Fast Fourier Transform for Option Pricing: Improved Mathematical Modeling and Design of an Efficient Parallel Algorithm

by

Sajib Barua

A thesis
submitted to the University of Manitoba
in partial fulfilment of the
requirements for the degree of
Master of Science
in
Computer Science

Winnipeg, Manitoba, Canada, 2004

Abstract

The Fast Fourier Transform (FFT) has been used in many scientific and engineering applications. The use of FFT for financial derivatives has been gaining momentum in the recent past. In this thesis, i) we have improved a recently proposed model of FFT for pricing financial derivatives to help design an efficient parallel algorithm. The improved mathematical model put forth in our research bridges a gap between quantitative approaches for the option pricing problem and practical implementation of such approaches on modern computer architectures. The thesis goes further by proving that the improved model of fast Fourier transform for option pricing produces accurate option values. ii) We have developed a parallel algorithm for the FFT using the classical Cooley-Tukey algorithm and improved this algorithm by introducing a data swapping technique that brings data closer to the respective processors and hence reduces the communication overhead to a large extent leading to better performance of the parallel algorithm.

We have tested the new algorithm on a 20 node SunFire 6800 high performance computing system and compared the new algorithm with the traditional Cooley-Tukey

algorithm. Option values are calculated for various strike prices with a proper selection of strike-price spacing to ensure fine-grid integration for FFT computation as well as to maximize the number of strikes lying in the desired region of the stock price. Compared to the traditional Cooley-Tukey algorithm, the current algorithm with data swapping performs better by more than 15% for large data sizes. In the rapidly changing market place, these improvements could mean a lot for an investor or financial institution because obtaining faster results offers a competitive advantages.

Keywords: Financial Derivatives; Option Pricing; Fast Fourier Transform; Mathematical Modeling; Parallel Algorithm; Data Locality.

This thesis is dedicated to my parents and grandparents.

Acknowledgement

I owe my thanks to many people who helped make this thesis possible.

First, my thesis committee members, Dr. Peter C.J. Graham and Dr. Abba Gumel, whom I thank for the valuable comments and suggestions that helped me to improve the quality of the final document.

I feel very privileged to have worked with my supervisor Dr. Ruppa K. Thulasiram (Tulsi). I received lot of help and advice from Dr. Parimala Thulasiraman as well through out the course of my thesis. To each of them I owe a great debt of gratitude for their patience, inspiration and friendship. Dr. Tulsi has taught me a great deal about the field of high performance computing in computational finance by sharing with me the joy of discovery and investigation that is the heart of research. He provided endless hours of advice and guidance regarding research, writing, speaking, life in general, and the freedom to pursue my own ideas.

Dr. Tulsi taught me to work as a researcher the subtleties of writing, the peculiarities

of publishing papers in computer science, the importance of public speaking and getting to know other researchers in the community. He suggested the connection between the parallel computing and the computational finance, and his insights and ideas are infused throughout this work. His tireless editorial effort vastly improved the quality of this dissertation, and of much of the rest of my research. Thanks must also go to Dr. Thulasiraman who introduced me with the fast Fourier transform and its application in the course project and helped me with her valuable comments and guidance for my defense.

Most of all, my parents and brother made this possible. Whether it is nature or nurture that makes more of a person is an old question, but I have no doubt that the nurture, care, and love given by my parents and brother is more responsible than anything else for making me who I am today. My family has been extremely understanding and supportive of my studies. I feel very lucky to have a family that shares my enthusiasm for academic pursuits.

A big thanks to each of the members of the Parallel Algorithm Research In Manitoba's ALgorithms and Applications (PARIMALA) Lab for helping mold my research through discussions and tearing me apart with questions during numerous weekly meetings. Without this rich environment I doubt that many of my ideas would have come to fruition. Thanks are also due to all my friends and house mates for keeping me happy and for providing me with a life outside of computer science.

I would like to acknowledge the Department of Computer Science, the University of Manitoba for the Departmental Scholarship and partial financial support from the NSERC and the University of Manitoba Research Grant Program (URGP) funded projects.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Some of the problems facing the finance industry have been recognized as grand challenge problems [Amm89, HL01, KNR00, Zen99] in addition to problems from science and engineering [Tuc97]. The finance industry demands efficient algorithms and high-speed for solving problems such as option pricing, risk analysis, and portfolio management. We address the problem of option pricing using advanced scientific computing techniques in this thesis.

The mathematical models of finance represented either in continuous or discrete form are generally nonlinear, multidimensional and require advanced knowledge in areas such as measure theory and stochastic partial differential equations. Further, they are very challenging in terms of finding closed form analytical solutions. Computational solutions are inevitable for such problems in finance as option pricing and portfolio optimization. However, numerical computing for finance problems applying advanced computer architectures has gained prominence only in the recent past.

One of the main characteristics of financial markets is that changes can occur very rapidly as Black Monday (October 19, 1987) [Hul02] has shown. In a very short time,

asset prices may change significantly. To respond adequately to such changes, it is necessary for large market participants, like brokers and banks, to evaluate information as fast as possible. This is important to make, for instance, the appropriate portfolio changes. Any delay in information access could mean a financial loss. According to the methodology developed by Black and Scholes [BS73], it is possible to approximate the implication of market changes on portfolio positions. Their method, however, requires the solution of a set of partial differential equations by means of numerical integration. Depending on the number of assets taken into account, obtaining such a solution can be highly computationally intensive. For this problem, supercomputers can facilitate real time information processing and near instantaneous response.

Traditionally the option pricing problem has been studied predominantly using the Black-Scholes model [BS73], using CRR model [CRR79] or using Monte-Carlo simulation [Boy77]. The Black-Scholes model uses a continuous approach in representing the option pricing problem as a partial differential equation while the other two models formulate the option problem using more intuitive and discrete approaches. Since the finance community is devoid of the computational knowledge for a long time, the researchers tried to solve the problem for closed form solution that led to many assumptions on the model, which diluted the problem under study. A recent entry into the finance community is the use of numerical techniques such as finite-differencing [TR00] for solving the Black-Scholes and other continuous models. A much more recent entry is the use of transform techniques such as Laplace transform [DHC$^+$04] and fast Fourier transform [CM99]. This thesis addresses the option pricing problem and its computational issues. We have employed a modern scientific computing approach, using Fast Fourier Transform (FFT), to solve this problem.

1.1 Definition

There are several areas of research in finance, both in industry and academia. Some of the important ones are option pricing, risk analysis, portfolio management, data mining, interest rate modeling, and volatility. These problems may occur among the different financial markets including stock, equities or bond markets; currency or foreign exchange markets; commodity markets such as oil, gold, copper, and electricity; as well as options markets such as complex derivative products, etc.

Only the relevant terms involved in above markets and research areas pertaining to option pricing are described in this section. For many other definitions, readers are directed to [Hul02]. The terms financial derivatives, derivative securities, derivative products, contingent claims or just derivatives are interchangeably used in the literature.

There are two parties in a contract: the *holder* and the *writer*. A call option holder has the right to purchase a security/asset at a set price, if he/she chooses, within a certain period of time. The option writer has the obligation to fulfill that option. There are two types of options considered:

Call Option: An agreement that gives an holder the right (but not the obligation) to *purchase* a prescribed asset, known as the underlying asset at a specified price known as the *exercise price* (or *strike price*) within a specific future date (or *expiration date*) [Hul02].

Put Option: An option contract that gives an holder the right (but not the obligation) to *sell* a prescribed asset, known as the underlying asset at a specified price known as the *exercise price* (or *strike price*) within a specific future date (or *expiration date*) [Hul02].

The issues that are important between the holder and the writer are:

- The cost for the *holder's* right;

- Minimizing the risk associated with the *writer's* obligation.

To gain an intuitive feel, let us consider the following example. Consider a stock price is $\$20$ on November $15, 2003$. With a call option, an investor enter a deal to buy that stock at $\$21$ on May $14, 2004$. On May $14, 2004$ suppose the stock price is $\$30$. On this day, he/she can exercise his/her option of buying the stock at $\$21$ and then by immediately selling in the open market at $\$30$, he/she can gain $\$9$. However, on May $14, 2004$, if the stock price is $\$15$, he/she is not obligated to buy the stock[1](why buy something at a higher price when the same is available at a lower cost in the open market?). The question then is why would anyone write an option? The answer is that the writer wants to make a profit by taking a "view or speculation" on the market.

