Mathematical Finance MSc Dissertation MTH775P, 2018/19

Accelerated Grids

Optimizing Solvers for Financial Partial Differential Equations

Mustafa Berke Erdis, ID 180883925

Supervisor: Dr. Sebastian del Bano Rollin

A thesis presented for the degree of
Master in Sciences in *Mathematical Finance*

School of Mathematical Sciences
and *School of Economics and Finance*
Queen Mary University of London

Declaration of original work

This declaration is made on August 20, 2019.

Student's Declaration: I, Mustafa Berke Erdis, hereby declare that the work in this thesis is my original work. I have not copied from any other students' work, work of mine submitted elsewhere, or from any other sources except where due reference or acknowledgement is made explicitly in the text, nor has any part been written for me by another person.

Referenced text has been flagged by:

1. Using italic fonts, **and**
2. using quotation marks "...", **and**
3. explicitly mentioning the source in the text.

This work is dedicated to my family.

Acknowledgements

Here you thank people that have helped you in the journey.
Vivamus vehicula leo a justo. Quisque nec augue. Morbi mauris wisi, aliquet vitae, dignissim eget, sollicitudin molestie, ligula. In dictum enim sit amet risus. Curabitur vitae velit eu diam rhoncus hendrerit. Vivamus ut elit. Praesent mattis ipsum quis turpis. Curabitur rhoncus neque eu dui. Etiam vitae magna. Nam ullamcorper. Praesent interdum bibendum magna. Quisque auctor aliquam dolor. Morbi eu lorem et est porttitor fermentum. Nunc egestas arcu at tortor varius viverra. Fusce eu nulla ut nulla interdum consectetuer. Vestibulum gravida. Morbi mattis libero sed est.

Abstract

Here you write a short summary, around 10 lines, of your work.
Vivamus vehicula leo a justo. Quisque nec augue. Morbi mauris wisi, aliquet vitae, dignissim eget, sollicitudin molestie, ligula. In dictum enim sit amet risus. Curabitur vitae velit eu diam rhoncus hendrerit. Vivamus ut elit. Praesent mattis ipsum quis turpis. Curabitur rhoncus neque eu dui. Etiam vitae magna. Nam ullamcorper. Praesent interdum bibendum magna. Quisque auctor aliquam dolor. Morbi eu lorem et est porttitor fermentum. Nunc egestas arcu at tortor varius viverra. Fusce eu nulla ut nulla interdum consectetuer. Vestibulum gravida. Morbi mattis libero sed est.

Queen Mary University of London
12th August 2019

Contents

Chapter 1

Introduction

In Ancient Greece, Thales was scorned for his poverty. Later that year, Thales utilized his skills in astrology to forecast an increase in olive yields. Using his limited capital, he rented oil presses in winter. Months later, over the oil making season, many people rushed to the presses because of the high yields that Thales predicted. As he rented the presses over the winter, he forced the terms he pleased. Thales showed it was easy for philosophers to be rich if they chose it and practically used the first financial derivative product [1].

In the modern world, financial derivatives are contracts between two or more parties. The value of the contract depends on one or several underlying assets. Commonly the assets are currencies, equities, bonds, interest rates, market indices or commodities. The vanilla call option gives the right but not the obligation to buy the underlying asset at the expiry date at a previously agreed strike price. Essentially, Thales bought call options for oil presses. If the olive yields didn't come as Thales expected he didn't have the obligation to use the olive presses. On the other hand, the vanilla put option gives the right but not the obligation to sell the underlying asset at the expiry date at a previously agreed strike price. Practical applications of the options include hedging or speculating the future asset price. Hence, accurately pricing the options is crucial for an efficient and mature financial market.

Merton and Scholes received the 1997 Nobel Prize in Economic Science for this work [19].

1.1 Motivation of the Project

Derivative pricing in the real world is a computationally intensive task. The existing numerical methods for partial differential equations are all constrained by the computational complexity. Being fast when evaluating new information is critical for the operations of hedge funds and investment banks. Therefore optimizing the existing numerical methods with hardware and software that can be installed on a trading floor is crucial. Goal of the project is to provide efficient methods for pricing options.

Purpose of this project to optimize numerical solutions of parabolic PDEs by testing high performance computing techniques and comparing compilers/os/32bit/64bit. The idea of this project is to study how to take advantage of this parallelism and explore how much faster we can make these calculations.

Included in your 'Introduction' section should be a clear summary of what you have achieved in the project work presented, such as any new results, generalisations, corollaries, examples, new connections, or computer investigations. The thesis is organized as follows. Chapter 2 presents an introduction to financial derivatives, Black-Scholes model and finite difference models. Chapter3 extends the 2d and gives examples of two dimensional heat equation. Chapter 4 develops the numerical methods and techniques considered to solve the PDE-based models for the option pricing problems. In the same chapter, finite difference method with improved algorithms to solve a large tridiagonal systems is discussed. Chapter 5 shows the numerical results of our numerical methods with several examples of. Chapter 6 concludes the thesis.

Chapter 2

Pricing Financial Derivatives

2.1 The Risk Neutral Approach

The Black-Scholes framework is a theoretical valuation formula for options. It reveals the relationship between the prices of the options and the underlying assets. Since almost all corporate liabilities can be viewed as combinations of options, the formula is applicable to common stocks and corporate bonds [3]. The Black-Scholes model makes the following assumptions:

- There does not exist any arbitrage opportunity in the financial market. The traders can't make instantaneous profit without any risk.
- The underlying asset value follows a geometric Brownian Motion $dS = \mu S dt + \sigma S dB$ where μ denotes the average rate of growth of the underlying assets, σ denotes the volatility of the asset price and B is a Brownian Motion.
- The market is frictionless. This means there are no transaction fees, the interest rates for borrowing and lending money from and to the bank are the same, every party in market has immediate information and all entities are available at anytime and in any size.

2.1.1 Black-Scholes Partial Differential Equation

The original model is used to price the vanilla option, which is the simplest type of option. The dividends can be included in the Black-Scholes formula. Presence of dividends can be included in the Black-Scholes formula. Since it doesn't effect the performance, for the sake of simplicity we will assume there are no dividends paid. Under the assumptions of Black-Scholes framework, the call or put option price satisfies the parabolic partial differential equation.

$$\frac{\partial V}{\partial t} = rS\frac{\partial V}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} - rV \tag{2.1}$$

Framework shows the price V(t,S) of a European option driven by one underlying asset that satisfies the PDE where r, σ, t, Srespectively denotes the risk-free interest rate, volatility, time and the underlying price. It is assumed that r and σ are constants. In more complicated models such as stochastic volatility models they can be a function.We will consider the PDE and conditions for the call options. In order to price a vanilla call option the PDE needs to satisfy the following boundary and initial conditions.

$$C(0,t) = 0,\ C(S_{\max},t) = S_{\max} - Ke^{-r(T-t)}, \quad 0 \le t \le T \tag{2.2}$$

$$C(S,T) = \max(S-K,0), \quad 0 \le S \le S_{\max} \tag{2.3}$$

2.1.2 Derivation of the Black-Scholes Equation

Black-Scholes model takes advantage of the properties of the geometric Brownian motion and Itô's lemma.

Definition 2.1.1. Brownian Motion
Brownian motion (also known as Wiener Process) was discovered by botanist Robert Brown as he observed a chaotic motion of particles suspended in water [33]. A Brownian motion, $B(t)$, is a continuous-time stochastic process with

the following properties:

- $B(0) = 0$.
- $B(t)$ is a continuous function of t.
- For $0 \leq s < t$ the increment $B(t) - B(s)$ has normal distribution $\mathcal{N}(0, t - s)$.
- For $t_0 \leq t_1 \leq ... \leq t_n$ the increments $B(t_k) - B(t_{k-1})$ where $k = 1, ..., n$ are independent random variables.

Brownian motion is the basic building block in stochastic calculus and geometric Brownian motion is used to model the stock prices in Black-Scholes model.

Lemma 2.1.2. *Itô's Lemma: Let $B(t)$ be a Brownian motion and $X(t)$ be an Ito process which satisfies the stochastic differential equation:*

$$dX(t) = \mu(X(t), t)dt + \sigma(X(t), t)dB(t) \tag{2.4}$$

If $f(x, t)$ is twice continuously differentiable function then $f(X(t), t)$ is also an Ito drift-diffusion process [16], with its differential given by:

$$d(f(X(t), t)) = \frac{\partial f}{\partial t}(X(t), t)dt + f'(X(t), t)dX + \frac{1}{2}f''(X(t), t)dX(t)^2 \tag{2.5}$$

With $dX(t)^2$ given by: $dt^2 = 0$, $dtdB(t) = 0$ and $dB(t)^2 = dt$.

