^3 u}{\partial x_1^3}\Big|^m_{i,j,k} - 3(\Delta x_1)^2 \Delta x_2 \frac{\partial^3 u}{\partial x_1^2 \partial x_2}\Big|^m_{i,j,k} + 3\Delta x_1 (\Delta x_2)^2 \frac{\partial^3 u}{\partial x_1 \partial x_2^2}\Big|^m_{i,j,k} \\
& \qquad\qquad - (\Delta x_2)^3 \frac{\partial^3 u}{\partial x_2^3}\Big|^m_{i,j,k} \Big) \\
& + \frac{1}{4!}\Big((\Delta x_1)^4 \frac{\partial^4 u}{\partial x_1^4}\Big|^m_{i,j,k} - 4(\Delta x_1)^3 \Delta x_2 \frac{\partial^4 u}{\partial x_1^3 \partial x_2}\Big|^m_{i,j,k} + 6(\Delta x_1)^2 (\Delta x_2)^2 \frac{\partial^4 u}{\partial x_1^2 \partial x_2^2}\Big|^m_{i,j,k} \\
& \qquad - 4\Delta x_1 (\Delta x_2)^3 \frac{\partial^4 u}{\partial x_1 \partial x_2^3}\Big|^m_{i,j,k} + (\Delta x_2)^4 \frac{\partial^4 u}{\partial x_2^4}\Big|^m_{i,j,k} \Big) \\
& + \mathcal{O}\Big((\Delta x_1)^5, (\Delta x_2)^5, (\Delta x_1)(\Delta x_2)^4, (\Delta x_1)^2(\Delta x_2)^3, (\Delta x_1)^3(\Delta x_2)^2, (\Delta x_1)^4(\Delta x_2) \Big).
\end{aligned}
\tag{C.3d}
$$

By first adding (C.3b) to (C.3a), and then subtracting (C.3c) and (C.3d) from the resulting sum, we get

$$
\begin{aligned}
u^m_{i+1,j+1,k} + u^m_{i-1,j-1,k} - u^m_{i-1,j+1,k} - u^m_{i+1,j-1,k} \approx\ & 4\Delta x_1 \Delta x_2 \frac{\partial^2 u}{\partial x_1 \partial x_2}\Big|^m_{i,j,k} \\
& + \frac{1}{4!}\Big(16(\Delta x_1)^3 \Delta x_2 \frac{\partial^4 u}{\partial x_1^3 \partial x_2}\Big|^m_{i,j,k} + 16\Delta x_1 (\Delta x_2)^3 \frac{\partial^4 u}{\partial x_1 \partial x_2^3}\Big|^m_{i,j,k} \Big) \\
& + \text{higher-order terms.}
\end{aligned}
$$

By rearranging the terms of the above expression, we obtain (3.2)

$$\frac{\partial^2 u}{\partial x_1 \partial x_2}\Big|_{i,j,k}^{m} \approx \frac{u_{i+1,j+1,k}^{m} + u_{i-1,j-1,k}^{m} - u_{i-1,j+1,k}^{m} - u_{i+1,j-1,k}^{m}}{4\Delta x_1 \Delta x_2} + \mathcal{O}((\Delta x_1)^2, (\Delta x_2)^2).$$

Note that, as verified above, each of the central FD formulas has a second-order truncation error.

C.1.2 One-sided FD Formulas

To derive the one-sided FD schemes (6.12a) and (6.13a), we consider the Taylor expansion about the reference point $(x_{10}, x_{2j}, x_{3k}, \tau_m)$ at the boundary for which $x_1 = L_{x_1}$:

$$u_{1,j,k}^{m} \approx u_{0,j,k}^{m} + \Delta x_1 \frac{\partial u}{\partial x_1}\Big|_{0,j,k}^{m} + \mathcal{O}((\Delta x_1)^2). \tag{C.4}$$

By rearranging the terms of (C.4), we obtain (6.12a)

$$\frac{\partial u}{\partial x_1}\Big|_{0,j,k}^{m} = \frac{u_{1,j,k}^{m} - u_{0,j,k}^{m}}{\Delta x_1} + \mathcal{O}(\Delta x_1).$$

Similarly, to derive the one-sided backward FD scheme (6.12b), we consider the Taylor expansion about the reference point $(x_{1n+1}, x_{2j}, x_{3k}, \tau_m)$ at the boundary for which $x_1 = U_{x_1}$:

$$u_{n,j,k}^{m} \approx u_{n+1,j,k}^{m} - \Delta x_1 \frac{\partial u}{\partial x_1}\Big|_{n+1,j,k}^{m} + \mathcal{O}((\Delta x_1)^2). \tag{C.5}$$

By rearranging the terms of (C.5), we obtain (6.12b)

$$\frac{\partial u}{\partial x_1}\Big|_{n+1,j,k}^{m} = \frac{u_{n+1,j,k}^{m} - u_{n,j,k}^{m}}{\Delta x_1} + \mathcal{O}(\Delta x_1).$$

To derive the one-sided FD formula (6.13a), we consider the following Taylor expansions about the reference point $(x_{10}, x_{2j}, x_{3k}, \tau_m)$ at the boundary for which $x_1 = L_{x_1}$:

$$u_{0,j+1,k}^{m} \approx u_{0,j,k}^{m} + \Delta x_2 \frac{\partial u}{\partial x_2}\Big|_{0,j,k}^{m} + \frac{1}{2!}(\Delta x_2)^2 \frac{\partial^2 u}{\partial (x_2)^2}\Big|_{0,j,k}^{m} + \frac{1}{3!}(\Delta x_2)^3 \frac{\partial^3 u}{\partial (x_2)^3}\Big|_{0,j,k}^{m} + \frac{1}{4!}(\Delta x_2)^4 \frac{\partial^4 u}{\partial (x_2)^4}\Big|_{0,j,k}^{m} + \mathcal{O}((\Delta x_2)^5), \tag{C.6a}$$

$$u^m_{0,j-1,k} \approx u^m_{0,j,k} - \Delta x_2 \frac{\partial u}{\partial x_2}\Big|^m_{0,j,k} + \frac{1}{2!}(\Delta x_2)^2 \frac{\partial^2 u}{\partial (x_2)^2}\Big|^m_{0,j,k} - \frac{1}{3!}(\Delta x_2)^3 \frac{\partial^3 u}{\partial (x_2)^3}\Big|^m_{0,j,k} + \frac{1}{4!}(\Delta x_2)^4 \frac{\partial^4 u}{\partial (x_2)^4}\Big|^m_{0,j,k} + \mathcal{O}((\Delta x_2)^5), \quad \text{(C.6b)}$$