Call and Put options, as discussed above, form a small section of derivative products. Options, as described earlier, which can only be exercised at maturity are called European options [Hul02]. An American option is an option that may be exercised at any time prior to expiry [Hul02]. American options are analogous to the *free boundary problems* [TR00] in engineering, a mathematically challenging problem since there is no set boundary to refer to or iterate on. The issue here is determining the best time to exercise the option. Exotic options have values, which depend on the history of an asset price. They are analogous to the *crack propagation problems* [TKA$^+$00, Thu03a] in engineering, where damage at one particular point on a material is a function of the distance from the crack source and its propagation along the material surface and interior.

[1]*Note: The call value is, therefore, a function of the strike price.*

1.2 Organization

The thesis is organized as follows. First, we have introduced in this chapter some basic terminology of the option pricing problem for the sake of completeness and clarity. At the same time we will use limited finance "jargon" in the thesis as a whole. The related work is described in chapter 2 including some background on option pricing using four different techniques (binomial lattice, Monte Carlo, finite difference, and fast Fourier transform). In chapter 3, we discuss the option pricing problem formulation using FFT based on the CM (Carr and Madan) model. We state the methodologies for the current research in chapter 4 with a detailed description of the improvement to the mathematical modeling of FFT for option pricing which is one of the major contributions of this thesis. Then in chapter 5, section 5.1 shows how the basic FFT equation can be parallelized. Design and development of a new parallel algorithm for calculating the call value using FFT, another major contribution of this thesis, is described in section 5.2. In chapter 6, we show the analytical and performance results of our new parallel FFT algorithm. Finally, we present conclusions in chapter 7 and future work in chapter 8.

Chapter 2

Background and Related Work

The use of FFT technique in option pricing is relatively new [Cer04] and the related literature is rather thin. However, there are many other numerical techniques available for solving the option pricing problem, such as: binomial lattice [CRR79], Monte-Carlo [Boy77], and finite-differencing techniques [TR00], as described below. Though a direct comparison of our results using the FFT computation with any of these techniques is not feasible, a general introduction to these techniques is warranted. While the finite-differencing technique is gaining momentum among finance engineers, the binomial lattice and Monte-Carlo method have been used predominantly. A thorough coverage of all these methods is beyond the scope of this current research and hence only very selected works that have relevance to parallel computing are discussed in this section. We describe each of these methods in the following sections.

In general, two types of numerical techniques are used for option valuation (i) techniques which directly approximate the underlying stochastic process (these techniques are more intuitive) and (ii) techniques which approximate the resulting partial differential equations. The first approach includes Monte Carlo simulation [Boy77] and different

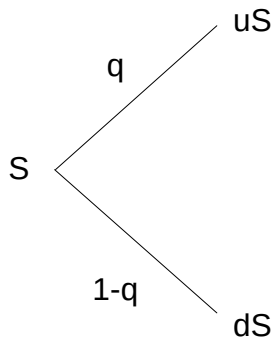

Figure 2.1: One-Step Binomial Model

lattice approaches used by Cox, Ross, and Rubinstein [CRR79]. The second approach includes numerical integration and finite difference schemes (both implicit and explicit) [TR00]. In this chapter, we will describe the binomial lattice approach in section 2.1, Monte Carlo simulation in section 2.2, the finite differencing technique in section 2.3. Option valuation using FFT as proposed by different authors and spread options based on CM [CM99] approach (chapter 3) are described in section 2.4. Details of work applying the FFT to option pricing are described in chapter 3.

2.1 Option Pricing Using the Binomial Lattice Approach

The binomial lattice approach was first popularized by Cox, Ross, and Rubinstein (C-R-R) in 1979 [CRR79]. The stock price movement of an underlying asset can be described by a strict multiplicative binomial process over successive periods as follows: A stock price S at the beginning of a period, can increase (by a multiplicative factor) to uS with probability q (shown in figure 2.1). It may decrease to dS with complementary probability $(1-q)$ at the end of the period. Here u and d denoted the rate of return when the stock goes up or down respectively, hence $d = 1/u$. When the asset price moves up

from $S \rightarrow Su$, the pay-off from the derivative is f_u where f is the derivative price. When the price goes down from $S \rightarrow Sd$, the pay-off from the derivative is f_d.

A binomial lattice can be used to capture the asset price movement directly. To capture the price movements closely, large numbers of intermediate periods between the start and maturity of the contract are needed. However, due to its simplicity and intuitiveness, this method has become a text book method for understanding the pricing process. This is an intuitive technique and, hence, is very useful in understanding the problem and pricing process generally. The advent of faster computers has given this method new life in the industry generating approximate solutions to the given problem which are then used as initial conditions in other more complicated numerical techniques. There is inherent concurrency in this method that can be exploited for parallel implementation. Many algorithms have been developed using this technique for the option pricing problem (for example [CJEV98, TDG00, TLN$^+$01, TB04]). The advantage of this technique is the intuitiveness of the method whereas a major disadvantage is in the inaccuracy of the results.

The original one-step binomial model for option pricing developed in [CRR79] sets up a portfolio of stocks and options in such a way that there is no uncertainty about the value of the portfolio at the end of the contract period. For this, it is assumed that the option is risk-free. As the portfolio has no risk, the return earned will be equal to the risk-free interest rate. This assumption is taken as the initial step to calculate the option value with the cost of setting up the portfolio.

We first construct a portfolio with a long position (the state of actually owing a security, contract, or commodity) in Δ s*hares of stocks* and a short position (in future contract, the promise to sell a fixed amount of goods at a fixed price in the future) in *one call option* then we calculate the value of Δ which makes the portfolio risk-less. First a closed form formula for Δ will be defined using the general terms. If the stock price increases

from $S \rightarrow S_u$, the value of the portfolio at the end of the period of the derivative will be $Su\Delta - f_u$ and if the stock price decreases the value of the portfolio will be $Sd\Delta - f_d$. For a portfolio which is to be risk-neutral:

$$\begin{aligned} Su\Delta - f_u &= Sd\Delta - f_d \\ S(u-d)\Delta &= f_u - f_d \\ \Delta &= \frac{f_u - f_d}{S(u-d)}. \end{aligned} \tag{2.1}$$

Equation (2.1) is the ratio of the change in the derivative price to the change in the stock price at time T. Let r be the risk-free interest rate. By discounting the end-of-the-period value of the portfolio $(Su\Delta - f_u)$ and using the growth rate formula [WHD95], we can find the present value of the portfolio to be $(Su\Delta - f_u)e^{-r\Delta t}$. Since the cost of setting up the portfolio is $S\Delta - f$, for a risk-less portfolio

$$S\Delta - f = (Su\Delta - f_u)e^{-r\Delta t}. \tag{2.2}$$

Substituting equation (2.1) in equation (2.2) and manipulating we can get

$$f = e^{-r\Delta t}(pf_u + (1-p)f_d), \tag{2.3}$$

where

$$p = \frac{e^{r\Delta t} - d}{u - d}. \tag{2.4}$$

Equation (2.3) and equation (2.4) calculate the one-step binomial model in pricing derivative. The value of p calculated in equation (2.4) can be treated as the probability of an up movement in the stock price. The function $(pf_u + (1-p)f_d)$ in equation (2.3) calculates the expected pay-off from the derivatives.

2.1.1 Two-Step Binomial Tree

The two-step binomial model (when the contract period is split into two steps) follows the same process as the one-step binomial model. For simplicity we will assume that

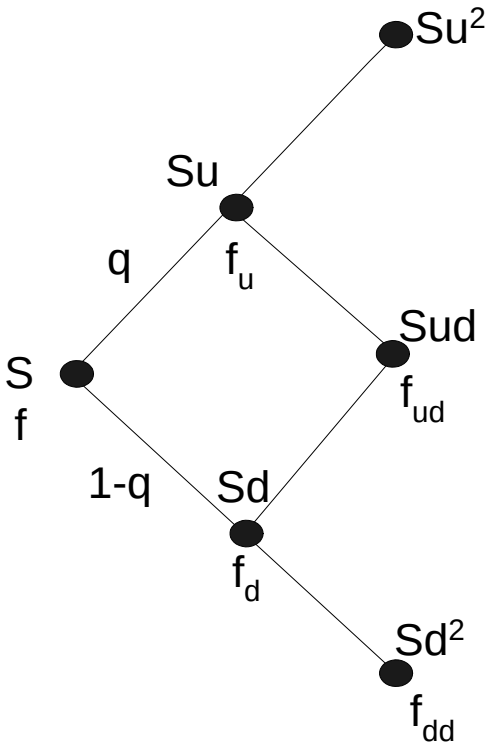

Figure 2.2: General Two-Step Binomial Model

the increase and decrease of stock price will be the same. Figure 2.2 shows a general two-step binomial model.