Theorem 2.1.3. *Assume that the asset price S follows a geometric Brownian motion. Under the assumptions of Black-Scholes framework, the call or put option price $V(t, S)$ satisfies the parabolic partial differential equation*

$$\frac{\partial V}{\partial t} = rS\frac{\partial V}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} - rV \tag{2.6}$$

Proof. Suppose an investor sets up a self-financing portfolio, $X(t)$, comprising one option and an Δ amount of the underlying asset. Therefore, value of the portfolio at time t is $X(t) = V(t) + \Delta S(t)$. Since the self-financing trading strategy has no capital influx or consumption, the value of portfolio change can be written as

$$dX = dV + \Delta dS \tag{2.7}$$

Applying the Itô's Lemma to the option price V(t,S)

$$dV = \frac{\partial V}{\partial t}dt + \frac{\partial V}{\partial S}(S,t)dS + \frac{1}{2}\frac{\partial^2 V}{\partial S^2}(S,t)dS^2 \tag{2.8}$$

Since the Black-Scholes model assumes that the stock price under the "market probability measure" follows a gBM.

$$dS = \mu S dt + \sigma S dW \tag{2.9}$$

Putting (1.4) and (1.6) together yields

$$dV = (\frac{\partial V}{\partial t} + \mu S\frac{\partial V}{\partial S} + \frac{1}{2}\sigma^2 S^2\frac{\partial^2 V}{\partial S^2} + \Delta\mu S)dt + (\sigma S\frac{\partial V}{\partial S} + \Delta\sigma S)dW \tag{2.10}$$

The fact that portfolio is risk-free implies that the second term involving the Brownian Motion, dW, must be zero. This technique is known as delta-hedging , otherwise, we would have an arbitrage opportunity. Thus, $\Delta = -\frac{\partial V}{\partial S}$. Hence, the growth rate of the portfolio must be the risk free rate which can be summarized as $dX = rXdt$. Substituting Δ and dX yields

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2\frac{\partial^2 V}{\partial S^2} = r(V - S\frac{\partial V}{\partial S}) \tag{2.11}$$

Rearranging the equation to get famous Black-Scholes equation:

$$\frac{\partial V}{\partial t} = rS\frac{\partial V}{\partial S} + \frac{1}{2}\sigma^2 S^2\frac{\partial^2 V}{\partial S^2} - rV \tag{2.12}$$

□

Definition 2.1.4. The resulting partial differential equation can be solved analytically using the following boundary conditions and initial conditions for call options.

$$\frac{\partial C}{\partial t} = rS\frac{\partial C}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 C}{\partial S^2} - rC \tag{2.13}$$

$$C(0,t) = 0,\ C(S_{\max},t) = S_{\max} - Ke^{-r(T-t)}, \quad 0 \le t \le T \tag{2.14}$$

$$C(S,T) = \max(S-K,0), \quad 0 \le S \le S_{\max} \tag{2.15}$$

Solving the equations, the formulae [41] for European call is

$$C = S\Phi(d_1) - Ke^{-r(T-t)}\Phi(d_2) \tag{2.16}$$

$$d_1 = \frac{log(S/K) + (r + \sigma^2/2)(T-t)}{\sigma\sqrt{T-t}} \tag{2.17}$$

$$d_2 = d_1 - \sigma\sqrt{T-t} \tag{2.18}$$

The following parameters will be used for the research purposes of this project.

Parameter	Value
Strike Price (K)	1.0
Volatility (σ)	20 %
Risk Free Rate (r)	5 %
Time to Expiry (T)	2.0
Maximum Share Price (S_{max})	2.0

Definition 2.1.5. Black-Scholes PDE has coefficient that depend on S. Meaning that PDE is not space homogeneous. Log spot Black-Scholes PDE is

an economically intrinsic way of looking at numbers, if two assets are similar it is conventional to investigate using the $x = lnS$ conversion.

Applying the chain rule to the first and second order derivatives

$$\frac{\partial C}{\partial S} = \frac{\partial C}{\partial x}\frac{\partial x}{\partial S} = \frac{\partial C}{\partial x}\frac{1}{S} \tag{2.19}$$

$$\frac{\partial^2 C}{\partial S^2} = \frac{\partial}{\partial S}(\frac{\partial C}{\partial S}) = \frac{\partial}{\partial S}(\frac{\partial C}{\partial x}\frac{1}{S}) = -\frac{1}{S^2}\frac{\partial C}{\partial x} + \frac{\partial}{\partial S}\frac{\partial C}{\partial x}\frac{1}{S} = \tag{2.20}$$

$$= -\frac{1}{S^2}\frac{\partial C}{\partial x} + \frac{\partial^2 C}{\partial x^2}\frac{1}{S^2} \tag{2.21}$$

Substituting the transformed derivatives into the original PDE

$$\frac{\partial C}{\partial t} = (r - \frac{\sigma^2}{2})\frac{\partial C}{\partial x} + \frac{1}{2}\sigma^2\frac{\partial^2 C}{\partial x^2} - rC \tag{2.22}$$

The transformation creates a PDE with constant coefficients rather than coefficients that depend on S.

Remark 2.1.6. Untradable Assets
Modern financial engineering created derivatives using untradable assets as an underlying such as multi asset derivatives like equity baskets, weather derivatives, non-deliverable swaps and non-deliverable forwards. Non-deliverable forwards are for offshore investors that want to trade non-convertible currencies such as Brazilian Real, South Korean Won. The Black-Scholes model is still used in these cases [15] [5] but not entirely applicable to assets that cannot be hedged.

2.2 Partial Differential Equations

Since the foundation of the world humanity tried to understand and model the nature. Differential equations serves this purpose by enabling us to describe natural phenomena for instance, heat, sound and fluid flow. Differential equations can be classified in to two categories. Ordinary differential equations serve to model a movement space or plane, an example would be the trajectory of a projectile launched from a cannon follows a curve determined by an ordinary differential equation that is derived from Newton's second law.

On the other hand, partial differential equations modelles a function, a typical example is the heat distribution. This distinction usually makes PDEs much harder to determine an analytical solution than ordinary differential equations. Therefore, we need to achieve a numerical solution to the problem. One of the most commong numerical method for partial differential equations is the finite difference methods. The methods consist of finding approximate solutions to the problem at a discrete set of points, normally on a rectangular grid of points. Finite difference methods are simple to construct and analyse but can compromise performance because of increased computational complexity when there are high dimensions.

Feynman-Kac theorem [16], establishes a link between partial differential equations and stochastic processes by writing the solution as a conditional expectation. Thanks to the theorem, Monte Carlo method is also utilized to find the numerical solutions to the partial differential equations. The convergence rate of Monte Carlo method for n simulations can be denoted as $\mathcal{O}(n^{\frac{-1}{2}})$ which holds for all dimensions (d). The error in d dimensional trapezoidal rule for twice continuously differentiable integrands is $\mathcal{O}(n^{\frac{-2}{d}})$ [12]. Thus, Monte Carlo is a method of choice when evaluating higher dimensions.

2.2.1 Heat Equation

The heat equation is fundamental to financial engineering. Heat equation is a component in the Black-Schole equation and Black-Scholes equation can be transformed to the heat equation by changing variables [42]. Therefore, understanding heat equation is crucial to grasping concepts of partial differential equations. Heat equation will serve as a benchmark with the following initial and boundary conditions.

$$u_t(x,t) = u_{xx}(x,t) \tag{2.23}$$

$$u(0,t) = u(x_{max},t) = 0, \quad 0 \leq t \leq T \tag{2.24}$$

$$u(x,0) = sin(\pi x), \quad 0 \leq x \leq x_{\max} \tag{2.25}$$

In our calculations, we will test the case where $T = 0.06$ and $x_{max} = 1.0$.

Definition 2.2.1. Analytical Solution of Heat Equation

Certain kinds of partial differential equations allows us to find an analytical solution with the help of the Separation of Variables technique.

$$u(x,t) = X(x)T(t) \tag{2.26}$$

$$u_{xx}(x,t) = X^{''}(x)T(t) \tag{2.27}$$

$$u_t(x,t) = X(t)T^{'}(t) \tag{2.28}$$

Using the partial derivatives the equation $u_t = u_{xx}$ becomes

$$\frac{T^{'}(t)}{T(t)} = \frac{X^{''}(x)}{X(x)} \tag{2.29}$$

Right hand side only depends on x and the left hand side depends only on t. Therefore, the equation is valid only when each side is equal to a constant, which we set to λ. Rearranging terms gives us the following equations:

$$\frac{T^{'}(t)}{T(t)} = \frac{X^{''}(x)}{X(x)} = -\lambda \tag{2.30}$$

$$X^{''}(x) + \lambda X(x) = 0 \tag{2.31}$$

$$T^{'}(t) + \lambda T(t) = 0 \tag{2.32}$$

$$X(0) = X(1) = 0 \tag{2.33}$$

Solving for X(x) is an example case of Sturm-Liouville problem [18] with three cases.

- Let $\lambda < 0$ and $\lambda = -k^2$. Then the solution to 2.31 is

  $$X = Ae^{kx} + Be^{-kx}$$

  Using the boundary conditions yield $X(0) = A + B = 0$ and $X(1) = Ae^{k} + Be^{-k} = 0$. Solving the equations $A = B = u = 0$ which is a trivial solution, thus discarded.

- Let $\lambda = 0$, the solution to 2.31 is

  $$X(x) = Ax + B$$

  The boundary conditions imply $X(0) = B = 0$ and $X(1) = A = 0$. Thus this case is discarded too.

- Finally, let $\lambda > 0$,the solution to 2.31 is

  $$X(x) = Acos(\sqrt{\lambda}x) + Bsin(\sqrt{\lambda}x)$$

  The boundary conditions leads to $X(0) = A = 0$ and $X(1) = Bsin(\sqrt{\lambda}) = 0$ Since we do not want a trivial solution where $B = 0$, the equation reduces to

  $$sin(\sqrt{\lambda}) = 0 \tag{2.34}$$

Thus $\sqrt{\lambda} = n\pi$ for $n = 1, 2, 3, ...$ Solution to 2.31 becomes,

$$X_n = b_n sin(n\pi x), \quad n = 1, 2, 3, ... \tag{2.35}$$

As we determined $\lambda = n^2\pi^2$ for $n = 1, 2, 3, ...$ Solving 2.32 for $T(t)$ gives the solution

$$T^{'}(t) = -n^2\pi^2 T(t) T_n = c_n exp(-n^2\pi^2 t) \tag{2.36}$$

$$\tag{2.37}$$

where c_n's are integration constants.