$$\begin{aligned} u^m_{1,j+1,k} \approx\ & u^m_{0,j,k} + \Delta x_1 \frac{\partial u}{\partial x_1}\Big|^m_{0,j,k} + \Delta x_2 \frac{\partial u}{\partial x_2}\Big|^m_{0,j,k} \\ & + \frac{1}{2!}\Big((\Delta x_1)^2 \frac{\partial^2 u}{\partial x_1^2}\Big|^m_{0,j,k} + 2\Delta x_1 \Delta x_2 \frac{\partial^2 u}{\partial x_1 \partial x_2}\Big|^m_{0,j,k} + (\Delta x_2)^2 \frac{\partial^2 u}{\partial x_2^2}\Big|^m_{0,j,k}\Big) \\ & + \frac{1}{3!}\Big((\Delta x_1)^3 \frac{\partial^3 u}{\partial x_1^3}\Big|^m_{0,j,k} + 3(\Delta x_1)^2 \Delta x_2 \frac{\partial^3 u}{\partial x_1^2 \partial x_2}\Big|^m_{0,j,k} + 3\Delta x_1 (\Delta x_2)^2 \frac{\partial^3 u}{\partial x_1 \partial x_2^2}\Big|^m_{0,j,k} \\ & \qquad + (\Delta x_2)^3 \frac{\partial^3 u}{\partial x_2^3}\Big|^m_{0,j,k}\Big) \\ & + \frac{1}{4!}\Big((\Delta x_1)^4 \frac{\partial^4 u}{\partial x_1^4}\Big|^m_{0,j,k} + 4(\Delta x_1)^3 \Delta x_2 \frac{\partial^4 u}{\partial x_1^3 \partial x_2}\Big|^m_{0,j,k} + 6(\Delta x_1)^2 (\Delta x_2)^2 \frac{\partial^4 u}{\partial x_1^2 \partial x_2^2}\Big|^m_{0,j,k} \\ & \qquad + 4\Delta x_1 (\Delta x_2)^3 \frac{\partial^4 u}{\partial x_1 \partial x_2^3}\Big|^m_{0,j,k} + (\Delta x_2)^4 \frac{\partial^4 u}{\partial x_2^4}\Big|^m_{0,j,k}\Big) \\ & + \mathcal{O}\Big((\Delta x_1)^5, (\Delta x_2)^5, (\Delta x_1)(\Delta x_2)^4, (\Delta x_1)^2(\Delta x_2)^3, (\Delta x_1)^3(\Delta x_2)^2, (\Delta x_1)^4 \Delta x_2\Big), \end{aligned} \quad \text{(C.6c)}$$

$$\begin{aligned} u^m_{1,j-1,k} \approx\ & u^m_{0,j,k} + \Delta x_1 \frac{\partial u}{\partial x_1}\Big|^m_{0,j,k} - \Delta x_2 \frac{\partial u}{\partial x_2}\Big|^m_{0,j,k} \\ & + \frac{1}{2!}\Big((\Delta x_1)^2 \frac{\partial^2 u}{\partial x_1^2}\Big|^m_{0,j,k} - 2\Delta x_1 \Delta x_2 \frac{\partial^2 u}{\partial x_1 \partial x_2}\Big|^m_{0,j,k} + (\Delta x_2)^2 \frac{\partial^2 u}{\partial x_2^2}\Big|^m_{0,j,k}\Big) \\ & + \frac{1}{3!}\Big((\Delta x_1)^3 \frac{\partial^3 u}{\partial x_1^3}\Big|^m_{0,j,k} - 3(\Delta x_1)^2 \Delta x_2 \frac{\partial^3 u}{\partial x_1^2 \partial x_2}\Big|^m_{0,j,k} + 3\Delta x_1 (\Delta x_2)^2 \frac{\partial^3 u}{\partial x_1 \partial x_2^2}\Big|^m_{0,j,k} \\ & \qquad - (\Delta x_2)^3 \frac{\partial^3 u}{\partial x_2^3}\Big|^m_{0,j,k}\Big) \\ & + \frac{1}{4!}\Big((\Delta x_1)^4 \frac{\partial^4 u}{\partial x_1^4}\Big|^m_{0,j,k} - 4(\Delta x_1)^3 \Delta x_2 \frac{\partial^4 u}{\partial x_1^3 \partial x_2}\Big|^m_{0,j,k} + 6(\Delta x_1)^2 (\Delta x_2)^2 \frac{\partial^4 u}{\partial x_1^2 \partial x_2^2}\Big|^m_{0,j,k} \\ & \qquad - 4\Delta x_1 (\Delta x_2)^3 \frac{\partial^4 u}{\partial x_1 \partial x_2^3}\Big|^m_{0,j,k} + (\Delta x_2)^4 \frac{\partial^4 u}{\partial x_2^4}\Big|^m_{0,j,k}\Big) \\ & + \mathcal{O}\Big((\Delta x_1)^5, (\Delta x_2)^5, (\Delta x_1)(\Delta x_2)^4, (\Delta x_1)^2(\Delta x_2)^3, (\Delta x_1)^3(\Delta x_2)^2, (\Delta x_1)^4 \Delta x_2\Big). \end{aligned} \quad \text{(C.6d)}$$

By first adding (C.6c) and (C.6b), then subtracting (C.6a) and (C.6d) from the resulting

sum, we get

$$u^m_{1,j+1,k}+u^m_{0,j-1,k}-u^m_{0,j+1,k}-u^m_{1,j-1,k} \approx 2\Delta x_1\Delta x_2\frac{\partial^2 u}{\partial x_1\partial x_2}\Big|^m_{0,j,k}+\frac{1}{3!}\Big(6(\Delta x_1)^2\Delta x_2\frac{\partial^3 u}{\partial x_1^2\partial x_2}\Big|^m_{0,j,k}\Big)$$
$$+\frac{1}{4!}\Big(8(\Delta x_1)^3\Delta x_2\frac{\partial^4 u}{\partial x_1^3\partial x_2}\Big|^m_{0,j,k}+8\Delta x_1(\Delta x_2)^3\frac{\partial^4 u}{\partial x_1\partial x_2^3}\Big|^m_{0,j,k}\Big)+\text{higher-order terms.} \quad \text{(C.7)}$$

Rearranging (C.7) gives (6.13a)

$$\frac{\partial^2 u}{\partial x_1\partial x_2}\Big|^m_{0,j,k} \approx \frac{u^m_{1,j+1,k}+u^m_{0,j-1,k}-u^m_{0,j+1,k}-u^m_{1,j-1,k}}{2\Delta x_1\Delta x_2}+\mathcal{O}\Big(\Delta x_1,(\Delta x_2)^2\Big).$$

The derivation of (6.13b) can be obtained in a similar fashion.

To derive the one-sided FD formula (6.14), we consider the following Taylor expansions

$$u^m_{0,1,k} \approx u^m_{0,0,k}+\Delta x_2\frac{\partial u}{\partial x_2}\Big|^m_{0,0,k}+\frac{1}{2!}(\Delta x_2)^2\frac{\partial^2 u}{\partial (x_2)^2}\Big|^m_{0,0,k}+\frac{1}{3!}(\Delta x_2)^3\frac{\partial^3 u}{\partial (x_2)^3}\Big|^m_{0,0,k}+\mathcal{O}((\Delta x_2)^4), \quad \text{(C.8a)}$$

$$u^m_{1,0,k} \approx u^m_{0,0,k}+\Delta x_1\frac{\partial u}{\partial x_1}\Big|^m_{0,0,k}+\frac{1}{2!}(\Delta x_1)^2\frac{\partial^2 u}{\partial (x_1)^2}\Big|^m_{0,0,k}+\frac{1}{3!}(\Delta x_1)^3\frac{\partial^3 u}{\partial (x_1)^3}\Big|^m_{0,0,k}+\mathcal{O}((\Delta x_1)^4), \quad \text{(C.8b)}$$