At each step of the lattice, the price goes up by a factor of u from the initial value or goes down by a factor of d from the initial value. Using equation (2.3) repeatedly, the following equations can be derived:

$$f_u = e^{-r\Delta t}(pf_{uu} + (1-p)f_{ud}). \quad (2.5)$$

$$f_d = e^{-r\Delta t}(pf_{ud} + (1-p)f_{dd}). \quad (2.6)$$

$$f = e^{-r\Delta t}(pf_u + (1-p)f_d). \quad (2.7)$$

Substitution of equation (2.5) and equation (2.6) in equation (2.7) gives:

$$f = e^{-2r\Delta t}(p^2 f_{uu} + 2p(1-p)f_{ud} + (1-p)^2 f_{dd}). \tag{2.8}$$

2.1.2 A Numerical Example

Figure 2.3 shows an example of the two-step binomial method. Node A is the current start time. Nodes B and C are the intermediate points in the contract period of 3 months. D, E, and F are the possible terminal points that the stock will reach after 3 months from B or C. The periods are assumed to be equal for simplicity and the up and down movement are also assumed to be at a constant rate (of 10%) where $u = 1.1$ and $d = 0.9$. In general, the increase and decrease of the stock price, however, does not need to be equal as is assumed in this numerical example. At D, E, and F (the leaf nodes) the

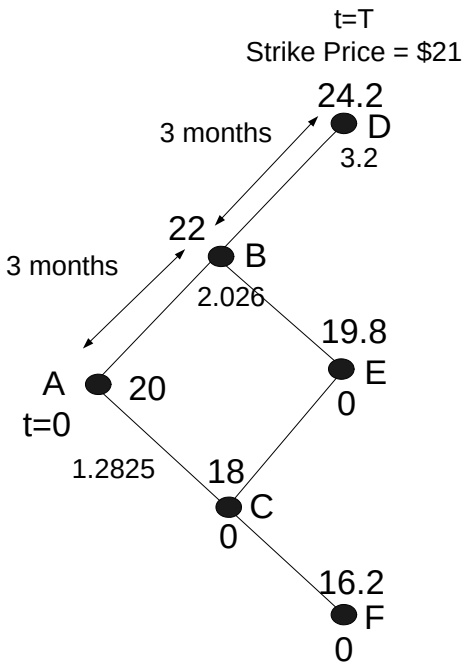

Figure 2.3: Numerical Example of a 2-Step Binomial Model

option prices calculated are the pay-off from the derivative. They can be calculated from the strike price and speculated stock price. Therefore, at D, the $option\ price = stock\ price - strike\ price = 24.2 - 21 = \$3.2$. This is known as local pay-off, that is, the pay-off from the option as soon as the node is reached. At nodes E and F since the $stock\ price < strike\ price$, the option value at these nodes is $\$0$. Also at C, the option price is 0. At B, the local pay-off can be calculated to be $\$1$ ($stock\ price - strike\ price = 22 - 21 = \$1$). Assuming a 10% increase and decrease (i.e., $u = 1.1$ and $d = 0.9$), at B, p can be calculated using equation (2.4) to be 0.6523. Therefore, the option value at B can be calculated using equation (2.3) as 2.026. If it is an American option, it is not desirable to exercise the option at node B reached as the option value calculated is $\$1$. Waiting for the second pass is a better decision after the option values at nodes D and E have been calculated. Again it is obvious that nodes B, D, and E are nothing but the one-step binomial model so all the corresponding formulas can be applied here.

After calculating the stock price and the option value at the nodes B and C, option value at node A is calculated as 1.2825 using the one-step binomial formula given by equation (2.3). In this model the up and down movement is assumed to be equal for the sake of simplicity. This assumption led to a constant p at each node. This assumption can be relaxed in a more realistic analysis of portfolio.

2.2 Option Pricing Using the Monte Carlo Technique

Another traditional method is the Monte-Carlo simulation [Boy77]. A random walk simulates the price movement of the underlying asset of an option. Thousands of such simulations capture all possible asset price movements. The drawback of this technique is the requirement of thousands of simulations before a reasonable error bound is achieved in

the option values [Var96]. Since the simulations are independent of each other, and there is no communication between them, we have large potential for parallelism in this technique. This is known as "embarrassingly parallel" problem [RST04]. The Monte-Carlo simulation does not pose serious challenges from the parallel computing perspective and is used as a benchmark technique in finance industries with a network of workstations. As mentioned before, more simulations would yield better results and hence with the availability of large computing power, accuracy of the solution can be improved. There are improvements reported in the literature on this technique [FLM$^+$01, RST04, Sri02] both from the algorithmic and parallel computing perspectives. The technique is common among finance practitioners due to its intuitive nature.

The Monte Carlo approach is an established numerical method for solving derivatives involving highly random parameters. With more sampling, more accurate results are obtained and the number of samples is dependent on the availability of computational resources.

The basic theme in the theory of derivative pricing is the random walk of asset prices [Boy77]. One such theory is given by the Black-Scholes formula [BS73] which gives the relationship between option prices and expectations. The Monte Carlo technique exploits this relationship to find option prices from simulations of the asset prices. In brief, the value of an option is the expected present value of the pay-off at the expiry under a risk-neutral random walk of the underlying asset. The risk neutral random walk for asset price S is:

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

where μ is the average rate of growth in the asset price called the drift (deterministic part in the price change), σ is the volatility (stochastic part in the price change), a measure of the standard deviation of the returns and t is the time. If there is no randomness involved; $dS = rSdt$. The term dX which is certainly a feature of asset price is known

as the *Weiner process* [WHD95] and has the following properties:

- dX is a random variable from a normal distribution
- the mean of $dX = 0$
- the variance of $dX = dt$

The option value can be calculated from the following equation:

$$option\ value = e^{-r(T-t)}E[payoff(S)], \tag{2.10}$$

where the expectation is with respect to the risk-neutral random walk. Equation (2.10) leads to an estimate of the option through the following approach:

Simulate the risk-neutral random walk over the contract period, starting at today's value of the asset. This gives one realization of the underlying asset's price path. For this realization, the option pay-off is calculated as $\max(S - X, 0)$, where X is the strike price. This calculation is done many times over the period and then the average pay-off over all this realization is calculated. Then the present value of this average is considered as the option value.

The first step of this algorithm requires the generation of random numbers from a standard normal distribution. The asset price at each time step over the period is updated from these generated random increments using this formula

$$\delta S = \mu S \delta t + \sigma S \phi \sqrt{\delta t}. \tag{2.11}$$

where δt is the increment in the time step and ϕ is the standard normal distribution.

The latest calculated value of S is then put in the right hand side of equation (2.11) to calculate δS and hence the next value of S. One can find a simple and exact time

stepping algorithm where the risk neutral stochastic differential equations for S will be of the form [Thu03b]:

$$d(\log S) = (r - \frac{1}{2}\sigma^2)dt + \sigma dx. \tag{2.12}$$

Integration of this will give:

$$S(t) = S(0)exp\Big[(r - \frac{1}{2}\sigma^2)t + \sigma \int_0^t dx\Big]. \tag{2.13}$$

Over a time step δt

$$\begin{aligned} S(t+\delta t) &= S(t) + \delta S \\ &= S(t)exp\Big[(r - \frac{1}{2}\sigma^2)\delta t + \sigma\sqrt{\delta t}\phi\Big]. \end{aligned} \tag{2.14}$$

Since this expression is exact and simple, it is the best time-stepping algorithm to use. Besides, if the pay-off only depends on the final asset value (i.e. European and path independent option), the final asset price can be simulated in one giant leap with a time-step of T.

The Monte Carlo approach for option pricing is better than other approaches because of its simple mathematics. Additionally, more simulations may be simply used to give better accuracy in the computed result.

2.3 Option Pricing Using the Finite Difference Technique

Finite difference is one of the easier techniques among the CFD techniques used for pricing option. In this method, the value of a derivative is calculated by solving the differential equations manifesting the derivatives under study. The differential equations are transformed into several difference equations and the set of difference equations is solved iteratively. The price of an option $c(S, t)$ can be obtained by solving the familiar

partial differential equation, known as the Black-Scholes equation [BS73].

$$\frac{\partial c(S_t,t)}{\partial S_t} + \frac{\partial c(S_t,t)}{\partial t} + \frac{1}{2}\frac{\partial^2 c(S_t,t)}{\partial S_t^2} S_t^2 \sigma^2 = rc(S_t,t). \tag{2.15}$$

Solving the simple Black-Scholes model for option pricing numerically has been a daunting task since the subject of numerical analysis is foreign to most of the finance academics. This model manifests the option pricing problem as a stochastic partial differential equation and solving this equation analytically is a formidable task. Computational Fluid Dynamics (CFD) techniques such as finite difference and finite element techniques [Cla98, CTIT04, TR00, TZG04, ZFV98, Zhe02] have been employed to solve the Black-Scholes model numerically. These techniques partition the solution space in fine rectangular/triangular meshes to represent time and asset prices of the underlying asset in the option. Marching the computation over large numbers of time steps yields the option values. Accuracy depends on the mesh size and the number of time steps. For a very accurate solution, to the 10th decimal place, millions of time steps are not uncommon. This technique is challenging both in terms of conceptualization and parallel algorithm development. The computational mesh has to be partitioned and assigned to various processors in a parallel system which raises both computational and communication issues.