Putting the solution of $T(t)$ and $X(x)$ together,

$$u(x,t) = \sum_{n=1}^{\infty} B_n exp(-n^2\pi^2 t) sin(n\pi x) \tag{2.38}$$

where we have set $B_n = c_n b_n$. The initial condition gives

$$u(x,0) = sin(\pi x) = \sum_{n=1}^{\infty} B_n sin(n\pi x) \tag{2.39}$$

which is a Fourier sine series. Solving for the B_n's, we use the orthogonality property for the eigenfunctions $sin(n\pi x)$

$$\int_0^1 sin(m\pi x) sin(\pi n x) dx = \begin{cases} 0, & \text{if } m \neq n \\ 1/2, & \text{m = n} \end{cases} = 0.5\delta_{mn}$$

where δ_{mn} is the kronecker delta,

$$\delta_{mn} = \begin{cases} 0, & \text{if } m \neq n \\ 1, & \text{m = n} \end{cases}$$

Solving 2.39 for B_n, multiplying both sides with $sin(m\pi x)$ and integrate from 0 to 1 and from the definition of kronecker delta yields

$$B_n = 2\int_0^1 sin(\pi x)sin(\pi nx)dx = \frac{2sin(\pi n)}{\pi - \pi n^2} \tag{2.40}$$

Combining the solutions

$$u(x,t) = \sum_{n=1}^{\infty} \frac{2sin(\pi n)}{\pi - \pi n^2} exp(-n^2\pi^2 t)sin(n\pi x) = exp(-\pi^2 t)sin(\pi x) \tag{2.41}$$

2.2.2 Two Dimensional Heat Equation

The natural extension of our study of the one-dimensional problem would now be to investigate partial differential equations with more than one space-like dimension. When more than one space dimensions are involved, we have to deal with equations such as two dimensional heat equation or multi-asset black-scholes equation. We will consider the following PDE and conditions for the purposes of research.

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \tag{2.42}$$

Initial and boundary condition

$$u(x,y,0) = 1, \quad 0 \leq x \leq x_{\max}, \quad 0 \leq y \leq y_{\max} \tag{2.43}$$

$$u(x,0,t) = u(x,y_{max},t) = 0, \quad 0 \leq t \leq T \tag{2.44}$$

$$u(x,0,t) = u(x,1,t) = 0, \quad 0 \leq t \leq T \tag{2.45}$$

In the calculations, we will test the case where $T = 0.06$, $x_{max} = 1.0$ and $y_{max} = 1.0$.

Definition 2.2.2. Analytical Solution of Two Dimensional Heat Equation

Similarly, applying separation of variables method to the equation

$$u(x,t) = X(x)Y(y)T(t) \tag{2.46}$$

$$X^{''}(x) - BX(x) = 0 \tag{2.47}$$

$$Y^{''}(y) - C(y) = 0 \tag{2.48}$$

$$T^{'}(t) - (B+C)T(t) = 0 \tag{2.49}$$

$$X(0) = X(1) = 0, \quad Y(0) = Y(1) = 0 \tag{2.50}$$

In 2.2.1, we have already seen that the solutions to X(x) and Y(y) are

$$X_m(x) = b_n sin(m\pi x) \tag{2.51}$$

$$Y_n(x) = a_m sin(n\pi y) \tag{2.52}$$

Using these values to solve for $T(t)$ gives

$$T_{mn}(t) = c_{mn} exp(-\pi^2(m^2+n^2)t) \tag{2.53}$$

Substituting the solutions yields

$$u_{mn}(x,y,t) = X_m(x)Y_n(y)T_{mn}(t) \tag{2.54}$$

$$u(x,y,t) = \sum_{m=1}^{\infty}\sum_{n=1}^{\infty} A_{mn} sin(m\pi x) sin(n\pi y) exp(-\pi^2(m^2+n^2)t) \tag{2.55}$$

where $A_{mn} = b_n a_m c_{mn}$. The initial condition gives

$$u(x,y,0) = 1 = \sum_{m=1}^{\infty}\sum_{n=1}^{\infty} A_{mn} sin(m\pi x) sin(n\pi y) \tag{2.56}$$

which is a double Fourier sine series. Thus, the coefficient A_{mn} is chosen such that

$$A_{mn} = 4\int_0^1\int_0^1 sin(\pi mx)sin(\pi ny)dxdy = \frac{4(cos(\pi n)-1)(cos(\pi m)-1)}{\pi^2 mn} \tag{2.57}$$

Combining the solutions

$$u_{mn}(x,y,t) = \sum_{m=1}^{\infty}\sum_{n=1}^{\infty}\frac{4(cos(\pi n)-1)(cos(\pi m)-1)}{\pi^2 mn} sin(m\pi x)sin(n\pi y)exp(-\pi^2(m^2+n^2)t) = 0 \tag{2.58}$$

2.3 Finite Difference Methods

2.3.1 Discretization

Essentially, solving a PDE is the problem of finding a function which depends on values at infinitely many points. Naturally, the finite difference methods first step is to make the problem discrete that we are able to solve [38]. As a result, we need to discretise the space dimensions and time dimension. The discretization procedure begins by replacing the domain $[0, x_{max}]$ x $[0, T]$ by a set of mesh points. In order to get a n x m equally spaced mesh points the step sizes are calculated as $\Delta t = \frac{T}{m}$, $\Delta x = \frac{x_{max}}{n}$.

In order to replace our PDE, we need to utilize finite difference approximations for the partial derivatives. Notationally, we will define u_i^n to be a function defined at the point $(i\Delta x, n\Delta t)$.

- Forward difference: $\frac{\partial u}{\partial t} = \frac{u_i^{n+1}-u_i^n}{\Delta t} + \mathcal{O}(\Delta t)$
- Central difference: $\frac{\partial u}{\partial x} = \frac{u_{i+1}^n-u_{i-1}^n}{\Delta t} + \mathcal{O}(\Delta x)$
- Backwards difference: $\frac{\partial u}{\partial x} = \frac{u_i^n-u_{i-1}^n}{\Delta t} + \mathcal{O}(\Delta x)$
- Second order central difference: $\frac{\partial^2 u}{\partial x^2} = \frac{u_{i+1}^n-2u_i^n+u_{i-1}^n}{(\Delta x)^2} + \mathcal{O}(\Delta x^2)$

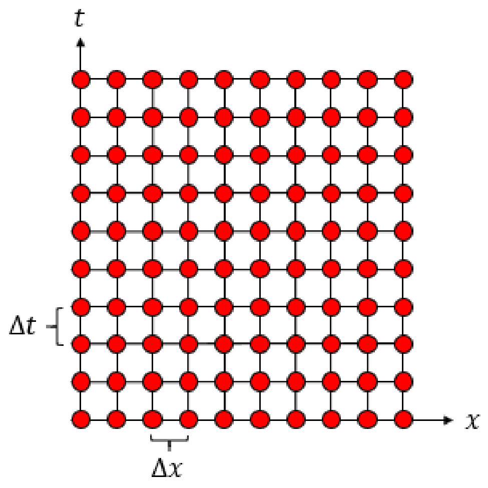

Figure 2.1: 10 x 10 grid.

We now have a grid that approximates our domain. Aiming to obtain a unique solution using numerical methods, we need initial and boundary conditions. Final step is applying the values given by such conditions.

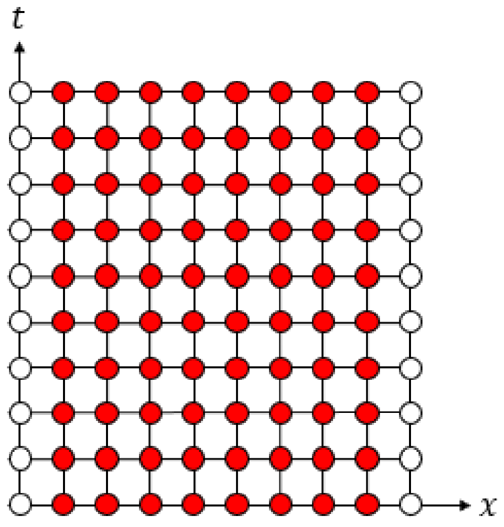

Figure 2.2: Boundary conditions.

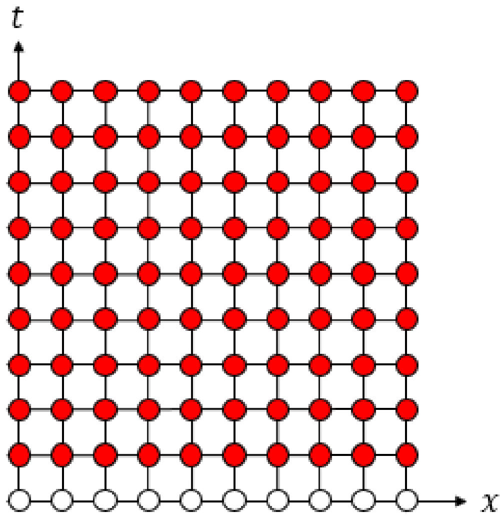

Figure 2.3: Initial condition.