$$\begin{aligned} u^m_{1,1,k} \approx\ & u^m_{0,0,k}+\Delta x_1\frac{\partial u}{\partial x_1}\Big|^m_{0,0,k}+\Delta x_2\frac{\partial u}{\partial x_2}\Big|^m_{0,0,k} \\ & +\frac{1}{2!}\Big((\Delta x_1)^2\frac{\partial^2 u}{\partial x_1^2}\Big|^m_{0,0,k}+2\Delta x_1\Delta x_2\frac{\partial^2 u}{\partial x_1\partial x_2}\Big|^m_{0,0,k}+(\Delta x_2)^2\frac{\partial^2 u}{\partial x_2^2}\Big|^m_{0,0,k}\Big) \\ & +\frac{1}{3!}\Big((\Delta x_1)^3\frac{\partial^3 u}{\partial x_1^3}\Big|^m_{0,0,k}+3(\Delta x_1)^2\Delta x_2\frac{\partial^3 u}{\partial x_1^2\partial x_2}\Big|^m_{0,0,k}+3\Delta x_1(\Delta x_2)^2\frac{\partial^3 u}{\partial x_1\partial x_2^2}\Big|^m_{0,0,k} \\ & \qquad +(\Delta x_2)^3\frac{\partial^3 u}{\partial x_2^3}\Big|^m_{0,0,k}\Big) \\ & +\mathcal{O}\Big((\Delta x_1)^4,(\Delta x_2)^4,(\Delta x_1)(\Delta x_2)^3,(\Delta x_1)^2(\Delta x_2)^2,(\Delta x_1)^3\Delta x_2\Big). \end{aligned} \quad \text{(C.8c)}$$

Subtracting (C.8a) and (C.8b) from (C.8c) gives

$$u_{1,1,k}-u^m_{0,1,k}-u^m_{1,0,k} \approx -u^m_{0,0,k}+\Delta x_1\Delta x_2\frac{\partial^2 u}{\partial x_1\partial x_2}\Big|^m_{0,0,k}+\frac{1}{3!}\Big(3(\Delta x_1)^2\Delta x_2\frac{\partial^3 u}{\partial x_1^2\partial x_2}\Big|^m_{0,0,k}$$
$$+3\Delta x_1(\Delta x_2)^2\frac{\partial^3 u}{\partial x_1\partial x_2^2}\Big|^m_{0,0,k}\Big)+\text{higher-order terms.} \quad \text{(C.9)}$$

Rearranging (C.9) gives (6.14)

$$\frac{\partial^2 u}{\partial x_1 \partial x_2}\Big|^m_{0,0,k} \approx \frac{u_{1,1,k} + u^m_{0,0,k} - u^m_{0,1,k} - u^m_{1,0,k}}{\Delta x_1 \Delta x_2} + \mathcal{O}\Big(\Delta x_1, \Delta x_2\Big).$$

Note that, as verified above, each of the one-sided FD formulas has a first-order truncation error.

C.2 Matrix Formulas

In this section, we present explicit formulas for the matrix $\mathbf{A}^m$ arising from the FD discretization of the differential operator $\mathcal{L}$, for the matrix $\mathbf{A}^m_0$, which is the part of $\mathbf{A}^m$ that comes from the FD discretizations of the cross derivative terms, and for the matrices $\mathbf{A}^m_1$, $\mathbf{A}^m_2$ and $\mathbf{A}^m_3$, which are the parts of $\mathbf{A}^m$ that correspond to the spatial derivatives in the x_1-, x_2- and x_3-directions, respectively. For simplicity, we present explicit formulas for these matrices only for the case of Dirichlet boundary conditions. Those arising in the case of linear boundary conditions can be obtained via a straightforward modification of the formulas presented in this section.

For presentation purposes, we further adopt the following notation. We denote by $\mathbf{I}_z$ the identity matrix of size $z \times z$, by $\mathbf{Q}_z$ and $\mathbf{T}_z$ the tridiagonal matrix representations of the classic first- and second-order FD operators on z points, $z = \{n, p, q\}$, respectively. Explicit formulas for the matrices $\mathbf{Q}_z$ and $\mathbf{T}_z$ are

$$\mathbf{Q}_z = \begin{pmatrix} 0 & 1 & 0 & \dots & 0 \\ -1 & 0 & 1 & \dots & 0 \\ \ddots & \ddots & \ddots & \ddots & \ddots \\ \ddots & -1 & 0 & 1 & \ddots \\ \ddots & \ddots & \ddots & \ddots & \ddots \\ 0 & \dots & -1 & 0 & 1 \\ 0 & \dots & 0 & -1 & 0 \end{pmatrix}_{z \times z}, \qquad \mathbf{T}_z = \begin{pmatrix} -2 & 1 & 0 & \dots & 0 \\ 1 & -2 & 1 & \dots & 0 \\ \ddots & \ddots & \ddots & \ddots & \ddots \\ \ddots & 1 & -2 & 1 & \ddots \\ \ddots & \ddots & \ddots & \ddots & \ddots \\ 0 & \dots & 1 & -2 & 1 \\ 0 & \dots & 0 & 1 & -2 \end{pmatrix}_{z \times z},$$

Note the changes in first and last rows of these matrices, due to Dirichlet boundary conditions.

Unless otherwise stated, assume that the gridpoints are ordered in the x_1-, x_2- , then x_3- directions. We denote by $\mathbf{C}_0^m$ the coefficient matrix for the u term, by $\mathbf{C}_i^m$, $i = 1, \ldots, 3$, the coefficient matrices for the terms $\dfrac{\partial u}{\partial x_i}$, by $\mathbf{C}_{ij}^m$, $i, j = 1, \ldots, 3$, $i \leq j$, the coefficient matrices for the terms $\dfrac{\partial^2 u}{\partial x_i \partial x_j}$ in (C.1). These coefficient matrices are diagonal matrices of size $npq \times npq$, and have the values of the respective coefficient function at interior gridpoints on the diagonal. Using tensor product notation [47], the matrix $\mathbf{A}_i^m$, $i = 0, \ldots, 3$ can be written in a compact form as follows

$$\mathbf{A}_0^m = \frac{\mathbf{C}_{12}^m}{4\Delta x_1 \Delta x_2}(\mathbf{I}_q \otimes \mathbf{Q}_p \otimes \mathbf{Q}_n) + \frac{\mathbf{C}_{13}^m}{4\Delta x_1 \Delta x_3}(\mathbf{Q}_q \otimes \mathbf{I}_p \otimes \mathbf{Q}_n) + \frac{\mathbf{C}_{23}^m}{4\Delta x_2 \Delta x_3}(\mathbf{Q}_q \otimes \mathbf{Q}_p \otimes \mathbf{I}_n), \tag{C.10a}$$

$$\mathbf{A}_1^m = \frac{\mathbf{C}_1^m}{2\Delta x_1}(\mathbf{I}_q \otimes \mathbf{I}_p \otimes \mathbf{Q}_n) + \frac{\mathbf{C}_{11}^m}{(\Delta x_1)^2}(\mathbf{I}_q \otimes \mathbf{I}_p \otimes \mathbf{T}_n) + \frac{\mathbf{C}_0^m}{3} \tag{C.10b}$$

$$\mathbf{A}_2^m = \frac{\mathbf{C}_2^m}{2\Delta x_2}(\mathbf{I}_q \otimes \mathbf{Q}_p \otimes \mathbf{I}_n) + \frac{\mathbf{C}_{22}^m}{(\Delta x_2)^2}(\mathbf{I}_q \otimes \mathbf{T}_p \otimes \mathbf{I}_n) + \frac{\mathbf{C}_0^m}{3} \tag{C.10c}$$

$$\mathbf{A}_3^m = \frac{\mathbf{C}_3^m}{2\Delta x_3}(\mathbf{Q}_q \otimes \mathbf{I}_p \otimes \mathbf{I}_n) + \frac{\mathbf{C}_{33}^m}{(\Delta x_3)^2}(\mathbf{T}_q \otimes \mathbf{I}_p \otimes \mathbf{I}_n) + \frac{\mathbf{C}_0^m}{3} \tag{C.10d}$$

and $\mathbf{A}^m$ is simply given by

$$\mathbf{A}^m = \mathbf{A}_0^m + \mathbf{A}_1^m + \mathbf{A}_2^m + \mathbf{A}_3^m.$$