For standard European options, a closed form solution of the Black-Scholes equation can be found but in most cases, no closed form solution can be found. For European call options, the boundary condition of equation (2.15) is $c(S_T, T) = \max(0, S_T - X)$ where X is the exercise price. Though the binomial lattice method can be used, the finite difference method is more efficient. Moreover, the binomial lattice method is a special case of the finite difference technique mathematically.

In this section, we briefly describe the $\log$ transform method proposed by Brennan and Schwartz [BS77] to derive a simple European option pricing formula. Let $y = \ln S$ and $w(y,t) = c(S,t)$ as the price of the call option at time t. Here, the price is defined in

terms of the log of the asset price and time t. Eliminating y and t notations, we can get:

$$\begin{aligned}\frac{\partial c}{\partial S} &= \frac{\partial w}{\partial y}e^{-y} \\ \frac{\partial^2 c}{\partial S^2} &= \left[\frac{\partial^2 w}{\partial y^2} - \frac{\partial w}{\partial y}\right]e^{-2y} \\ \frac{\partial c}{\partial t} &= \frac{\partial w}{\partial t}.\end{aligned}$$

Substitution of all the above equations in equation (2.15) gives

$$\frac{1}{2}\sigma^2\frac{\partial^2 w}{\partial y^2} + \left[r - \frac{1}{2}\sigma^2\right]\frac{\partial w}{\partial y} + \frac{\partial w}{\partial y} - rw = 0. \tag{2.16}$$

The above equation can be expressed as:

$$\frac{1}{2}\sigma^2\frac{\Delta^2 w}{\Delta y^2} + \left[r - \frac{1}{2}\sigma^2\right]\frac{\Delta w}{\Delta y} + \frac{\Delta w}{\Delta y} - rw = 0. \tag{2.17}$$

Here the range of the log of the asset prices can be portioned into finite numbers of intervals where the minimum asset price is zero and the maximum is infinity. The value of the asset price will be $0, \Delta y, 2\Delta y, ..., y - 2\Delta, y - \Delta, y, y + \Delta y, y + 2\Delta, ..., \infty$ where the Δy is desired to be as small as possible and the maximum asset price will be finite. As $\ln 0$ is undefined, the minimum value of $\ln S$ is kept close to zero (ε). Letting $\tau = T - t$, the total span of the option's life is portioned into discrete intervals equaling $\tau, \tau - \Delta t, \tau - 2\Delta t, ..., 2\Delta t, \Delta t, 0$. These gradations of time and asset price can be shown on a grid. Each dot in the grid represents an option price of the corresponding log of the asset price in the given row and the corresponding expiry time in the given column.

For zero asset price, the call is worthless regardless of the expiry date and hence, one boundary condition in space is:

$$c(0, t) = 0 \quad \text{for all } t. \tag{2.18}$$

For column $\ln S$, $\ln S = \epsilon$ where ϵ is infinitesimally small, in which case:

$$w(\epsilon, t) = 0 \quad \text{for all } t. \tag{2.19}$$

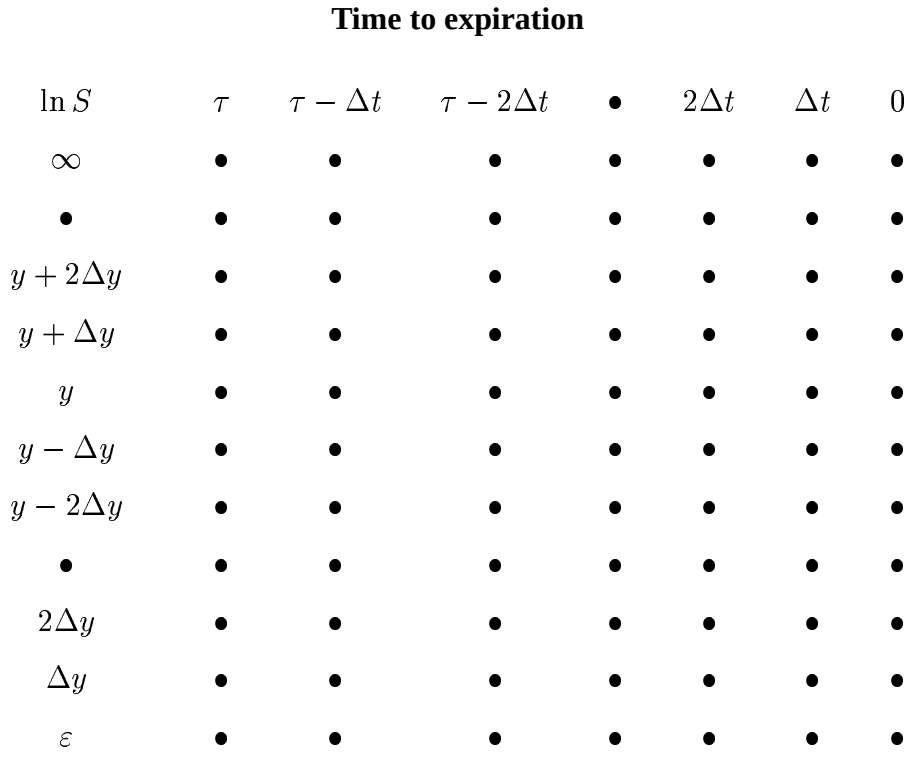

Figure 2.4: Time and Asset Price Grid

which allows us to fill the bottom rows of the grid shown in figure 2.4 with zeros [Cha97]. When $S \to \infty$, the first derivative of the call price (that is equal to asset price) with respect to the asset price is 1:

$$\lim_{S \to \infty} \frac{\partial c(S,t)}{\partial S} = 1 \quad \text{for all } t. \tag{2.20}$$

Since $\partial c(S,t)/\partial S = (\partial w/\partial y)e^{-y}$, it follows that:

$$\frac{\partial w(y,t)}{\partial y} = \frac{\partial c(S,t)}{\partial S} e^{y} = e^{y} = S \quad \text{for all } t \text{ when } \ln S \to \infty. \tag{2.21}$$

This means that if the second highest call value is known, the highest value can be determined by adding $\Delta y e^{y}$ to it.

The intrinsic value (the value of an option if it would expire with the underlying asset at its current price) at the expiry date is given as

$$c(S, 0) = \max(0, S - X) \quad \text{for all } S. \tag{2.22}$$

with respect to column y, this will be:

$$w(y, 0) = \max(0, e^y - X) \quad \text{for all } y. \tag{2.23}$$

So we can fill the right column and some portion of the grid will look like figure 2.5. Each expression under the dot denotes the price of a particular option with the $\log$ of the

$$\begin{array}{ccc}
\bullet & \bullet & \bullet \\
(w(y + \Delta y, t - \Delta t)) & (w(y + \Delta y, t)) & (w(y + \Delta y, t + \Delta t)) \\
\bullet & \bullet & \bullet \\
(w(y, t - \Delta t)) & (w(y, t)) & (w(y, t + \Delta t)) \\
\bullet & \bullet & \bullet \\
(w(y - \Delta y, t - \Delta t)) & (w(y - \Delta y, t)) & (w(y - \Delta y, t + \Delta t))
\end{array}$$

Figure 2.5: A Stencil of the Grid

asset at the level indicated and the time to expiration as shown in the grid.

The computation can proceed either explicitly or implicitly. Implicit method could lead to higher accuracy of the option values, but at a higher computational cost. These two approaches are explained in the following two sections. As can be seen from figure 2.4, the computational domain could be easily partitioned among several processors for concurrent computations, and hence the option valuations could be sped up.

The Explicit Finite Difference Method

Like with the binomial lattice technique, we adopt the backward marching idea to the current date from the future (expiration date). That is, this technique calculates $w(y,t)$ with respect to the values of $w(y+\Delta y, t+\Delta t), w(y, t+\Delta t)$. Shown in figure 2.6, are the option prices one time step forward and one asset price up from the current asset price and one asset price down. The three prices one time step ahead in figure 2.6 are known due to backward stepping in time.