2.3.2 Explicit Method

Explicit method generalises the parabolic partial differential equation by applying the forward difference to the time derivative and the centred second difference (FTCS scheme).

$$u_t = a(t,x)u_{xx} + b(t,x)u_x + c(t,x)u \tag{2.59}$$

We will be applying the finite differences to the equation 2.59 for the purposes of simplicity since heat equation and Black-Scholes equation can be generalized in the form for certain choices of coefficients. Applying the forward time and centred space differences where $r = \frac{\Delta t}{\Delta x^2}$.

$$\begin{aligned}\frac{u_i^{n+1} - u_i^n}{\Delta t} &= a(t,x)\frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{(\Delta x)^2} + b(t,x)\frac{u_{i+1}^n - u_{i-1}^n}{\Delta x} + +c(t,x)u_i^n \\ u_i^{n+1} &= u_{i+1}^n(\frac{-rb(t,x)}{\Delta x} - ra(t,x)) + u_i^n(1 + 2ra(t,x) - c(t,x)\Delta t) \\ &\qquad + u_{i-1}^n(-ra(t,x) + \frac{rb(t,x)}{\Delta x}) \end{aligned} \tag{2.60}$$

We will replace the coefficients of $u_{i+1}^n, u_i^n, u_{i-1}^n$ terms with γ, α, β respectively. The formula reduces to

$$u_j^{n+1} = \gamma u_{j+1}^n + \beta u_j^n + \alpha u_{j-1}^n. \tag{2.61}$$

The formula expresses one unknown nodal value directly in terms of known nodal values [10]. It can be expanded as

$$
\begin{aligned}
u_1^{n+1} &= \gamma u_2^n + \beta u_1^n + \alpha u_0^n \\
u_2^{n+1} &= \gamma u_3^n + \beta u_2^n + \alpha u_1^n \\
&. \\
&. \\
&. \\
u_{j-1}^{n+1} &= \gamma u_j^n + \beta u_{j-1}^n + \alpha u_{j-2}^n
\end{aligned} \tag{2.62}
$$

Using the boundary conditions and initial condition, the expanded formula can be condensed in the following matrix form.

$$
\begin{bmatrix} u_1^{n+1} \\ u_2^{n+1} \\ u_3^{n+1} \\ . \\ . \\ . \\ u_{j-1}^{n+1} \end{bmatrix}
=
\begin{bmatrix} \alpha u_0^n \\ 0 \\ 0 \\ . \\ . \\ . \\ \gamma u_j^n \end{bmatrix}
+
\begin{bmatrix} \beta & \gamma & 0 & . & . & 0 \\ \alpha & \beta & \gamma & 0 & ... & . \\ 0 & \alpha & \beta & \gamma & 0 & . \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ . & . & . & . & . & \gamma \\ 0 & 0 & 0 & 0 & \alpha & \beta \end{bmatrix}
\begin{bmatrix} u_1^n \\ u_2^n \\ u_3^n \\ . \\ . \\ . \\ u_{j-1}^n \end{bmatrix}
$$

Deriving the coefficients γ, α, β in the case of heat equation yields

$$
\alpha = r \quad \beta = 1 - 2r \quad \gamma = r \tag{2.63}
$$

In the case of Black-Scholes formula, since the share price S_j increases linearly with Δx we can replace it as $S_j = j\Delta x$.

$$
\alpha = \frac{\sigma^2 j^2 \Delta t}{2} - \frac{rj\Delta t}{2} \quad \beta = 1 - \sigma^2 j^2 \Delta t - r\Delta t \quad \gamma = \frac{\sigma^2 j^2 \Delta t}{2} + \frac{rj\Delta t}{2} \tag{2.64}
$$

Lastly, solving heat equation and Black-Scholes differs in time stepping. Black-Scholes formula is solved backwards in time. On the other hand, heat equation is solved forwards in time.

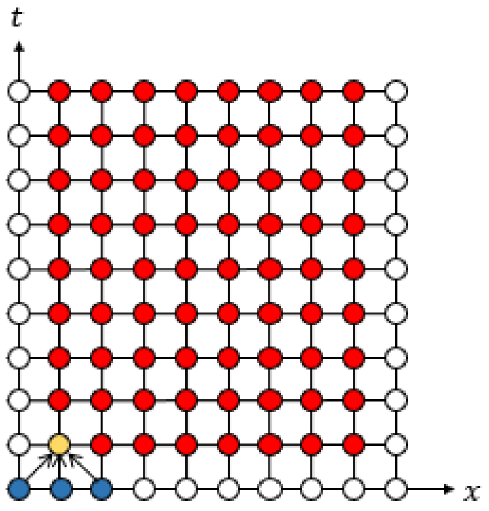

Figure 2.4: Computational stencil of heat equation.

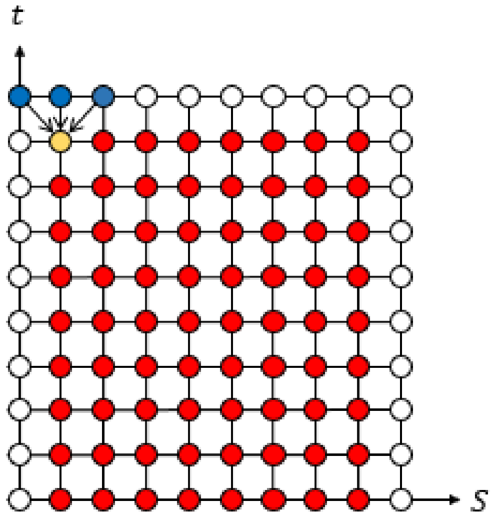

Figure 2.5: Computational stencil of Black-Scholes.

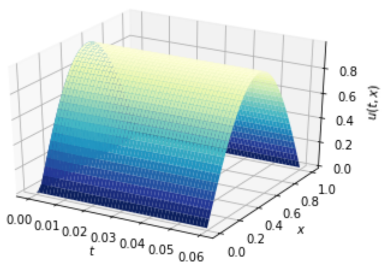

Figure 2.6: Output grid of heat equation.

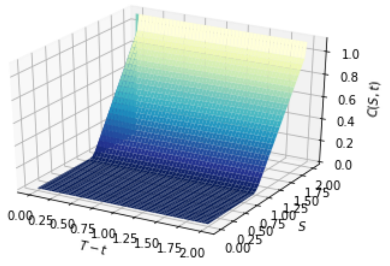

Figure 2.7: Output grid of Black-Scholes equation.

2.3.3 Crank-Nicholson Method

The explicit method is computationally cheap. However, this brings a serious drawback, for explicit method to attain reasonable accuracy the step size must be kept small [34]. Thankfully, the Crank-Nicolson finite difference scheme was introduced by John Crank and Phyllis Nicolson [7]. Considering numerous articles and publications in the financial engineering literature use Crank-Nicolson as the de-facto scheme for time discretisation, the method has become one of the most popular finite difference schemes for approximating the solution of the Black - Scholes equation and its generalisations [35].

If we apply backwards time difference instead of forward time difference that Explicit method used and a central space approximation in space again, we get the BTCS scheme. Applying the BTCS to the base equation 2.59. yields

$$\frac{u_i^{n+1}-u_i^n}{\Delta t}=a(t,x)\frac{u_{i+1}^{n+1}-2u_i^{n+1}+u_{i-1}^{n+1}}{(\Delta x)^2}+b(t,x)\frac{u_i^{n+1}-u_i^{n+1}}{\Delta t}+c(t,x)u_i^{n+1} \tag{2.65}$$

Crank-Nicolson method takes a weighted average of the FTCS and BTCS schemes. Therefore the approximations become

$$u(t,x)\approx\frac{1}{2}(u_i^{n+1}+u_i^n)$$

$$\frac{\partial u}{\partial t}\approx\frac{u_i^{n+1}-u_i^n}{\Delta t}$$

$$\frac{\partial u}{\partial x}\approx\frac{u_{i+1}^n-u_{i-1}^n+u_{i+1}^{n+1}-u_{i-1}^{n+1}}{4\Delta x}$$

$$\frac{\partial^2 u}{\partial x^2}\approx\frac{u_{i+1}^n-2u_i^n+u_{i-1}^n+u_{i+1}^{n+1}-2u_i^{n+1}+u_{i-1}^{n+1}}{2(\Delta x)^2}$$

Applying the new finite differences to the base partial differential equation equation 2.59

$$\begin{aligned}(-A-B)u_{i+1}^{n+1}+(1+2A-C)u_i^{n+1}+(-A+B)u_{i-1}^{n+1}&=\\ =(A+B)u_{i+1}^n+(1-2A+C)u_i^n+(A-B)u_{i-1}^n\end{aligned} \tag{2.66}$$

$$A = a(t,x)\frac{\Delta t}{\Delta x^2}, B = b(t,x)\frac{\Delta t}{4\Delta x}, C = c(t,x)\frac{\Delta t}{2}$$

Note that in contrast to the FTCS scheme, we now have three unknowns in this equation, the three values of u at the higher time level. We respectively denote the coefficients in the right hand side as γ, β, α and coefficients in the left hand side as λ, θ, ω for simplicity.

$$\lambda u_{i+1}^{n+1} + \theta u_i^{n+1} + \omega u_{i-1}^{n+1} = \; = \gamma u_{i+1}^n + \beta u_i^n + \alpha u_{i-1}^n \tag{2.67}$$

The left hand side groups the unknowns and the right hand side groups knowns. The system of equations can be reduced to a matrix system.

$$\begin{bmatrix} \theta & \lambda & 0 & . & . & 0 \\ \omega & \theta & \lambda & 0 & ... & . \\ 0 & \omega & \theta & \lambda & 0 & . \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ . & . & . & . & . & \lambda \\ 0 & 0 & 0 & 0 & \omega & \theta \end{bmatrix} \begin{bmatrix} u_1^{n+1} \\ u_2^{n+1} \\ u_3^{n+1} \\ . \\ . \\ . \\ u_{j-1}^{n+1} \end{bmatrix} = \begin{bmatrix} \alpha u_0^n \\ 0 \\ 0 \\ . \\ . \\ . \\ \gamma u_j^n \end{bmatrix} + \begin{bmatrix} \beta & \gamma & 0 & . & . & 0 \\ \alpha & \beta & \gamma & 0 & ... & . \\ 0 & \alpha & \beta & \gamma & 0 & . \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ . & . & . & . & . & \gamma \\ 0 & 0 & 0 & 0 & \alpha & \beta \end{bmatrix} \begin{bmatrix} u_1^n \\ u_2^n \\ u_3^n \\ . \\ . \\ . \\ u_{j-1}^n \end{bmatrix}$$

The problem reduces to a tridiagonal matrix system. This system of equations can be solved by various algorithms such as Gaussian elimination or Thomas algorithm.