It is important to point out that, with the current ordering, i.e. the gridpoints are ordered in the x_1-, x_2- , then x_3- directions, the matrix $\mathbf{A}_1^m$ in (C.10b) is block-diagonal with tridiagonal blocks, due to the fact that both the matrices $(\mathbf{I}_q \otimes \mathbf{I}_p \otimes \mathbf{Q}_n)$ and $(\mathbf{I}_q \otimes \mathbf{I}_p \otimes \mathbf{T}_n)$ are block-diagonal with tridiagonal blocks. However, the matrices $\mathbf{A}_2^m$ and $\mathbf{A}_3^m$, given by (C.10c) and (C.10d), respectively, are not. In order for each of these two matrices to have a block-diagonal structure, the gridpoints must be ordered in the direction of respective spatial variable. For example, if the gridpoints are ordered in the x_2-, x_1- , then x_3- directions, then the matrix $\mathbf{A}_2^m$ is block-diagonal with tridiagonal blocks, while,

if the gridpoints are ordered in the x_3-, x_1- , then x_2- directions, then the matrix $\mathbf{A}_3^m$ is block-diagonal with tridiagonal blocks.

Appendix D

Pricing PRDC Swaps with Preconditioned Iterative Methods

This appendix presents an application of iterative methods combined with preconditioning techniques for the solution of the resulting block banded linear system at each timestep arising from the CN timestepping scheme in the context of PRDC swaps, i.e. linear system (3.3). Our approach is to use the preconditioned GMRES method with the preconditioner solved by FFT techniques.

The remainder of this appendix is organized as follows. In Section D.1, we discuss our choice for the preconditioner and the associated FFT techniques In Section D.2, we present selected numerical results to demonstrate the accuracy and efficiency of the numerical methods applied to pricing Bermudan cancelable PRDC swaps.

D.1 GMRES with a Preconditioner Solved by FFT Techniques

For the reader's convenience, we repeat (3.3) below

$$(\mathbf{I} - \frac{1}{2}\Delta\tau\mathbf{A}^m)\mathbf{u}^m = (\mathbf{I} + \frac{1}{2}\Delta\tau\mathbf{A}^{m-1})\mathbf{u}^{m-1} + \frac{1}{2}\Delta\tau(\mathbf{g}^m + \mathbf{g}^{m-1}).$$

To avoid the high computational cost of direct methods, we choose to apply an iterative method to solve (3.3), namely GMRES. We choose GMRES because the matrix $\mathbf{I} - \frac{1}{2}\Delta\tau\mathbf{A}^m$ is neither symmetric nor positive (semi-) definite in general. Thus, commonly used iterative schemes, such as the Conjugate Gradient method, designed primarily for symmetric positive-definite systems, are not likely to converge. A detailed description of the GMRES method, and the "restarted" version of the method, can be found in [25]. In our implementation, at each timestep except the first, the initial guess for GMRES is constructed by linear extrapolation of the numerical solution from the two previous timesteps. It is important to mention that, due to the use of this initial guess and the use of a preconditioner which is described below, only a few iterations (usually 3 or 4) are required for the GMRES method to converge, hence no restarting is needed.

We use as preconditioner for $\mathbf{I} - \frac{1}{2}\Delta\tau\mathbf{A}^m$ the matrix $\mathbf{P}$ arising from the discretization of

$$\frac{\partial u}{\partial \tau} = \frac{\partial^2 u}{\partial s^2} + \frac{\partial^2 u}{\partial r_d^2} + \frac{\partial^2 u}{\partial r_f^2} + u$$

using the CN timestepping method. This choice of preconditioner is motivated by the work in [10]. In the rest of this section, we describe an efficient algorithm for solving linear systems of the form $\mathbf{Pv} = \mathbf{b}$. Following Appendix C.2, using tensor products, we can write the matrix $\mathbf{P}$ as

$$\mathbf{P} = \mathbf{I} - \frac{1}{2}\Delta\tau\Big(\frac{1}{(\Delta s)^2}(\mathbf{I}_q \otimes \mathbf{I}_p \otimes \mathbf{T}_n) + \frac{1}{(\Delta r_d)^2}(\mathbf{I}_q \otimes \mathbf{T}_p \otimes \mathbf{I}_n) + \frac{1}{(\Delta r_f)^2}(\mathbf{T}_q \otimes \mathbf{I}_p \otimes \mathbf{I}_n) + \mathbf{I}\Big). \quad \text{(D.1)}$$

Here, we adopt the same notation for the matrices $\mathbf{I}_z$ and $\mathbf{T}_z$, $z = \{n, p, q\}$, as in Appendix C.2. It is known [47] that, if we let $\mathbf{V}_n$ be the matrix of normalized eigenvectors of $\mathbf{T}_n$ and $\mathbf{F}_n$ be the Discrete Sine Transform (DST) matrix of order n, then we have $\mathbf{V}_n^{-1} = \mathbf{F}_n$. Similarly, we have $\mathbf{V}_p^{-1} = \mathbf{F}_p$ and $\mathbf{V}_q^{-1} = \mathbf{F}_q$. A useful observation is that $\mathbf{P}$ can be block-diagonalized via

$$\begin{aligned}\mathbf{B} &= (\mathbf{V}_q^{-1} \otimes \mathbf{V}_p^{-1} \otimes \mathbf{I}_n)\mathbf{P}(\mathbf{V}_q \otimes \mathbf{V}_p \otimes \mathbf{I}_n) \\ &= \mathbf{I} - \frac{1}{2}\Delta\tau\Big(\frac{1}{(\Delta s)^2}(\mathbf{I}_q \otimes \mathbf{I}_p \otimes \mathbf{T}_n) + \frac{1}{(\Delta r_d)^2}(\mathbf{I}_q \otimes \mathbf{\Lambda}_p \otimes \mathbf{I}_n) + \frac{1}{(\Delta r_f)^2}(\mathbf{\Lambda}_q \otimes \mathbf{I}_p \otimes \mathbf{I}_n) + \mathbf{I}\Big),\end{aligned} \tag{D.2}$$