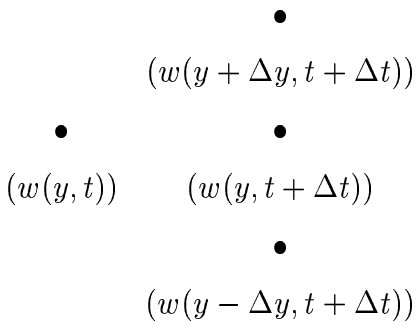

Figure 2.6: Stencil for Explicit Finite Difference Formulation

The Implicit Finite Difference Method

The implicit finite difference technique computes option prices from the grid in a different way. In figure 2.7, $w(y,t)$ is calculated from the option prices $w(y+\Delta y, t), w(y-\Delta y, t)$ in the same column. The calculation of option prices is obtained by solving the option prices in the same column simultaneously which could lead to higher accuracy, but it requires solving simultaneous equations and hence has higher computational cost.

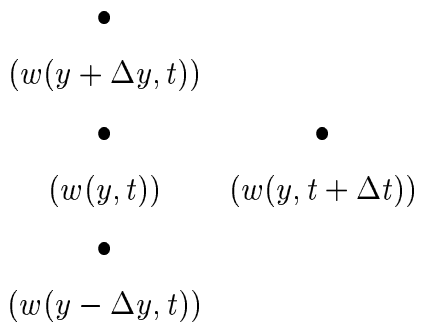

Figure 2.7: Stencil for Implicit Finite Difference Formulation

2.4 Fourier Analysis and Option Pricing

Some authors have reported on the use of Fourier analysis to calculate option prices [Bat97, CS92, GC97, Hes93, Sco97]. They worked to obtain analytical solution of pricing option for single or multiple assets using the FFT. Option value calculation in these approaches assumes the option to be risk-neutral. The drawback of the aforementioned approaches is that it does not take the advantage of the computational power of the fast Fourier transform. Besides, the mathematical model of the FFT for option valuation proposed by most authors lacks to obtain a numerical solution in parallel computing perspective. Carr and Madan [CM99] proposed two methods to determine option values numerically using the FFT and inverse Fourier transform. If the call value function is considered to be a function of the $\log$ of its strike, the Fourier transform of the call value can not be determined as this function is not square integrable (a function $f(x)$ is square integrable if $\int_{\infty}^{\infty} |f(x)|^2 dx$ is finite). In their first approach, the call pricing function is multiplied by an exponential dampening factor to make it square integrable and in the second approach the intrinsic value is subtracted from the call price function to obtain the time value (which, before the expiration, is the difference between intrinsic value and

the value of an option). The modified call price function now becomes square integrable and then the Fourier transform of the call price function can be obtained numerically when the characteristic function (described in section 2.4.1) of the risk neutral density is known analytically. FFT is applied numerically to invert the modified call pricing function and desired call value is obtained.

The first approach performs better than the second approach. The first approach requires careful selection of the dampening coefficient, whereas the second approach is robust and used when stability is the preference. In section 2.4.1, we present a review of the Fourier analysis as it is used in option pricing. We will then improve the first approach so as to make it amenable for parallel computing. We then discuss the improved model of Carr and Madan's first approach for use of FFT in parallel to calculate the option values described in chapter 4.

2.4.1 Review of Fourier Analysis in Option Pricing

In this section, we state how most authors have applied Fourier analysis to determine option prices. Consider a European call option of an underlying asset whose terminal spot price is S_T with maturity T. The characteristic function of s_T ($\equiv \ln S_T$) is defined as:

$$\phi_T(u) \equiv E[e^{ius_T}], \tag{2.24}$$

where E is the expectation. Assuming this characteristics function is known analytically, Bakshi and Madan [BM97] and Scott [Sco97] calculated the risk neutral probability of

finishing in-the-money[1] as:

$$Pr(S_T > K) = \Pi_2 = \frac{1}{2} + \frac{1}{\pi}\int_0^\infty Re\Big[\frac{e^{-iuk\phi_T(u)}}{iu}\Big]du, \tag{2.25}$$

where $k = \ln K$ is the log of the strike price. The delta of the option is numerically calculated as:

$$\Pi_1 = \frac{1}{2} + \frac{1}{\pi}\int_0^\infty Re\Big[\frac{e^{-iuk\phi_T(u-i)}}{iu\phi_T(-i)}\Big]du. \tag{2.26}$$

Considering a constant risk less rate r and no dividends, the option value is then calculated as:

$$C = S\Pi_1 - Ke^{-rT}\Pi_2. \tag{2.27}$$

But in this method, the FFT can not be applied to evaluate the integral due to the restriction of the integrand to its real part. Further discussion of FFT for option pricing as developed by Carr and Madan [CM99], is presented separately in chapter 3 since one of our aims is to improve on their model. Use of the Fourier technique for another special option, as proposed by Dempster and Hong, is described in the next section.

2.4.2 Spread Option Valuation and Fast Fourier Transform

In this section, the use of an FFT model for spread option is described based on Carr and Madan's [CM99] approach. A *Spread option* is a derivative which has the terminal pay-off equal to the difference between the prices of two assets and a fixed exercise (strike) price. S_1, S_2 are two underlying processes referring to asset or futures prices, equity indices or bond yields forming the spread. The pay-off can then be calculated as

[1] *in-the-money* call option is a situation where the underlying asset price of the option is larger than the strike price; *at-the-money* call means the asset price equals the strike price; a natural extension is for *out-of-the-money* call, which corresponds to a situation where the asset price is smaller than the strike price. These definitions are reversed for a put option.

$[(S_1(T) - S_2(T)) - K]_+$ where K is the strike price. Here '$+$' means only the positive result matters; if the result is negative then the pay-off is zero.

Other work using FFT for option pricing has been done by Dempster and Hong [DH00]. They proposed the use of fast Fourier transform for pricing generic spread options beyond the classical two-factor (deterministic and nondeterministic parts) Black-Scholes framework. They extended Carr and Madan's FFT technique to the multi factor setting so this method can be applied to price more than one asset when the joint characteristic function of the underlying assets forming the spread can be determined analytically. This method also allows the incorporation of stochasticity in the volatility and correlation structure by adding some other factors. Dempster and Hong's investigation shows that computational time does not increase significantly for considering additional random factors as the fast Fourier transform remains two dimensional for the two prices defining the spread. They also showed that the FFT technique for the spread option gives better performance over other traditional pricing methods such as Monte Carlo and PDE techniques.

Moreover, Dempster and Hong introduced a new technique for pricing spread options for a class of models such as the Variance Gamma (VG) model [MCC98], the inverse Gaussian model [BN97]numerous stochastic where the analytical characteristic functions for the underlying asset prices or market rates are known. They extended Carr and Madan's approach to option pricing to the multi factor setting for options where the pay-off is more complex than the piecewise-linear structure of single asset.

Review of the FFT Method

Dempster and Hong derived the value of a correlation of an option defined in [BM00] using the approaches and notation of Madan, et al. [MCC98] for deriving European call option on a single asset.

If S_1 and S_2 are two underlying asset prices, a *correlation option* is a two-factor analog of a European option where the pay-off is calculated as $[S_1(T)-K_1]\cdot[S_2(T)-K_2]$ at maturity T. If K_1, K_2, S_1, S_2 are the strike and asset prices (and k_1, k_2, s_1, s_2 - their logarithms) then the value of the option can be calculated as:

$$\begin{aligned} C_T(k_1,k_2) &:= E_Q\Big[e^{-rT}[S_1(T)-K_1]_+\cdot[S_2(T)-K_2]_+\Big] \\ &\equiv \int_{k_1}^{\infty}\int_{k_2}^{\infty} e^{-rT}(e^{s_1}-e^{k_1})(e^{s_2}-e^{k_2})q_T(s_1,s_2)ds_2ds_1. \end{aligned} \quad (2.28)$$

In equation (2.28), Q is the risk-neutral measure and $q_T(s_1,s_2)$ is the corresponding joint density of $s_1(T)$ and $s_2(T)$.