2.3.4 Alternating Direction Implicit Method

The alternating direction implicit (ADI) method is one of the most common techniques to numerically solve two dimensional parabolic PDEs. ADI scheme give us advantages of the implicit finite difference method and computationally requires to solve only tridiagonal matrices. The scheme was first proposed by Peaceman and Rachford in 1955 for oil reservoir modelling [31].

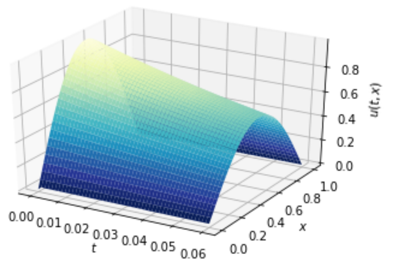

Figure 2.8: Output grid of heat equation.

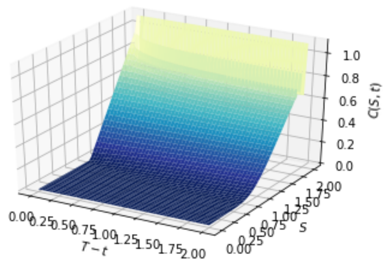

Figure 2.9: Output grid of Black-Scholes equation.

Basically the methods to split the spatial dimensions and solve a two dimensional problem as two consecutive one dimensional problems. It is possible to use ADI in more than three dimensions which produces the same number of consecutive one dimensional problems [8]. In order to develop a more compact notation, we introduce the finite difference operator notation δ^2.

$$\delta x^2 u^n_{i,j} = \frac{u^n_{i+1,j} - 2u^n_{i,j} + u^n_{i-1,j}}{\Delta x^2} \tag{2.68}$$

Explicit method in two space dimensions can be abbreviated as

$$\frac{u^{n+1}_{i,j} + u^n_{i,j}}{\Delta t} = \delta x^2 u^n_{i,j} + \delta y^2 u^n_{i,j} \tag{2.69}$$

and implicit method in two space dimensions can be written as

$$\frac{u^{n+1}_{i,j} + u^n_{i,j}}{\Delta t} = \delta x^2 u^{n+1}_{i,j} + \delta y^2 u^{n+1}_{i,j}. \tag{2.70}$$

Dividing each time step in half we introduce a temporary intermediate unknown $u^{n+1/2}_{i,j}$. Firstly, the two dimensional heat equation is approximating implicitly x and explicitly over y. The total work involved in one time step amounts to solving $N_{steps} - 1$ tridiagonal systems [28].

$$\frac{u_{i,j}^{n+1/2} + u_{i,j}^{n}}{0.5\Delta t} = \frac{\delta x^2 u_{i,j}^{n+1/2}}{\Delta x^2} + \frac{\delta y^2 u_{i,j}^{n}}{\Delta y^2} \tag{2.71}$$

Rearranging the set of equations yields a tridiagonal system which is solved for the temporary intermediate unknown $u_{i,j}^{n+1/2}$.

$$-r_x u_{i+1,j}^{n+1/2} + (1+2r_x) u_{i,j}^{n+1/2} - r_x u_{i,j}^{n+1/2} = r_y u_{i,j+1}^{n} + (1+2r_y) u_{i,j}^{n} + r_y u_{i,j-1}^{n} \tag{2.72}$$

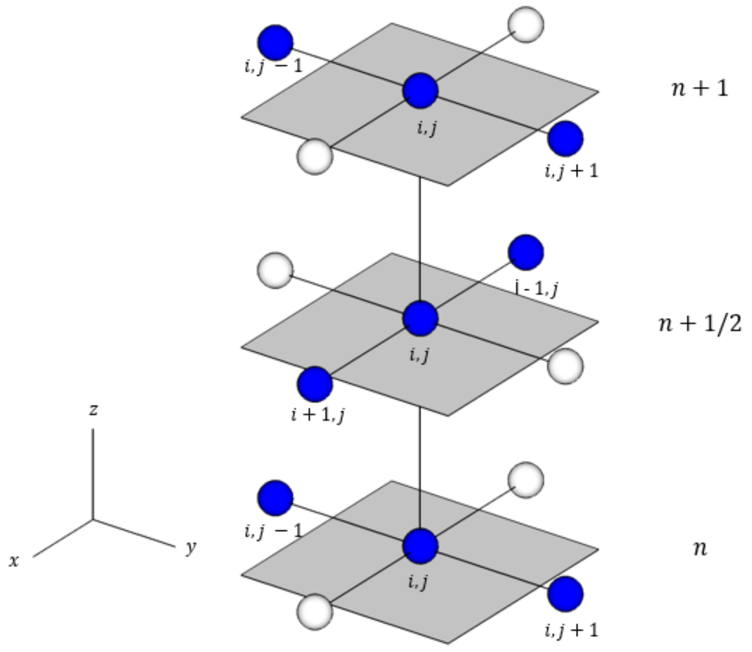

Figure 2.10: Computational stencil of alternating direction implicit method

Next step of the grid $u_{i,j}^{n+1}$ is calculated by approximating explicitly x and implicitly over y.

$$\frac{u_{i,j}^{n+1} + u_{i,j}^{n+1/2}}{0.5\Delta t} = \frac{\delta x^2 u_{i,j}^{n+1/2}}{\Delta x^2} + \frac{\delta y^2 u_{i,j}^{n+1}}{\Delta y^2} \tag{2.73}$$

Rearranging the set of equations yields a tridiagonal system which can be solved using Gaussian elimination, cyclic reduction or Thomas algorithm.

$$-r_y u_{i,j+1}^{n+1} + (1+2r_y)u_{i,j}^{n+1} - r_y u_{i,j-1}^{n+1} = r_x * u_{i+1,j}^{n+1/2} + (1+2r_x)u_{i,j}^{n+1/2} + r_x u_{i,j}^{n+1/2} \tag{2.74}$$

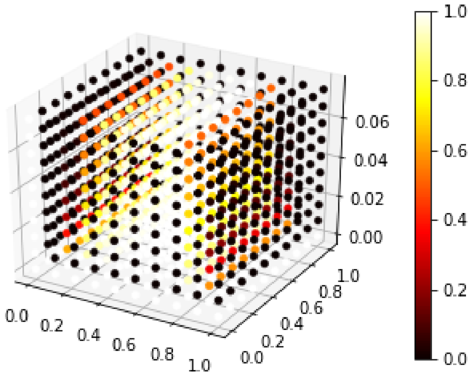

Figure 2.11: Solution of two dimensional heat equation using ADI.

Chapter 3

Optimizing Solvers

Attempting to progress in solving complex problems using numerical methods is impossible without applying optimizations. However, nowadays the problem is not to solve a given problem but rather solve it in a given computing environment while exploiting the resources in an optimal way. Thus, it is necessary to investigate methods that allow for efficient implementations. The aim of this section is to introduce practical optimization techniques that can be easily implemented on a regular trading floor. Main optimization techniques that will be tested are parallelizing tridiagonal solvers, Visual Studio optimization switches, compilers and solution platforms.

3.1 Solution Platforms

The CPU accesses data from RAM using the register that stores memory addresses. 32 bit and 64 bit refers to the amount of data the system can access. so a 32-bit system can address a maximum of 4 GB (4,294,967,296 bytes) of RAM where a 64-bit register can theoretically reference 18,446,744,073,709,551,616 bytes, or 17,179,869,184 GB (16 exabytes) of memory. Since 32 bit does not have access to more than 4 GB, if the system has more than 4 GB of RAM, it will be inaccesible by the CPU, thus A 64 bit system will be needed.

The memory increase of 64 bit systems means it is capable of very fast processing of numerical quantities. One disadvantage of the 64 bit systems is more requirement of memory because addresses are 64 bits (8 bytes) wide instead of 32 bits (4 bytes) wide. Due to the increased size of pointers and data structures, 64-bit programs will occupy more memory than an 32-bit version.

3.2 Compilers

A good compiler should let us focus on the process of writing programs rather than struggling with the inadequacies of the compiler. It should compile the most recent language standards without complaint. We should feel confident in its ability to produce well-optimized code from even the most abstract codebase. Lastly, it should compile source code as quickly as possible.

Regardless of the implementation of the program, the performance can be different under varying compilers.

Modern CPUs are highly pipelined, superscalar machines that execute instructions out-of-order, use speculative execution, prefetching, and other performance-enhancing techniques. This makes it difficult to predict exactly how well a sequence of instructions will execute on any given microarchitecture. Hardware engineers use highly sophisticated simulations to overcome these problems when designing new CPUs. Compilers have to use heuristics to decide how to target specific CPU microarchitectures and thus have to be tuned to produce good code. Furthermore, the tuning is workload-specific, i.e., a generally sub-par compiler may produce the best code for certain workloads, even though it generally produces poorer code on average.