where $\mathbf{\Lambda}_p = \mathbf{V}_p^{-1}\mathbf{T}_p\mathbf{V}_p$ is a diagonal matrix with the eigenvalues of $\mathbf{T}_p$ on its diagonal, and $\mathbf{\Lambda}_q$ is defined similarly. The matrix $\mathbf{B}$ has pq blocks, each of size $n \times n$. These observations give rise to the following fast solver for the preconditioner using FFT techniques, more specifically, using Fast Sine Transforms (FSTs). (Note that the main computational requirement of an FST is that of an FFT.) Consider the linear system $\mathbf{Pv} = \mathbf{b}$. Note that $\mathbf{b} \in \mathbb{R}^{npq}$ and for the sake of presentation, denote by $\mathbf{b}_{q\times pn}$ the $q \times pn$ matrix with entries being the components of $\mathbf{b}$ laid out in q rows and pn columns, column by column. Taking (D.2) into account, the solution $\mathbf{v}$ to the system $\mathbf{Pv} = \mathbf{b}$ can be written as

$$\mathbf{v} = \mathbf{P}^{-1}\mathbf{b} = (\mathbf{V}_q \otimes \mathbf{V}_p \otimes \mathbf{I}_n)\mathbf{B}^{-1}(\mathbf{V}_q^{-1} \otimes \mathbf{V}_p^{-1} \otimes \mathbf{I}_n)\mathbf{b} = (\mathbf{F}_q^{-1} \otimes \mathbf{F}_p^{-1} \otimes \mathbf{I}_n)\mathbf{B}^{-1}(\mathbf{F}_q \otimes \mathbf{F}_p \otimes \mathbf{I}_n)\mathbf{b}. \tag{D.3}$$

The FST algorithm for performing the computation in (D.3) consists of the following steps:

1. Perform the FST on each of the pn columns of $(\mathbf{b}_{pn\times q})^T$ to obtain $(\mathbf{b}^{(1)})_{q\times pn} = \mathbf{F}_q(\mathbf{b}_{pn\times q})^T$.

2. Perform the FST on each of the qn columns of $((\mathbf{b}^{(1)})_{qn\times p})^T$ to obtain $(\mathbf{b}^{(2)})_{p\times qn} = \mathbf{F}_p(\mathbf{b}^{(1)}_{qn\times p})^T$, or, equivalently $\mathbf{b}^{(2)} = (\mathbf{F}_q \otimes \mathbf{F}_p \otimes \mathbf{I}_n)\mathbf{b}$.

3. Solve the block-diagonal system $\mathbf{Bb}^{(3)} = \mathbf{b}^{(2)}$.

4. Perform the inverse FST on each of the pn columns of $((\mathbf{b}^{(3)})_{pn\times q})^T$ to obtain

$(\mathbf{b}^{(4)})_{q\times pn} = \mathbf{F}_q^{-1}((\mathbf{b}^{(3)})_{pn\times q})^T$.

5. Perform the inverse FST on each of the qn columns of $(\mathbf{b}^{(4)}_{qn\times p})^T$ to obtain $\mathbf{v}_{p\times qn} = \mathbf{F}_p^{-1}(\mathbf{b}^{(4)}_{qn\times p})^T$, or equivalently $\mathbf{v} = (\mathbf{F}_q^{-1}\otimes\mathbf{F}_p^{-1}\otimes\mathbf{I}_n)\mathbf{B}^{-1}(\mathbf{F}_q\otimes\mathbf{F}_p\otimes\mathbf{I}_n)\mathbf{b}$.

The above five steps form an FFT technique for solving the linear system $\mathbf{Pv} = \mathbf{b}$. Clearly, the five steps involve $\mathcal{O}(npq\log(npq))$ flops, which is an almost optimal computational complexity for solving an $npq\times npq$ linear system.

D.2 Numerical Results

leverage	l (τ)	n (s)	p (r_d)	q (r_f)	underlying swap ADI – GMRES value (%)	change	ratio	cancelable swap ADI – GMRES value	change	ratio	performance ADI time (s)	GMRES time (s)	GMRES avg. iter.
low	4	12	6	6	-11.41			11.39			0.79	0.87	3.5
	8	24	12	12	-11.16	2.5e-3		11.30	8.6e-4		8.71	7.92	3.5
	16	48	24	24	-11.11	5.0e-4	5.0	11.28	1.7e-4	5.0	165.78	164.38	3.6
	32	96	48	48	-11.10	1.0e-4	5.0	11.28	4.1e-5	4.1	3172.98	3195.75	3.6
medium	4	12	6	6	-13.87			13.42					
	8	24	12	12	-12.94	9.3e-3		13.76	3.3e-3				
	16	48	24	24	-12.75	1.9e-3	4.7	13.85	9.5e-4	3.5			
	32	96	48	48	-12.70	5.0e-4	3.9	13.88	2.6e-4	3.6			
high	4	12	6	6	-13.39			18.50					
	8	24	12	12	-11.54	1.8e-2		19.31	8.1e-3				
	16	48	24	24	-11.19	3.5e-3	5.2	19.56	2.5e-3	3.2			
	32	96	48	48	-11.12	8.0e-4	4.3	19.62	5.4e-4	4.6			

Table D.2.1: Values of the underlying PRDC swap and cancelable PRDC swap with FX skew for various leverage levels; "change" is the difference in the solution from the coarser grid; "ratio" is the ratio of the changes on successive grids; "avg. iter." is the average number of iterations.

The code for the experiments carried out in this section is written in MATLAB, and double precision is used. Selected numerical results are presented in Table D.2.1.

In terms of accuracy, both the ADI and the preconditioned GMRES methods give identical prices to four digits of accuracy for the underlying PRDC swap and the can-

celable PRDC swap. (Hence, we do not present prices obtained by the two methods separately to save space in the table.) As expected from the discretization methods, second-order accuracy is obtained.

Since for different leverage levels, the computation times of the methods considered are virtually the same, we report only the selected computation statistics for the low leverage case in the last three columns of Table D.2.1 obtained from pricing the underlying PRDC swap. We note that, asymptotically, for each doubling of the number of timesteps and gridpoints in all directions, both the ADI and GMRES computation times increase by a factor of about 19, which is close to the optimal factor of 16. It is also evident that the ADI method and the GMRES method are almost equivalently efficient, with the ADI method being asymptotically a little bit more efficient. It is worth noting that the average number of iterations required by the GMRES method per timestep is quite small, and more importantly, is independent of the size of discretized problem. These results show the combined effect of using an effective preconditioner and a good initial guess based on linear extrapolation.

Appendix E

Miscellaneous Derivations

E.1 Fixed Notional Method

In this section, we discuss the underlying idea of the fixed notional method. Although we present the method in the context of PRDC swaps, the idea can be employed to price the floating leg in other types of interest rate swaps. First, we consider a LIBOR payment, for example $\nu_\alpha L_d(T_\alpha)N_d$ scheduled at time T_α for the period $[T_{\alpha-1}, T_\alpha]$. This LIBOR inflow is equivalent to (i) receiving the fixed notional N_d at time $T_{\alpha-1}$, and (ii) paying the fixed notional N_d at time T_α. More specifically, because the domestic LIBOR rate $L_d(T_\alpha)$ is known at time $T_{\alpha-1}$, the issuer, who receives the domestic LIBOR payments, could:

1. at time $T_{\alpha-1}$, upon receiving the amount N_d from the investor, invest this mount at rate $L_d(T_\alpha)$
2. at time T_α, receive the interest amount $\nu_\alpha L_d(T_\alpha)N_d$ and the principal N_d;
3. return the principal N_d to the investor.