The characteristic function of this density is defined by the following equations:

$$\begin{aligned} \phi(u_1,u_2) &:= E_Q\Big[exp(iu_1s_1(T)+iu_2s_2(T))\Big] \\ &\equiv \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{i(u_1s_1+u_2s_2)}q_T(s_1,s_2)ds_2ds_1. \end{aligned} \quad (2.29)$$

The integral on the right hand side of equation (2.28) is not square integrable. So $C_T(k_1,k_2)$ is multiplied with a decaying term so that the integration will be integrable in k_1, k_2 over the negative axes:

$$c_T(k_1,k_2) := e^{\alpha_1k_1+\alpha_2k_2}C_T(k_1,k_2) \quad \alpha_1,\alpha_2>0. \quad (2.30)$$

The Fourier transform of this modified option price is:

$$\begin{aligned} \psi_T(v_1,v_2) &:= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{i(v_1k_1+v_2k_2)}c_T(k_1,k_2)dk_2dk_1 \\ &= \int\int_{R^2} e^{(\alpha_1+iv_1)k_1+(\alpha_2+iv_2)k_2}\int_{k_2}^{\infty}\int_{k_1}^{\infty} e^{-rT}(e^{s_1}-e^{k_1})(e^{s_2}-e^{k_2})q_T(s_1,s_2)ds_2ds_1dk_2dk_1 \\ &= \int\int_{R^2} e^{-rT}q_T(s_1,s_2)\times \\ &\quad \int_{\infty}^{s_2}\int_{\infty}^{s_1} e^{-rT}(e^{s_1}-e^{k_1})(e^{s_2}-e^{k_2})e^{(\alpha_1+iv_1)k_1+(\alpha_2+iv_2)k_2}dk_2dk_1ds_2ds_1 \\ &= \int\int_{R^2}\frac{e^{-rT}q_T(s_1,s_2)e^{(\alpha_1+1+iv_1)s_1+(\alpha_2+1+iv_2)s_2}}{(\alpha_1+iv_1)(\alpha_1+1+iv_1)(\alpha_2+iv_2)(\alpha_2+1+iv_2)}ds_2ds_1 \\ &= \frac{e^{-rT}\phi_T(v_1-(\alpha_1+1)i,v_2-(\alpha_2+1)i)}{(\alpha_1+iv_1)(\alpha_1+1+iv_1)(\alpha_2+iv_2)(\alpha_2+1+iv_2)}. \end{aligned} \quad (2.31)$$

The Fourier transform ψ_T of the option price can be calculated analytically if the characteristics function ϕ_T is known in closed form. From the inverse Fourier transform of equation (2.31), the option price can be calculated using:

$$C_T(k_1,k_2)=\frac{e^{-\alpha_1 k_1-\alpha_2 k_2}}{(2\pi)^2}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}e^{-i(v_1k_1+v_2k_2)}\psi_T(v_1,v_2)dv_2dv_1. \tag{2.32}$$

Using the trapezoidal rule to approximate the integral in equation (2.32), we have:

$$C_T(k_1,k_2)=\frac{e^{-\alpha_1 k_1-\alpha_2 k_2}}{(2\pi)^2}\sum_{m=0}^{N-1}\sum_{n=0}^{N-1}e^{-i(v_{1,m}k_1+v_{2,n}k_2)}\psi_T(v_{1,m},v_{2,n})\Delta_2\Delta_1, \tag{2.33}$$

where $\Delta_1\Delta_2$ are the integration steps and

$$v_{1,m}=\left(m-\frac{N}{2}\right)\Delta_1 \quad v_{2,n}=\left(n-\frac{N}{2}\right)\Delta_2 \quad m,n=0,...,N-1. \tag{2.34}$$

If $\{X[j_1,j_2]\in C|j_1=0,...,N_1-1,j_2=0,...,N_2-1\}$ is a two-dimensional complex input array, the Fourier transform of this input array is

$$Y[l_1,l_2]:=\sum_{j_1=0}^{N_1-1}\sum_{j_2=0}^{N_2-1}e^{-\frac{2\pi i}{N_1}j_1l_1-\frac{2\pi i}{N_2}j_2l_2}X[j_1,j_2], \quad \text{for all } l_1=0,...,N_1-1,l_2=0,...,N_2-1. \tag{2.35}$$

If we define a grid size of $N\times N,\Lambda:=\{(k_{1,p},k_{2,q}):0\le p,q\le N-1\}$ where

$$k_{1,p}:=\left(p-\frac{N}{2}\right)\lambda_1, k_{2,q}:=\left(q-\frac{N}{2}\right)\lambda_2 \tag{2.36}$$

Equation (2.33) can be evaluated as:

$$\Gamma(k_1,k_2):=\sum_{m=0}^{N-1}\sum_{n=0}^{N-1}e^{-i(v_{1,m}k_1+v_{2,n}k_2)}\psi_T(v_{1,m},v_{2,n}) \tag{2.37}$$

Putting $\lambda_1\Delta_1=\lambda_2\Delta_2=\frac{2\pi}{N}$ gives the following values of $\Gamma(k_1,k_2)$ on Λ:

$$\begin{aligned}\Gamma(k_{1,p},k_{2,q}) &= \sum_{m=0}^{N-1}\sum_{n=0}^{N-1}e^{-i(v_{1,m}k_{1,p}+v_{2,n}k_{2,q})}\psi_T(v_{1,m},v_{2,n})\\ &= \sum_{m=0}^{N-1}\sum_{n=0}^{N-1}e^{-\frac{2\pi i}{N}\left[(m-N/2)(p-N/2)+(n-N/2)(q-N/2)\right]}\psi_T(v_{1,m},v_{2,n})\\ &= (-1)^{p+q}\sum_{m=0}^{N-1}\sum_{n=0}^{N-1}e^{-\frac{2\pi i}{N}(mp+nq)\left[(-1)^{m+n}\psi_T(v_{1,m},v_{2,n})\right]}.\end{aligned} \tag{2.38}$$

Now equation (2.38) can be computed by the fast Fourier transform equation (2.35) where the two dimensional input array will be:

$$X[m,n] = (-1)^{m+n}\psi_T(v_{1,m}, v_{2,n}), \quad \forall m,n = 0,...,N-1. \tag{2.39}$$

For $N \times N$ different ($\log$) strikes, the option price can be calculated approximately as:

$$C_T(k_{1,p}, k_{2,q}) \approx \frac{e^{-\alpha_1 k_{1,p} - \alpha_2 k_{2,q}}}{(2\pi)^2}\Gamma(k_{1,p}, k_{2,q})\Delta_2\Delta_1 \quad 0 \leq p,q \leq N. \tag{2.40}$$

Chapter 3

FFT for Option Pricing - The CM Model

3.1 Background

The solution for the optimal exercise policy must typically be performed numerically, and is usually a computationally intensive problem. To price an American option, the binomial tree approach [CRR79] has been used extensively. Recently, the option pricing problem has been studied using the Fast Fourier Transform (FFT) [CM99, DH00]. Introducing a one-to-one mapping Fourier analysis to the computational domain of the Cooley-Tukey FFT algorithm [CLW77], [TT03] explored the use of high performance computing for this problem. Other research [TT01] showed that the FFT yields much better performance for the derivatives under study in comparison to the binomial lattice approach. Our goal in this thesis is to extend [TT03] the work done in two directions: improving the CM-FFT [CM99] model mathematically (chapter 4) and designing a new parallel FFT algorithm to improve the performance of its evaluation (chapter 5).

3.2 The CM Model

In this section, we describe the fundamental idea behind the use of FFT for option pricing. Further basic ideas on the use of FFT for pricing problems are described by Cerny [Cer04]. Financial engineers use Fourier analysis to identify cyclic patterns in asset price movements. Such processes can be either described in the *time domain* by *h*, which is a function of time *h(t)*, or in the *frequency domain* where the process is specified by giving its amplitude, *H*, as a function of frequency *f*, that is *H(f)*, with -$\infty < f < \infty$. One goes back and forth between the representations by means of the continuous Fourier transform equations $H(f) = \int_{-\infty}^{\infty} h(t)e^{2\pi ift}dt$ and $h(t) = \int_{-\infty}^{\infty} H(f)e^{-2\pi ift}df$ or their discretized forms given by:

$$H(f) = \frac{1}{N}\sum_{t=0}^{N-1} h(t)e^{2\pi ift/N}. \tag{3.1}$$

and

$$h(t) = \frac{1}{N}\sum_{t=0}^{N-1} H(f)e^{-2\pi ift/N}. \tag{3.2}$$

Since call value is a function of strike price, by approximately mapping call value and strike price to the above equations, we can apply the Fourier transform to the option pricing problem.