3.3 Visual Studio Optimization Switches

Visual Studio Optimization Switches, also known as /O options controls various optimizations to be chosen according to the needs of the project. There are various switches for different goals such as minimizing the size of the code (/O1) but since the scope of this project is limited with speed optimizations. Speed optimization flags are /O2 and /Ox. /O2 is a combination of /Og, /Oi, /Ot, /Oy, /Ob2, /GF and /Gy flags. /Ox is a subset of /O2 without the /GF and /Gy flags. These additional options applied by /O2 can cause pointers to strings or to functions to share a target address, which can affect debugging and strict language conformance [25].

- /Og: Enables local and global optimizations (subexpression elimination), automatic-register allocation, and loop optimization [26].
- /Oi: Generates intrinsic functions for appropriate function calls. Compiler may not replace the function call with an intrinsic if it will result in better performance [23].
- /Ot: Favors optimizations for speed over optimizations for size by instructing the compiler to reduce many C and C++ constructs to functionally similar sequences of machine code. If /Ot is used, /Og must be specified to optimize the code [24].
- /Oy: Suppresses the creation of frame (base) pointers on the call stack for quicker function calls. Frees one register for general usage [27].
- /Ob2: Controls inline expansion of functions. Under /O2 and /Ox, allows the compiler to expand any function including the ones that are not explicitly marked for no inlining. Function-calling-overheads are saved thus inline functions run faster than the normal functions with a memory penalty [22].

- /GF: Enables the compiler to create a single copy of identical strings in the program image and in memory during execution. This is an optimization called string pooling that can create smaller programs. Under this flag, strings are pooled as read-only, trying to modify strings throws an error [20].
- /Gy: Enables function-level linking. Allows the compiler to package individual functions in the form of packaged functions (COMDATs) or order individual functions in a DLL or .exe file [21].

/O2 and /Ox flags are tested for maximum speed against the /Od flag which disables all the optimizations.

3.4 Tridiagonal Solvers

Tridiagonal solvers are the most demanding part of the solvers. Hence, development and improvement of such solvers is of great interest [36] [6] [29] [2] concerned with this problem.

Large tridiagonal systems of linear equations appear in many numerical analysis applications. In our work, they arise in

Implementing Crank Nicolson and Alternating Direction Implicit methods requires to solve tridiagonal systems which is the most computationally intensive part of the program. Therefore, choosing efficient tridiagonal solvers is crucial for the speed of the solver. In this experiment three different algorithms will be tested.

Thomas algorithm which takes $O(N)$ steps and cyclic reduction which takes $2log_2N$ steps

The Thomas algorithm is avery efficient algorithm for solving tridiagonal systems of equations in serial [15].It is equivalent to Gaussian elimination without pivoting; [9]

In this book we are mainly interested in solving tridiagonal systems of equations that arisewhen we discretise PDEs and two-point boundary value problems. In particular, these systemsarise when we approximate partial derivatives in the space dimension by three-point divideddifferences. In most cases we can use the popular Thomas algorithm to solve the resultinglinear system, although the Double Sweep method performs better. In particular, the Thomasalgorithm is used in the Alternating Direction Implicit (ADI) method which is one of the pop-ular schemes in computational finance

3.4.1 Thomas Algorithm

Thomas Algorithm is the most commonly used method for solving tridiagonal system of equations. The method is used to solve a tridiagonal matrix system invented by Llewellyn Thomas [39]. The algorithm is a simplified version of the gaussian elimination.

The system equations can be written as

$$\begin{bmatrix} b_1 & c_1 & 0 & 0 & ... & 0 \\ a_2 & b_2 & c_2 & 0 & ... & 0 \\ 0 & a_3 & b_3 & c_3 & 0 & 0 \\ . & . & & & & . \\ . & . & & & & . \\ . & . & & & & c_{k-1} \\ 0 & 0 & 0 & 0 & a_k & b_k \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ . \\ . \\ . \\ f_k \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ . \\ . \\ . \\ d_k \end{bmatrix}$$

The method begins by forming coefficients c_i^* and d_i^* in place of a_i, b_i and c_i as follows:

$$c_i^* = \begin{cases} \frac{c_1}{b_1} & ; i = 1 \\ \frac{c_i}{b_i - c_{i-1}^* a_i} & ; i = 2, 3, ..., k-1 \end{cases}$$

$$d_i^* = \begin{cases} \frac{d_1}{b_1} & ; i = 1 \\ \frac{d_i - d_{i-1}^* a_i}{b_i - c_{i-1}^* a_i} & ; i = 2, 3, ..., k-1 \end{cases}$$

With these new coefficients, the matrix equation can be rewritten as:

$$\begin{bmatrix} 1 & c_1^* & 0 & 0 & ... & 0 \\ 0 & 1 & c_2^* & 0 & ... & 0 \\ 0 & 0 & 1 & c_3^* & 0 & 0 \\ . & . & & & & . \\ . & . & & & & . \\ . & . & & & & c_{k-1}^* \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ . \\ . \\ . \\ f_k \end{bmatrix} = \begin{bmatrix} d_1^* \\ d_2^* \\ d_3^* \\ . \\ . \\ . \\ d_k^* \end{bmatrix}$$

The algorithm for the solution of these equations is now straightforward and works 'in reverse':

$$f_k = d_k^*, \qquad f_i = d_k^* - c_i^* x_{i+1}, \qquad i = k-1, k-2, ..., 2, 1$$

3.4.2 Intel Math Kernel Library

Intel Math Kernel Library implements routines for solving systems of linear equations from the standard LAPACK library. Variety of atrix types are supported by the routines. Specifically gtsv function is utilized from the package. Using Gaussian elimination with partial pivoting, gstv computes the solution to the system of linear equations with a tridiagonal coefficient matrix [14].

3.4.3 Cyclic Reduction

Cyclic reduction was proposed by R. W. Hockney in the 1960s for solving the resulting linear systems from the discretization of the Poisson equation [13]. Both cyclic reductionand recursive doubling are designed for fine grained parallelism, where each process or owns exactly one row of the tridiagonal matrix.

If we number equations succesively, then in cyclic reduction the odd-neighbor equations of an even equation are used to eliminate the off-diagonal entries in the even equation.In this step, a tridiagonal system of linear equations for the even equation is gener-ated. This reduced system has only about one-half as many equations as the originalsystem. Now the same procedure is applied recursively to the reduced system untilthere remains only one linear equation with one unknown. The solution of successivereduced systems can be computed to finally yield the solution of the original system.Cyclic reduction requires $2log2NPsteps$ Cyclic reduction takes more steps so it requires parallelization to be effective CPU parallelization will be used to make cyclic reduction work.

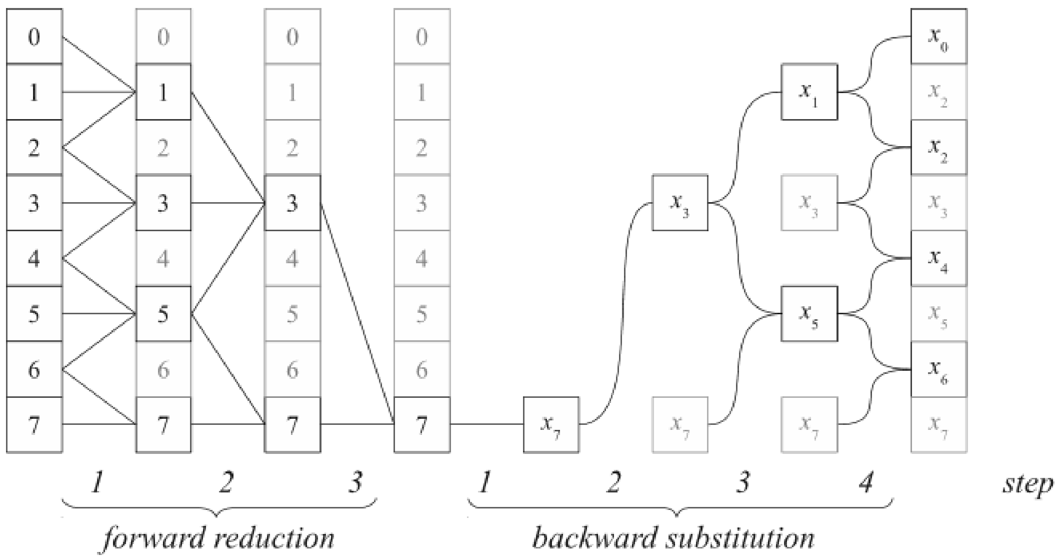

Figure 3.1: Cyclic reduction for an eight equation system.

3.5 Open Multi-Processing

Traditionally, programs are serially computed on a single processor. On the other hand, parallel computation is used to break our code execution in pieces so that it utilizes parallelism. Multithreading uses the CPUs cores to run calculations concurrently in each core. The concurrent programs are called a thread. If the code is executed on parallel processors, one of the biggest problem is the processors generally require results that have been calculated on other processors. The main issue in this case is that processors clocks are not synchronized and execute the code at minimally different speeds.

In order to solve this problem, a group of major computer hardware and software vendors and major parallel computing user facilities joined forces to form The Open Multi-Processing Architecture Review Board (The OpenMP ARB)[37]. Open Multi-Processing (OpenMP) is an implementation of multithreading for C, C++ and Fortran and it was introduced to public in 1997. OpenMP is aiming to standardize high level parallelism that is performant, productive and portable.