The above three steps gives a net fund inflow of size $\nu_\alpha L_d(T_\alpha)N_d$ at time T_α, which is exactly the same as the LIBOR inflow scheduled at this time.

Alternatively, the above idea could be interpreted as follows. Since the domestic LIBOR rate $L_d(T_\alpha)$ is known at time $T_{\alpha-1}$, the value at time $T_{\alpha-1}$ of the LIBOR payment $\nu_\alpha L_d(T_\alpha)N_d$ is simply

$$\nu_\alpha L_d(T_\alpha)N_d P_d(T_\alpha),$$

which, by replacing $L_d(T_\alpha)$ with (2.7), becomes

$$(1 - P_d(T_\alpha))N_d = N_d - N_d P_d(T_\alpha). \tag{E.1}$$

The amount $-N_d P_d(T_\alpha)$ is the value at time $T_{\alpha-1}$ of the fund outflow of size N_d at time T_α. As a result, (E.1) essentially represents a fund inflow and a fund outflow, both of size N_d, at time $T_{\alpha-1}$ and T_α, respectively.

We now extend this idea to multiple periods. Consider the domestic LIBOR fund inflows scheduled on dates $\{T_{\alpha+1}, \ldots T_\beta\}$, as expressed in Figure E.1.1 (a). We are concerned with the value at time T_α of these fund inflows. We apply the fixed notional method described earlier to each of these LIBOR fund inflows, which is illustrated by Figure E.1.1 (b).

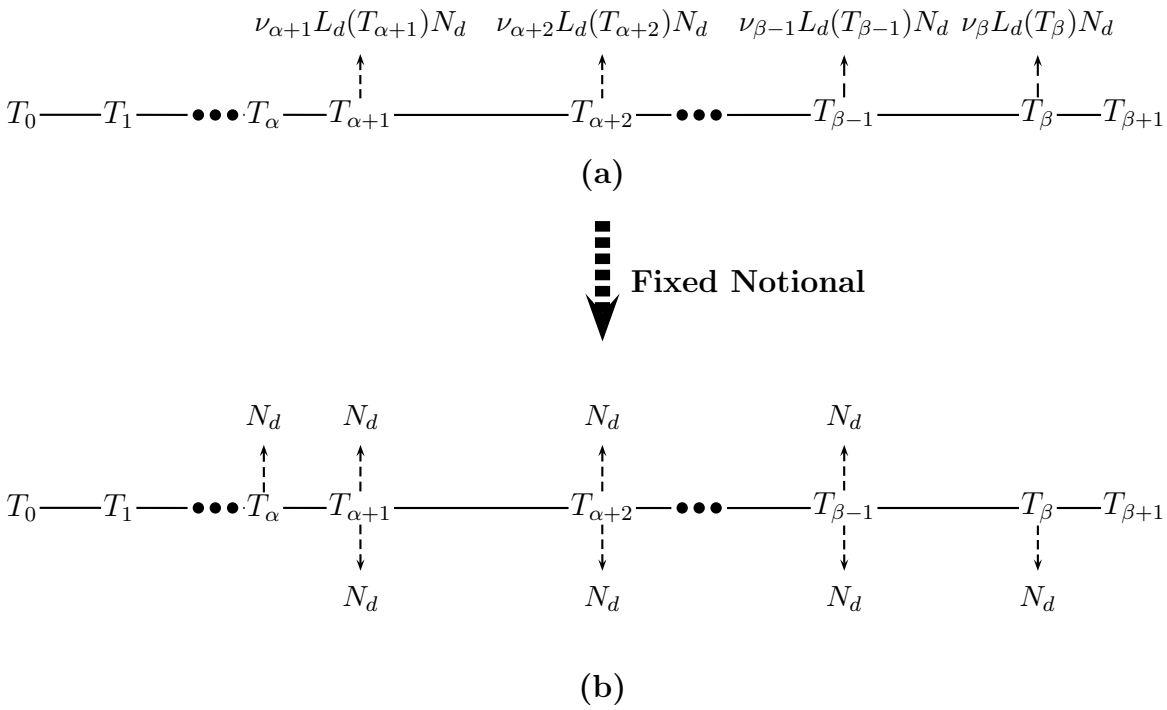


Figure E.1.1: An illustration of the fixed notional method

As obvious from Figure E.1.1 (b), this methods gives rise to (i) a zero fund flow at each of $\{T_{\alpha+1}, \ldots T_{\beta-1}\}$, (ii) a fund inflow of size N_d at time T_α, and (iii) a fund outflow of size N_d at time T_β. As a result, the value at time T_α of all the domestic LIBOR fund inflows scheduled on dates $\{T_{\alpha+1}, \ldots T_\beta\}$ is simply given by

$$N_d - N_d P_d(T_\alpha, T_\beta) = (1 - P_d(T_\alpha, T_\beta))N_d. \tag{E.2}$$

E.2 A Derivation for the Dynamics of Geometric Average Processes

In this section, we derive the dynamics (6.30)–(6.33), for the geometric average process $g(t)$ defined by (6.29). Note that, since each of the $s_i(t)$, $i = 1, 2, 3$, follows a standard

geometric Brownian motion, we have [63]

$$s_i(t) = s_i(0)e^{\left((r-d_i-\frac{1}{2}(\sigma_i)^2)t+\sigma_i W_t(t)\right)}, \quad i = 1, 2, 3. \tag{E.3}$$

It follows from the definition of $g(t)$ in (6.29) and (E.3) that

$$\begin{aligned} g(t) &= \Big(\prod_{i=1}^{3} s_i(t)\Big)^{\frac{1}{3}} \\ &= \Big(\prod_{i=1}^{3} s_i(0)\Big)^{\frac{1}{3}} e^{\left\{\frac{1}{3}\left(\sum_{i=1}^{3}((r-d_i-\frac{1}{2}(\sigma_i)^2)t+\sigma_i W_i(t))\right)\right\}} \\ &= g(0)e^{\left\{\frac{1}{3}\left(\sum_{i=1}^{3}(r-d_i-\frac{1}{2}(\sigma_i)^2)t\right)+\hat{B}_g(t)\right\}}, \end{aligned} \tag{E.4}$$

where

$$\begin{aligned} g(0) &= \Big(\prod_{i=1}^{3} s_i(0)\Big)^{\frac{1}{3}}, \\ \hat{B}_g(t) &= \frac{1}{3}\sum_{i=1}^{3} \sigma_i W_i(t). \end{aligned}$$

We now consider the quantity $d\hat{B}_g(t)d\hat{B}_g(t)$, referred to as the *quadratic variation* of $\hat{B}_g$ up to time t. We have

$$\begin{aligned} d\hat{B}_g(t)d\hat{B}_g(t) &= d\Big(\frac{1}{3}\sum_{i=1}^{3}\sigma_i W_i(t)\Big)d\Big(\frac{1}{3}\sum_{i=1}^{3}\sigma_i W_i(t)\Big) \\ &= \frac{1}{3^2}\Big(\sum_{i=1}^{3}\sigma_i dW_i(t)\Big)\Big(\sum_{i=1}^{3}\sigma_i dW_i(t)\Big) \\ &= \frac{1}{3^2}\Big(\sum_{i,j=1}^{3}\rho_{ij}\sigma_i\sigma_j\Big)dt. \end{aligned} \tag{E.5}$$