We write the call price function given by Carr and Madan [CM99] as,

$$C_T(k) = \frac{exp(-\alpha k)}{\pi}\int_0^{\infty} e^{-ivk}\psi_T(v)dv, \tag{3.3}$$

where $\psi_T(v)$ is the Fourier transform of this call price (also given by [CM99]):

$$\begin{aligned}\psi_T(v) &= \int_{-\infty}^{\infty} e^{ivk}c_T(k)dk \\ &= \int_{-\infty}^{\infty} e^{ivk}e^{-rT}\int_k^{\infty} e^{\alpha k}(e^s - e^k)q_T(s)dsdk. \end{aligned}\tag{3.4}$$

$$\psi_T(v) = \frac{e^{-rT}\phi_T(v-(\alpha+1)i)}{\alpha^2+\alpha-v^2+i(2\alpha+1)v}. \tag{3.5}$$

$\psi_T(v)$ is odd in its imaginary part and even in its real part. Here k is the log strike price K ($k = \log(K)$) and is identical to t in equation (3.2) [TT03]. That is, the call option price needs to be computed at various strike prices of the underlying assets in the option contract. Further, v corresponds to f, $\psi_T(v)$ is the Fourier transform of the call price $C_T(k)$ and q_T(s) is the risk-neutral density function of the pricing model. The integral in the right hand side of equation (3.3) is a direct Fourier transform and lends itself to the application of the FFT in the form of summation given by equations (3.1) and (3.2).

If $M = e^{-\alpha k}/\pi$ and $\omega = e^{-i}$ then

$$C_T(k) = M \int_0^\infty \omega^{vk} \psi_T(v) dv. \tag{3.6}$$

If $v_j = \eta(j-1)$ and the trapezoid rule is applied for the integral on the right of equation (3.6), $C_T(k)$ can be written as

$$C_T(k) \approx M \sum_{j=1}^{N} \psi_T(v_j) \omega^{v_j k} \eta, k = 1, \ldots, N, \tag{3.7}$$

where the effective upper limit of integration is $N\eta$ and v_j corresponds to various prices with η spacing.

Therefore, in line with Carr and Madan [CM99], one can solve the option pricing problem using Fourier analysis. For further details on the fundamentals of FFT please refer to Cormen et al. [CLRS01] and for details on parallel FFT refer to Grama et al. [GGKK03].

3.3 Drawback of the CM Model

To calculate the call values, equation (3.6) has to be solved analytically. The discrete form of the above equation given as equation (3.7) is not suitable to feed into the exiting

FFT algorithms such as Cooley-Tukey [CLW77]. Hence the CM model in its current form cannot be used for faster pricing. This is a major drawback of using CM model for practical purposes.

To obtain numerical solution and to take the advantage of parallel computing for real time pricing we need to improve this mathematical model. Without loss of generality, we will relax some of the parametric conditions and introduce modification so that the improved mathematical model will generate feasible and tractable input values for the FFT algorithm which will calculate accurate call values. We will then parallelize the algorithm.

This leads us to state the objectives of this thesis as:

- Improving the mathematical model which will provide accurate solutions that will be tractable for parallel computing,
- Designing an efficient parallel FFT algorithm which can map the mathematics to the computational domain from the improved model and implementing the algorithm on distributed memory architecture and study the performance. In parallel algorithm two types of latency are incurred - synchronization latency and communication latency. FFT is inherently a synchronous algorithm. We have designed a new parallel FFT algorithm which will exploit data locality by bringing data closer to the local processor before computation so that the communication latency will be reduced.

Chapter 4

Mathematical Improvement to the CM-FFT Option Pricing Model

Most recent research on option valuation has successfully applied Fourier analysis to calculate option prices. As shown earlier in equation (3.3), to obtain the analytically solvable Fourier transform, the call price function needs to be multiplied by an exponential factor, $e^{\alpha k}$ ($c_T(k) = e^{\alpha k} C_T(k)$). The calculation of $\psi_T(v)$ in equation (3.5) depends on the factor $\phi_T(u)$, where $u = v - (\alpha + 1)i$. The calculation of the intermediate function $\phi_T(u)$ requires specification of the risk neutral density function, $q_T(s)$. The limits on the integral have to be selected in such a way as to generate real values for the FFT inputs. To generate the closed form expression of the integral, the integrands, especially the function $q_T(s)$ which is the risk neutral density function of the terminal $\log$ price s_T, have to be selected appropriately. Without loss of generality, we use uniform distribution for $q_T(s)$. This implies occurrence of a range of terminal log prices at equal probability, which could, of course, be relaxed and a normal or more sophisticated distribution could be employed. Since the volatility is assumed (low) the variation in the drift is expected

to cause a stiffness in the system. However, since we have assumed uniform distribution for $q_T(s)$, variation in the drift is eliminated and hence the stiffness is avoided.

For numerical calculation purposes, the upper limit of equation (3.6) is assumed as a constant value and the lower limit is assumed to be zero. The upper limit will be dictated based on the terminal spot price. In other words, to finish the call option in-the-money, the upper limit will have to be smaller than the terminal asset price. Therefore, the equation is:

$$\phi_T(u) = \int_0^{\lambda} e^{ivk} q_T(s) ds = \int_0^{\lambda} (\cos(vk) + i\sin(vk)) q_T(s) ds. \tag{4.1}$$

Without loss of generality, modifications are required as derived below to essentially achieve the mapping from Fourier space to FFT computational domain. In other words, the purpose of these modifications is to generate feasible and tractable initial input condition to the FFT from these equations. These modifications make the implementation easier. From equation (3.4),

$$\begin{aligned}
\psi_T(v) &= \int_{-\alpha}^{\alpha} e^{ivk} e^{-rT} \int_k^{\alpha} e^{\alpha k}(e^s - e^k) q_T(s) ds dk & (4.2)\\
&= \frac{e^{-rT}\Phi_T(v-(\alpha+1)i)}{\alpha^2+\alpha-v^2+i(2\alpha+1)v} \\
&= \frac{e^{-rT}\Phi_T(v-(\alpha+1)i)((\alpha^2+\alpha-v^2)-i(2\alpha+1)v)}{((\alpha^2+\alpha-v^2)+i(2\alpha+1)v)((\alpha^2+\alpha-v^2)-i(2\alpha+1)v)} \\
&= \frac{e^{-rT}\Phi_T(v-(\alpha+1)i)((\alpha^2+\alpha-v^2)-i(2\alpha+1)v)}{((\alpha^2+\alpha-v^2)^2+(2\alpha+1)^2v^2)}. & (4.3)
\end{aligned}$$

Now,

$$\Phi_T(u) = \int_0^{\lambda} e^{ius} q_T(s) ds. \tag{4.4}$$

where λ is the $\log$ of terminal spot price and integration is taken only in the positive axis. To calculate $\phi_T(v-(\alpha+1)i)$, $v-(\alpha+1)i$ is substituted for u in equation (4.4) which gives:

$$\phi_T(v-(\alpha+1)i) = \int_0^{\lambda} e^{(iv+\alpha+1)s} q_T(s) ds. \tag{4.5}$$

Assuming $q_T(s)$ as a uniform distribution function of the terminal $\log$ price, equation (4.5) can be shown as:

$$\begin{aligned}
\phi_T(v-(\alpha+1)i) &= q_T(s)\frac{e^{(iv+\alpha+1)s}}{iv+\alpha+1}\Big|_0^\lambda \\
&= \frac{q_T(s)}{iv+\alpha+1}(e^{iv\lambda}e^{(\alpha+1)\lambda}-1) \\
&= \frac{q_T(s)(\alpha+1-iv)}{(\alpha+1)^2+v^2}(e^{(\alpha+1)\lambda}(\cos(\lambda v)+i\sin(\lambda v))-1) \\
&= \frac{q_T(s)}{(\alpha+1)^2+v^2}\Big[[e^{(\alpha+1)\lambda}\{(\alpha+1)\cos(\lambda v)+v\sin(\lambda v)\}-(\alpha+1)] \\
&+ i[e^{(\alpha+1)\lambda}\{(\alpha+1)\sin(\lambda v)-v\cos(\lambda v)\}+v]\Big]. \qquad (4.6)
\end{aligned}$$

if we assume $e^{(\alpha+1)\lambda}\{(\alpha+1)\cos(\lambda v)+v\sin(\lambda v)\}-(\alpha+1)=\Delta$ and $e^{(\alpha+1)\lambda}\{(\alpha+1)\sin(\lambda v)-v\cos(\lambda v)\}+v=\Delta_x$ then equation (4.6) can be written as:

$$\phi_T(v-(\alpha+1)i)=\frac{q_T(s)}{(\alpha+1)^2+v^2}(\Delta+i\Delta_x). \qquad (4.7)$$

Substituting equation (4.7) in equation (4.3) gives the following (denoted as BTT-CM):

$$\psi_T(v)=\frac{e^{-rT}q_T(s)}{\{(\alpha+1)^2+v^2\}\{(\alpha^2+\alpha-v^2)^2+(2\alpha+1)^2v^2\}}\times$$
$$\Big[\{(\alpha^2+\alpha-v^2)\Delta+(2\alpha+1)v\Delta_x\}+i\{(\alpha^2+\alpha-v^2)\Delta_x-(2\alpha+1)v\Delta\}\Big]. \qquad (4.8)$$

The expression above is used for the new parallel FFT algorithm (chapter 5) to compute the call price function. The financial input data set for our parallel FFT algorithm is the calculated data points of $\psi_T(v)$ for different values of v. We call equation (4.8) as BTT-CM equation or BTT-CM model.