OpenMP approach to multithreading is the fork-join programming model. Firstly, the program start as a single thread of execution called the initial thread. The fork stage begins when the program encounters an OpenMP parallel construct. Parallel execution takes place and multiple threads are created in the parallel region. The initial thread becomes the master and collaborates with the newly created threads to execute the code dynamically. Finally, at the join stage all threads are synchronized, threads are terminated except the original thread [4].

Algorithms like cyclic reduction 3.3.3 uses nested loops to calculate the solution. Therefore nested parallelism is needed to build efficient programs. Nesting the parallel constructs results in nested parallelism. Each thread that encounters the next parallel region creates a new parallel region at runtime [40].

In order to specify and control the paralelization procedure, OpenMP uses

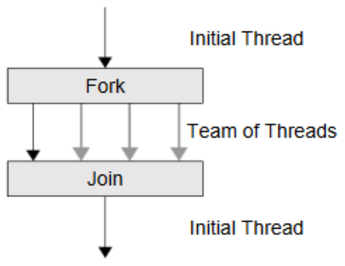

Figure 3.2: The fork-join programming model.

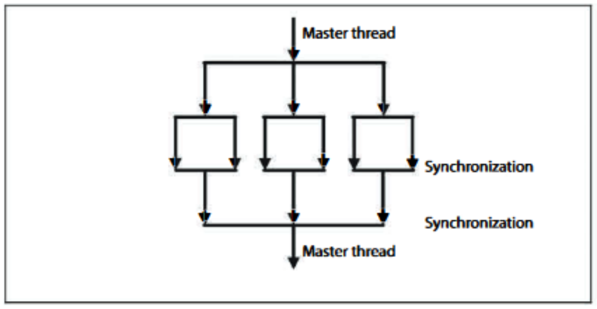

Figure 3.3: 3 threads create 2 threads, nested parallelism.

compiler directives, runtime functions, and environment variables. OpenMP enables developers to just give a high-level specification of the parallelism by indicating the regions to be executed in parallel using compiler directives, runtime library routines, and environment variables. The details of the parallelism is up to the compiler which makes OpenMP comparatively easy to use.

3.6 Timing the Code

Measuring execution time intervals accurately is an important aspect to compare the efficiency and speed of different environments and implementations.

3.6.1 Windows Application Programming Interface

Windows Application Programming Interface (API) is the lowest level of interaction between applications and the windows operating system. Thus every program is built upon the API. Mostly, the interaction is hidden, the runtime and support libraries manage it in the background [32]. The APIs can be used in the C++ environment. Runtime can be calculated by "QueryPerformanceCounter" or "QueryPerformanceFrequency" functions. Respectively, the functions retrieve a high resolution time stamp and the frequency of the performance counter.

3.6.2 Chrono Library

Using the Windows API for just timing the code is slightly excessive given the amount of work it takes. Luckily, Chrono library was introduced part of the C++11's standard library. Timers and clocks might differ on distinct systems, thus Chrono library is designed to work effortlessly with date and time. The "high resolution clock" provides the smallest possible tick period and with the "now" method, returns a value corresponding to the call's point in time. Once the start and end time of the code is recorded , the duration::count method is used to get the elapsed time.

3.7 Comparison of Optimizations

This section documents the performance of attempted optimizations. Experiments are conducted at W307 computer laboratory, Queen Mary University of London. Each computer has Windows 10 Enterprise 64 bit, 16 GB of RAM, Intel Core i7-6700 CPU with 4 cores clocked at 3.40 GHz. The source code is written in C++ and compiled with Microsoft Visual Studio Enterprise 2017, Version 15.3.3 in the release mode. External tools utilized in the tests include Intel Compiler, version 18.0.3 and Intel Math Kernel Library.

Numerical analysis and computer simulations is undertaken to put theory and observation together to gain insight into the workings of numerical solutions of partial differential equations. First step was solving the base cases 2.1.4, 2.23, 2.43 by hand and Excel. Following the simple implementations, the solvers are ported to C++ to measure and optimize the performance. Different solution platforms, compilers, optimization flags and tridiagonal solvers will be tested against each other.

3.7.1 Optimal Grid Size

Previously defined analytical solutions 2.1.4, 2.2.1, 2.2.2 are utilised to calculate errors for the solutions using different grid sizes, the grid with the lowest error is used for further tests. Figure X and X demonstrates the error compared to the grid sizes. The optimal grid size for heat equation and Black-Scholes equation is 50 by 50.

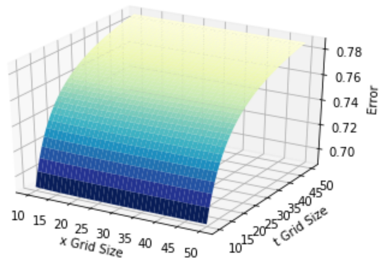

Figure 3.4: Error of grid sizes.

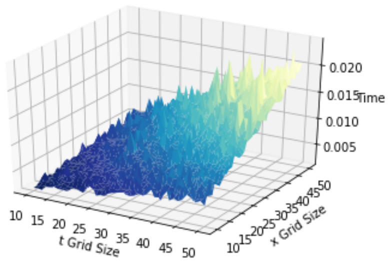

Figure 3.5: Timing grid sizes.

3.7.2 Base Case

Timing the base case for thomas algorithm, intel solver and cyclic reduction. The timings are mean of 1000 trials and each trial adds a random number $0 < \epsilon < 10^{-}7$ to the step size to avoid compiler optimizations.

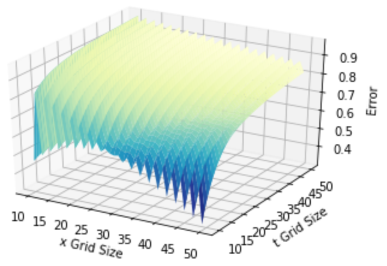

Figure 3.6: Error of grid sizes.

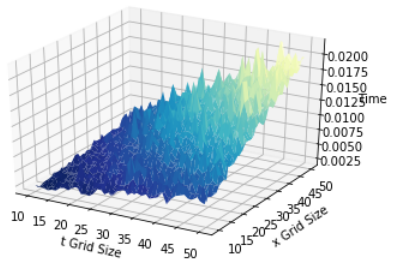

Figure 3.7: Timing grid sizes.

Environment	Explicit Method	Crank Nicolson
Visual Studio Compiler x86	0.02496743	0.02590181
Visual Studio Compiler x64	0.02105383	0.02216256
Intel Compiler x86	0.02563324	0.02633357
Intel Compiler x64	0.02210323	0.02298945

Table 3.1: Solving Black-Scholes with Thomas Algorithm base case.

Environment	Explicit Method	Crank Nicolson
Visual Studio Compiler x86		
Visual Studio Compiler x64		
Intel Compiler x86		
Intel Compiler x64	0.02171834	0.02286124

Table 3.2: Solving Black-Scholes with Intel Solver base case.

Testing	Explicit Method	Crank Nicolson	ADI
Visual Studio Compiler x86	0.02502006	0.02536817	0.07766176
Visual Studio Compiler x64	0.02185875	0.02211793	0.06512516
Intel Compiler x86	0.02685609	0.02694334	0.08277156
Intel Compiler x64	0.0225934	0.0226115	0.06614703

Table 3.3: Solving heat equation with Thomas Algorithm base case.

Testing	Explicit Method	Crank Nicolson	ADI
Visual Studio Compiler x86			
Visual Studio Compiler x64			
Intel Compiler x86			
Intel Compiler x64	0.02239165	0.02195975	0.06337759

Table 3.4: Solving heat equation with Intel Solver base case.

3.7.3 Visual Studio Optimizations

The optimization switch for enabling most speed optimizations /Ox will be utilized in this section.

Environment	Explicit Method	Crank Nicolson
Visual Studio Compiler x86		
Visual Studio Compiler x64		
Intel Compiler x86		
Intel Compiler x64		

Table 3.5: Solving Black-Scholes with Thomas Algorithm using Visual Studio optimizations.

Environment	Explicit Method	Crank Nicolson
Visual Studio Compiler x86		
Visual Studio Compiler x64		
Intel Compiler x86		
Intel Compiler x64		

Table 3.6: Solving Black-Scholes with Intel Solver using Visual Studio optimizations.

3.7.4 Cyclic Reduction with OpenMP

Testing	Explicit Method	Crank Nicolson	ADI
Visual Studio Compiler x86			
Visual Studio Compiler x64			
Intel Compiler x86			
Intel Compiler x64			

Table 3.7: Solving heat equation with Thomas Algorithm using Visual Studio optimizations.

Testing	Explicit Method	Crank Nicolson	ADI
Visual Studio Compiler x86			
Visual Studio Compiler x64			
Intel Compiler x86			
Intel Compiler x64			

Table 3.8: Solving heat equation with Intel Solver using Visual Studio optimizations.

Environment	Cyclic Reduction
Visual Studio Compiler x86	0
Visual Studio Compiler x64	0
Intel Compiler x86	0
Intel Compiler x64	0

Table 3.9: Solving Black-Scholes with cyclic reduction.

Environment	Cyclic Reduction
Visual Studio Compiler x86	0
Visual Studio Compiler x64	0
Intel Compiler x86	0
Intel Compiler x64	0

Table 3.10: Solving heat equation with cyclic reduction.