Let

$$\sigma_g = \Big(\frac{1}{3^2}\sum_{i,j=1}^{3}\rho_{ij}\sigma_i\sigma_j\Big)^{\frac{1}{2}}, \tag{E.6}$$

and consider

$$B_g(t) = \frac{\hat{B}_g(t)}{\sigma_g} = \frac{1}{3\sigma_g}\sum_{i=1}^{3}\sigma_i W_i(t).$$

By (E.5) and the definition of $B_g(t)$, we conclude that $dB_g(t)dB_g(t) = dt$. Furthermore, $B_g(t)$ is a (1-D) continuous martingale, with $B_g(0) = 0$. Hence, by Lévy's Theorem (see

Theorem 4.6.4 in [63]), $B_g(t)$ is a standard Brownian motion. Thus, by this result and (E.4), we can rewrite $g(t)$ in terms of $B_g(t)$ as

$$g(t) = g(0)e^{\left\{\frac{1}{3}\left(\sum_{i=1}^{3}(r-d_i-\frac{1}{2}(\sigma_i)^2)t\right)+\sigma_g B_g(t)\right\}},$$

which can be further re-arranged into

$$\begin{aligned} g(t) &= g(0)e^{\left\{\left(r-\left(\frac{1}{3}\sum_{i=1}^{3}(d_i+\frac{1}{2}\sigma_i^2)-\frac{1}{2}\sigma_g^2\right)-\frac{1}{2}\sigma_g^2\right)t+\sigma_g B_g(t)\right\}} \\ &= g(0)e^{\left\{\left(r-d_g-\frac{1}{2}\sigma_g^2\right)t+\sigma_g B_g(t)\right\}}, \end{aligned} \tag{E.7}$$

where

$$d_g = \frac{1}{3}\sum_{i=1}^{3}\left(d_i+\frac{1}{2}\sigma_i^2\right)-\frac{1}{2}\sigma_g^2. \tag{E.8}$$

It is obvious from (E.7) that $g(t)$ follows a standard geometric Brownian motion with the dynamics specified by

$$\frac{dg(t)}{g(t)} = (r-d_g)dt + \sigma_g dB_g(t),$$

where the volatility σ_g and the "effective" dividend yield d_g are respectively defined by (E.6) and (E.6).

Note that the approach presented above can be generalized to find the dynamics of the geometric average process defined on an arbitrary number of geometric Brownian motions. An alternative approach to derive the dynamics for $g(t)$ is to apply the multi-dimensional Itô's formula to (6.29). However, in this case, the computation is much more involved than the one presented above.

Bibliography

[1] L. A. Abbas-Turki and B. Lapeyre. American options pricing on multi-core graphic cards. In *Proceedings of International Conference on Business Intelligence and Financial Engineering*, pages 307–311. IEEE Computer Society, 2009.

[2] 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*, pages 1–8. IEEE Computer Society, 2009.

[3] L. Andersen and J. Andreasen. Jump-diffusion processes: Volatility smile fitting and numerical methods for option pricing. *Review of Derivatives Research*, 4:231–262, 2000.

[4] L. B.G. Andersen and V. V. Piterbarg. *Interest Rate Modeling*. Atlantic Financial Press, first edition, 2010.

[5] J. Andreasen. Back to the future. *Risk magazine*, 18:104–109, 2005.

[6] B.A.Wade, A.Q.M. Khaliq, M.Yousuf, J. Vigo-Aguiard, and R. Deiningere. On smoothing of the Crank-Nicolson scheme and higher order schemes for pricing barrier options. *Journal of Computational and Applied Mathematics*, 204:144–158, 2007.

[7] F. Black and M. Scholes. The pricing of options and corporate liabilities. *Journal of Political Economics*, 81:637–659, 1973.

[8] D. Brigo and F. Mercurio. *Interest Rate Models - Theory and Practice*. Springer, second edition, 2006.

[9] C. Christara and D. M. Dang. Adaptive and high-order methods for valuing American options. *Journal of Computational Finance*, 14(4):73–113, Summer 2011.

[10] C. C. Christara and K. S. Ng. Fast Fourier transform solvers and preconditioners for quadratic spline collocation. *BIT*, 42(4):702–739, 2002.

[11] J. Cox, J. Ingersoll, and S. Ross. An intertemporal general equilibrium model of asset prices. *Econometrica*, 53:363–384, 1985.

[12] J. Cox, J. Ingersoll, and S. Ross. A theory of the term structure of interest rates. *Econometrica*, 53:385–407, 1985.

[13] J.D. Craig and A.D. Sneyd. An alternating-direction implicit scheme for parabolic equations with mixed derivatives. *Comp. Math. Appl.*, 16:341–350, 1988.

[14] M.A.H. Dempster and J.P. Hutton. *Numerical valuation of cross-currency swaps and swaptions*. Cambridge University Press, 1997.

[15] J. Douglas and H.H. Rachford. On the numerical solution of heat conduction problems in two and three space variables. *Trans. Amer. Math. Soc*, 82:421–439, 1956.

[16] D. Duffie, J. Pan, and K. Singleton. Transform analysis and asset pricing for affine jump-diffusions. *Econometrica*, 68:1343–1376, 2000.

[17] D. J. Duffy. *Finite Difference methods in financial engineering: a Partial Differential Equation approach*. John Wiley & Sons, 2006.

[18] P. A. Forsyth and K. Vetzal. Quadratic convergence for valuing American options using a penalty method. *SIAM J. Sci. Comput.*, 23(6):2095–2122, 2002.

[19] M. C. Fu, S. B. Laprise, D. B. Madan, Y. Su, and R. Wu. Pricing American options: A comparison of Monte Carlo simulation approaches. *Journal of Computational Finance*, 4(3):39–88, Spring 2001.

[20] A. Gaikwad and I. M. Toke. GPU-based sparse grid technique for solving multi-dimensional options pricing PDEs. In *Proceedings of the 2nd Workshop on High Performance Computational Finance*, pages 1–8. IEEE Computer Society, 2009.

[21] P. Glasserman. *Monte Carlo Methods in Financial Engineering*. Springer, first edition, 2003.

[22] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI-2: Advanced Features of the Message Passing Interface*. MIT Press, first edition, 1999.

[23] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: portable parallel programming with the message-passing interface*. MIT Press, second edition, 1999.

[24] A. V. Haastrecht and A. Pelsser. Generic pricing of FX, inflation and stock options under stochastic interest rates and stochastic volatility. *SSRN eLibrary*, 2009. Available at http://ssrn.com/paper=1197262.

[25] W. Hackbush. *Iterative Solution of Large Sparse Systems of Equations*. Springer, 1993.

[26] E. Hairer and G. Wanner. *Solving ordinary differential equations II: stiff and differential-algebraic problems*. Springer-Verlag, second edition, 1996.

[27] M. Harris. Mapping computational concepts to GPUs. In *GPU Gems 2*, pages 493–508. NVIDIA, 2005.

[28] M. Harris, S. Sengupta, and J. D. Owens. Parallel prefix sum (scan) with CUDA. In *GPU Gems 3*, pages 851–877. NVIDIA, 2007.