Chapter 4

Conclusion

A Critique of the Crank Nicolson Scheme Strengths and Weaknesses for FinancialInstrument Pricing + rannacher AVX and Intrinsics CPUs are pipelining and use of SSE/SIMD kusswurm registers with Advanced Vector Extensions(AVX 512)GPGPU In the case of General Purpose GPUs, CUDA or Open Computing Language(OpenCL) can be utilized but can be challenging because of the requirement of delicate memory management. [30], cloud functions [11]. change black scholes to a different pricing pde like interest rate derivatives The HJM model [17] Results concludela

Appendix A

Usage of chrono class

Should code example be in appendix or stay here?

```
auto start = std::chrono::high_resolution_clock::now();
    Portion of code to be timed
auto finish = std::chrono::high_resolution_clock::now();
std::chrono::duration<double> elapsed = finish - start;
std::cout << "Elapsed time: " << elapsed.count() << " s\n";
```

Appendix B

Implementation of the `PDE` class

Parabolic partial differential equation can be denoted as

$$\frac{\partial u}{\partial t} = a(t,x)\frac{\partial^2 u}{\partial x^2} + b(t,x)\frac{\partial u}{\partial x} + c(t,x)u(t,x) + d(t,x)$$

$a(t,x)$ denotes diffusion coefficient, $b(t,x)$ convection coefficient, $c(t,x)$ reaction coefficient, $d(t,x)$ source coefficient analytic solution, initial conditions boundary conditions

Appendix C

Implementation of the `FiniteDifferenceMethod` class

```
void stepSize();
void initialConditions();
void boundaryConditions();
void innerDomain();
void timeMarch();
```

Bibliography

[1] Saeed Amen. *Trading Thalesians: What the Ancient World Can Teach Us About Trading Today*, pages 39–60. Palgrave Macmillan UK, London, 2014.

[2] TRAVIS M Austin, Markus Berndt, and J David Moulton. A memory efficient parallel tridiagonal solver. *Preprint LA-VR-03-4149*, 2004.

[3] Fischer Black and Myron Scholes. The pricing of options and corporate liabilities. *Journal of Political Economy*, 81(3):637–654, 1973.

[4] Barbara Chapman, Gabriele Jost, and Ruud van der Pas. *Using OpenMP: Portable Shared Memory Parallel Programming (Scientific and Engineering Computation)*, pages 23 – 25. The MIT Press, 2007.

[5] Jaehyuk Choi. Sum of all black–scholes–merton models: An efficient pricing method for spread, basket, and asian options. *Journal of Futures Markets*, 38(6):627–644, 2018.

[6] Gustavo Chávez, George Turkiyyah, Stefano Zampini, Hatem Ltaief, and David Keyes. Accelerated cyclic reduction: A distributed-memory fast solver for structured linear systems. *Parallel Computing*, 74:65 – 83, 2018. Parallel Matrix Algorithms and Applications (PMAA'16).

[7] J. Crank and P. Nicolson. A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type. *Advances in Computational Mathematics*, 6(1):207–226, Dec 1996.

[8] Jim Douglas, Jr. Alternating direction methods for three space variables. *Numer. Math.*, 4(1):41–63, December 1962.

[9] D.J. Duffy. *Financial Instrument Pricing Using C++*. The Wiley Finance Series. Wiley, 2013.

[10] G. Evans, J.M. Blackledge, and P. Yardley. *Numerical methods for partial differential equations*. Springer undergraduate mathematics series. Springer, 2000.

[11] Sadjad Fouladi, Francisco Romero, Dan Iter, Qian Li, Shuvo Chatterjee, Christos Kozyrakis, Matei Zaharia, and Keith Winstein. From laptop to lambda: Outsourcing everyday jobs to thousands of transient functional containers. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 475–488, Renton, WA, July 2019. USENIX Association.

[12] P. Glasserman. *Monte Carlo Methods in Financial Engineering*, pages 2 – 3. Applications of mathematics : stochastic modelling and applied probability. Springer, 2004.

[13] R. W. Hockney. A fast direct solution of poisson's equation using fourier analysis. *J. ACM*, 12(1):95–113, January 1965.

[14] Intel. Intel math kernel library for c, gtsv. `https://software.intel.com/en-us/mkl-developer-reference-c-gtsv`, 2019. [Online; accessed 7-August-2019].

[15] Stephen Jewson and Mihail Zervos. The black-scholes equation for weather derivatives. *SSRN Electronic Journal*, 09 2003.

[16] F.C. Klebaner. *Introduction to Stochastic Calculus with Applications*. Introduction to Stochastic Calculus with Applications. Imperial College Press, 2005.

[17] P. Kohl-Landgraf. *PDE Valuation of Interest Rate Derivatives: From Theory to Implementation*, pages 423–438. Books on Demand, 2007.

[18] Matthew J. Hancock. The 1-d heat equation (mit course 18.303, linear partial differential equations). https://ocw.mit.edu/courses/mathematics/18-303-linear-partial-differential-equations-fall-2006/lecture-notes/heateqni.pdf, 2006. [Online; accessed 7-August-2019].

[19] Robert C. Merton. Option pricing when underlying stock returns are discontinuous. *Journal of Financial Economics*, 3(1):125 – 144, 1976.

[20] Microsoft. /gf (eliminate duplicate strings). https://docs.microsoft.com/en-us/cpp/build/reference/gf-eliminate-duplicate-strings?view=vs-2019, 2016. [Online; accessed 10-August-2019].

[21] Microsoft. /gy (enable function-level linking). https://docs.microsoft.com/en-us/cpp/build/reference/gy-enable-function-level-linking?view=vs-2019, 2016. [Online; accessed 10-August-2019].

[22] Microsoft. /ob (inline function expansion). https://docs.microsoft.com/en-us/cpp/build/reference/ob-inline-function-expansion?view=vs-2019, 2016. [Online; accessed 10-August-2019].

[23] Microsoft. /oi (generate intrinsic functions). https://docs.microsoft.com/en-us/cpp/build/reference/oi-generate-intrinsic-functions?view=vs-2019, 2016. [Online; accessed10-August-2019].

[24] Microsoft. /os, /ot (favor small code, favor fast code). https://docs.microsoft.com/en-us/cpp/build/reference/os-ot-favor-small-code-favor-fast-code?view=vs-2019, 2016. [Online; accessed 10-August-2019].

[25] Microsoft. /o options (optimize code). https://docs.microsoft.com/en-us/cpp/build/reference/o-options-optimize-code?view=vs-2019, 2017. [Online; accessed 10-August-2019].

[26] Microsoft. /og (global optimizations). https://docs.microsoft.com/en-us/cpp/build/reference/og-global-optimizations?view=vs-2019, 2017. [Online; accessed 10-August-2019].

[27] Microsoft. /oy (frame-pointer omission). https://docs.microsoft.com/en-us/cpp/build/reference/oy-frame-pointer-omission?view=vs-2019, 2018. [Online; accessed 10-August-2019].

[28] K. W. Morton and D. F. Mayers. *Numerical Solution of Partial Differential Equations: An Introduction*. Cambridge University Press, 2 edition, 2005.

[29] Martin Neuenhofen. A time-optimal algorithm for solving (block-) tridiagonal linear systems of dimension n on a distributed computer of n nodes. *arXiv preprint arXiv:1801.09840*, 2018.

[30] Samuel Palmer. Accelerating implicit finite difference schemes using a hardware optimised implementation of the thomas algorithm for fpgas. *arXiv preprint arXiv: 1402.5094, 2014*, 2014.

[31] D. W. Peaceman and H. H. Rachford. The numerical solution of parabolic and elliptic differential equations. *Journal of the Society for Industrial and Applied Mathematics*, 3(1):28–41, 1955.

[32] Charles Petzold. *Programming Windows, Fifth Edition*. Microsoft Press, Redmond, WA, USA, 5th edition, 1998.

[33] Sriram Ramaswamy. Pollen grains, random walks and einstein. *Resonance*, 5, 03 2000.

[34] G.D. Smith, G.D. Smith, G.D.S. Smith, M. Smither, and Oxford University Press. *Numerical Solution of Partial Differential Equations: Finite Difference Methods*. Oxford applied mathematics and computing science series. Clarendon Press, 1985.

[35] D. Tavella and C. Randall. *Pricing Financial Instruments: The Finite Difference Method*. Wiley Series in Financial Engineering. Wiley, 2000.

[36] Andrew V Terekhov. Parallel dichotomy algorithm for solving tridiagonal system of linear equations with multiple right-hand sides. *Parallel Computing*, 36(8):423–438, 2010.

[37] The OpenMP Architecture Review Board. What problem does openmp solve? `https://www.openmp.org/about/openmp-faq/`, 2018. [Online; accessed 10-August-2019].

[38] J.W. Thomas. *Numerical Partial Differential Equations: Finite Difference Methods*. Texts in Applied Mathematics. Springer New York, 2013.

[39] Llewellyn Hilleth Thomas. Elliptic problems in linear difference equations over a network. *Watson Sci. Comput. Lab. Rept., Columbia University, New York*, 1, 1949.

[40] Ruud van der Pas, Eric Stotzer, and Christian Terboven. *Using OpenMP – The Next Step: Affinity, Accelerators, Tasking, and SIMD*, pages 25 – 28. The MIT Press, 1st edition, 2017.

[41] Paul Wilmott. *Paul Wilmott Introduces Quantitative Finance*. Wiley-Interscience, New York, NY, USA, 2 edition, 2007.

[42] Tomasz Zastawniak and Maciej J. Capinski. *Numerical Methods in Finance with C++*, pages 149 – 150. Mastering Mathematical Finance. Cambridge University Press, 7 2012.