[29] S.L. Heston. A closed-form solution for options with stochastic volatility with applications to bond and currency options. *Review of Financial Studies*, 6:327–343, 1993.

[30] R.W. Hockney and C.R. Jesshope. *Parallel Computers 2: Architecture, Programming and Algorithms*. Brisol, England; Philadelphia, PA, USA: A. Hilger, second edition, 1988.

[31] J. Hull and A. White. One factor interest rate models and the valuation of interest rate derivative securities. *Journal of Financial and Quantitative Analysis*, 28(2):235–254, 1993.

[32] John C. Hull. *Options, Futures, and Other Derivatives*. Prentice Hall, seventh edition, 2008.

[33] W. Hundsdorfer. Accuracy and stability of splitting with stabilizing corrections. *Appl. Numer. Math*, 42:213–233, 2002.

[34] W. Hundsdorfer and J.G. Verwer. *Numerical Solution of Time-Dependent Advection-Diffusion-Reaction Equations, Springer Series in Computational Mathematics*, volume 33. Springer-Verlag, Berlin, 2003.

[35] K. J. in't Hout and S. Foulon. ADI finite difference schemes for option pricing in the Heston model with correlation. *International Journal Of Numerical Analysis And Modeling*, 7(2):303–320, 2010.

[36] K.J. in't Hout and B.D. Welfert. Unconditional stability of second-order ADI schemes applied to multi-dimensional diffusion equations with mixed derivative terms. *Appl. Numer. Math*, 59:677–692, 2009.

[37] I. Popova J. Duan and P. Ritchken. Option pricing under regime switching. *Quantitative Finance*, 2:116–132, 2002.

[38] R. Jarrow and Y. Yildirim. Pricing treasury inflation protected securities and related derivatives using an HJM model. *Journal of Financial and Quantitative Analysis*, 38(2):409–430, 2003.

[39] L. Jiang and M. Dai. Convergence of binomial tree methods for European and American path-dependent options. *SIAM J. Numer. Anal.*, 42:1094–1109, 2004.

[40] H. Johnson. Options on the maximum or the minimum of several assets. *Journal of Financial and Quantitative Analysis*, 22:277–283, 1987.

[41] I. Karatzas and S. E. Shreve. *Brownian Motion and Stochastic Calculus*. Springer, first edition, 1997.

[42] P. Kovalov, V. Linetsky, and M. Marcozzi. Pricing multi-asset American options: A finite element method-of-lines with smooth penalty. *Journal of Scientific Computing*, 33:209–237, 2007.

[43] Y.K. Kwok. *Mathematical Models of Financial Derivatives*. Springer Finance, Springer-Verlag, Singapore, second edition, 1998.

[44] C.C.W. Leentvaar and C.W. Oosterlee. On coordinate transformation and grid stretching for sparse grid pricing of basket options. *Journal of Computational and Applied Mathematics*, 222:193–209, 2008.

[45] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym. NVIDIA Tesla: A unified graphics and computing architecture. *IEEE Micro*, 28:39–55, 2008.

[46] A. Lipton. *Mathematical Methods for Foreign Exchange: A Financial Engineers Approach*. World Scientific, 2001.

[47] C. Van Loan. *Computational Frameworks for the Fast Fourier Transform*. Society for Industrial and Applied Mathematics, 1992.

[48] F. A. Longstaff and E. S. Schwartz. Valuing American options by simulation: A simple least-squares approach. *The Review of Financial Study*, 14:113–149, 2001.

[49] A. Eswaran M.A.H. Dempster and D.G. Richards. Wavelet methods in PDE valuation of financial derivatives. In *Lecture Notes in Computer Science*, pages 217–239. Springer Berlin, 2000.

[50] R. C. Merton. The theory of rational option pricing. *Bell Journal of Economics and Management Science*, 4:141–183, 1973.

[51] 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*, pages 1–4. IEEE Computer Society, 2010.

[52] M. Musiela and M. Rutkowski. *Martingale Methods in Financial Modelling*. Springer, second edition, 2005.

[53] S. F. Gray N. Bollen and R. E. Whaley. Regime switching in foreign exchange rates: Evidence from currency option prices. *Journal of Econometrics*, 94(1–2):239–276, 2000.

[54] B. Nielsen, O. Skavhaug, and A. Tveito. Penalty and front-fixing methods for the numerical solution of American option problems. *Journal of Computational Finance*, 5(4):69–97, 2002.

[55] B. F. Nielsen, O. Skavhaug, and A. Tveito. Penalty methods for the numerical solution of American multi-asset option problems. *Journal of Computational and Applied Mathematics*, 222:3–16, 2008.

[56] J. Nocedel and S. J. Wright. *Numerical Optimization*. Springer, second edition, 2006.

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

[58] V. Piterbarg. Smiling hybrids. *Risk magazine*, 19(2):66–70, 2006.

[59] D. M Pooley, K. R. Verzal, and P. A. Forsyth. Convergence remedies for non-smooth payoffs in option pricing. *Journal of Computational Finance*, 6:25–40, Summer 2003.

[60] R. Rannacher. Finite element solution of diffusion problems with irregular data. *Numerische Mathematik*, 43(2):309–327, 1984.

[61] C. Reisinger and G. Wittum. Efficient hierarchical approximation of high-dimensional option pricing problems. *SIAM J. Sci. Comput.*, 29(1):440–458, 2007.

[62] Y. Saad. *Iterative methods for sparse linear systems*. Society for Industrial and Applied Mathematics (SIAM), second edition, 2003.

[63] S. E. Shreve. *Stochastic calculus for finance II: Continuous-time models*. Springer, first edition, 2004.

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

[65] T. P. Stefański and T. D. Drysdale. Acceleration of the 3D ADI-FDTD method using Graphics Processor Units. In *Proceedings of the International Microwave Symposium*, pages 241–244. IEEE Computer Society, 2009.

[66] D. Tavella and C. Randall. *Pricing financial instruments: The finite difference method*. John Wiley & Sons, Chichester, 2000.

[67] 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, pages 1–8. IEEE Computer Society, 2010.

[68] J. Toivanen. Numerical valuation of European and American options under Kou's jump-diffusion model. *SIAM J. Sci. Comput.*, 30:1949–1970, 2008.

[69] P. Wilmott, J. Dewynne, and S. Howison. *Option pricing: Mathematical Models and Computation*. Oxford Financial Press, 1993.

[70] P. Wilmott, S. Howison, and J. Dewynne. *Mathematics of Financial Derivatives*. Cambridge Univeristy Press, 1995.

[71] 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*, 8:65–92, Fall, 2004.

[72] T. P. Witelski and M. Bowen. ADI schemes for higher-order nonlinear diffusion equations. *Applied Numerical Mathematics*, 45:331–351, 2001.

[73] Y. Zhu, X. Wu, and I. Chern. *Derivative Securities and Difference Methods*. Springer Verlag, New York, 2004.

[74] R. Zvan, P. A. Forsyth, and K. Vetzal. Penalty methods for American options with stochastic volatility. *Journal of Computational and Applied Mathematics*, 91:199–218, 1998.

[75] R. Zvan, P.A. Forsyth, and K.R. Vetzal. Discrete Asian barrier options. *Journal of Computational Finance*, 3:41–68, 1999.