We then calculate call value for different strike price values v_j where j will range from 1 to N. The lower limit of the strike price is zero and the upper limit is $(N-1)\eta$ where η is the spacing in the line of integration. Smaller values of η give fine grid

integration and a smooth characteristic function of strike price and the corresponding calculated call value.

The value of k on the left side of equation (3.7) represents the $\log$ of the ratio of strike to terminal spot price. The implementation of the FFT mathematical model returns N values of k with a spacing size of γ and these values are fed into a parallel algorithm to calculate N values of $C_T(k)$. Here we consider cases in the range of in-the-money to at-the-money call values. The value of k will be 0 for at-the-money call - that is where the strike price and the exercise price are equal. The FFT yields N values of k and with a regular spacing of size γ, the values for k can be obtained from the following equation:

$$k_u = -p + \gamma(u-1), \text{ for } u = 1, \ldots, N. \tag{4.9}$$

Hence, the $\log$ of the ratio of strike to exercise price will range from $-p$ to p where $p = \frac{N\gamma}{2}$. Substituting equation (4.9) in equation (3.7) gives:

$$C_T(k_u) \approx \frac{exp(-\alpha k_u)}{\pi} \sum_{j=1}^{N} e^{-iv_j(-p+\gamma(u-1))} \psi_T(v_j)\eta, \text{ for } u = 1, \ldots, N. \tag{4.10}$$

Replacing v_j with $(j-1)\eta$ in equation (4.10), we get

$$C_T(k_u) \approx \frac{exp(-\alpha k_u)}{\pi} \sum_{j=1}^{N} e^{-i\gamma\eta(j-1)(u-1))} e^{ipv_j} \psi_T(v_j)\eta, \text{ for } u = 1, \ldots, N. \tag{4.11}$$

The basic equation of the FFT is

$$Y(k) = \sum_{j=1}^{N-1} e^{-i\frac{2\pi}{N}(j-1)(k-1)} x(j), \text{ for } k = 1, \ldots, N. \tag{4.12}$$

Comparing equation (4.10) with the basic FFT equation, we can easily say that equation (4.10) is also a FFT equation. To apply the FFT, we also note that $\gamma\eta = \frac{2\pi}{N}$. Smaller values of η will ensure fine grid for the integration. But call prices at relatively large strike spacings (γ), few strike prices will lie in the desired region near the stock price [CM99]. Furthermore, if we increase the values of N, we will get more intermediate points of the

calculated call prices ($C_T(k_u)$) corresponding to different strike prices (v_j). This helps the investor to capture the call price movements of an option for different strike prices in the market. In the experimental result (section 6.2.1) of 1024 (N) numbers of calculated call values, assuming $\eta = 0.25$ with the intuition that it will ensure fine grid integration, γ is calculated as 0.02454. Similar to basic FFT equation, equation (4.10) can also be parallelized. An efficient parallel FFT algorithm can compute fast and accurate solution of equation (4.10).

The improvement to the CM model described in this section and given in the final form of equation (4.8) fulfills the first objective (section 3.3) of this thesis. This improved model generates the input data sets for our parallel FFT algorithm (described in next chapter) to compute the call values.

Chapter 5

A New Parallel FFT Algorithm for Option Pricing

In this chapter, first we describe the parallelization of the basic FFT equation for the sake of completeness. Then we introduce our new data swap algorithm followed by the actual pseudocode.

5.1 Parallelization of the FFT Equation

A sequence X of length n is $X = \langle X[0], X[1].....X[n-1] \rangle$. The discrete Fourier transform (DFT) of this sequence X is a sequence $Y = \langle Y[0], Y[1].....Y[n-1] \rangle$ of the same length, where

$$Y[i] = \sum_{k=0}^{n-1} X[k]\omega^{ki}\,, 0 \leq i < n. \tag{5.1}$$

with ω as the primitive n^{th} root of unity in the complex plane; that is $\omega = e^{-2\pi\sqrt{-1}/n}$. In FFT computations, the powers of ω are called twiddle factors. Each calculation of

$Y[i]$ needs n multiplications of complex numbers and n additions. Thus, the sequential complexity using the straight forward algorithm for calculating n values of Y in the sequence is $\Theta(n^2)$. The Fast Fourier Transform algorithm described below reduces the complexity to $\Theta(n \log n)$.

The following steps [GGKK03] defines the basics of the FFT algorithm. Assuming n is a power of two, an n-point DFT calculation can be converted into two $(n/2)$-point DFT computations, as follows:

$$\begin{aligned} Y[i] &= \sum_{k=0}^{(n/2)-1} X[2k]\omega^{2ki} + \sum_{k=0}^{(n/2)-1} X[2k+1]\omega^{(2k+1)i} \\ Y[i] &= \sum_{k=0}^{(n/2)-1} X[2k]e^{2(-2\pi\sqrt{-1}/n)ki} + \sum_{k=0}^{(n/2)-1} X[2k+1]\omega^{i}e^{2(-2\pi\sqrt{-1}/n)ki} \\ Y[i] &= \sum_{k=0}^{(n/2)-1} X[2k]e^{-2\pi\sqrt{-1}ki/(n/2)} + \omega^{i}\sum_{k=0}^{(n/2)-1} X[2k+1]e^{-2\pi\sqrt{-1}ki/(n/2)}. \end{aligned} \tag{5.2}$$

Let $\tilde{\omega} = e^{-2\pi\sqrt{-1}/(n/2)} = \omega^2$; that is, $\tilde{\omega}$ is the primitive $(n/2)$nd root of unity. Then, we can rewrite equation (5.2) as follows:

$$Y[i] = \sum_{k=0}^{(n/2)-1} X[2k]\tilde{\omega}^{ki} + \omega^{i}\sum_{k=0}^{(n/2)-1} X[2k+1]\tilde{\omega}^{ki}. \tag{5.3}$$

Each summation on the right-hand side of equation (5.3) is an $(n/2)$-point DFT computation. At each level, each DFT computation can be split into two smaller computations in a recursive manner. This leads to the development of Fast Fourier Transform (FFT) algorithm. The FFT algorithm takes the advantage of the special properties of the complex root of unity and it employs divide-and-conquer strategy [CLRS01]. If n is a power of two, the maximum number of levels for the FFT computation of a series of X of length n is $\log n$. At each level $\Theta(n)$ operation will be done. Therefore, the overall sequential complexity will be $\Theta(n \log n)$.

Currently parallel FFT algorithm designers have given importance to block data distribution only, instead of the parallel architecture on which the algorithm is implemented or the interconnection network among the processors. The most commonly used algorithms for FFT computation are the Cooley-Tukey [CLW77] algorithm and the Gentleman-Sande [GS66] algorithm. These approaches differ only in their communication patterns [TT03]. Two approaches can be applied to computing the FFT algorithm: recursive and iterative schemes. The recursive approach can be easily implemented on a shared memory architecture and the iterative approach can be easily implemented on distributed architectures where every processor has its own local memory and data is exchanged among them using message passing. In such a distributed memory architecture, if the communication time can be somehow reduced, it would be highly beneficial.

The FFT calculation can be parallelized. The mathematical model for the FFT computation can be thought of as an algebraic function that works on data-vectors. A fine-grained distribution of loops that will implement the FFT functions at each level will result in effective parallelization.

5.2 A New Data Swap FFT Algorithm

For general FFT computations, one of the most notable algorithm was proposed by Cooley-Tukey [CLW77]. Figure 5.1 illustrates the data transfers performed by the parallel Cooley-Tukey algorithm and figure 5.2 illustrates the underlying *butterfly* computation. Let us assume we have N ($N = 2^m$) data elements and P ($P = 2^p$) processors where $N > P$. A butterfly computation is performed on each of the data points in every iteration. The butterfly computation can be conceptually described as follows: *a* and *b* are two points (i.e. real or complex numbers). The upper part of the butterfly operation computes the summation of *a* and *b* with a twiddle factor ω while the lower part computes

the difference. In each iteration, there are $\frac{N}{2}$ summations and $\frac{N}{2}$ differences calculated on processors.

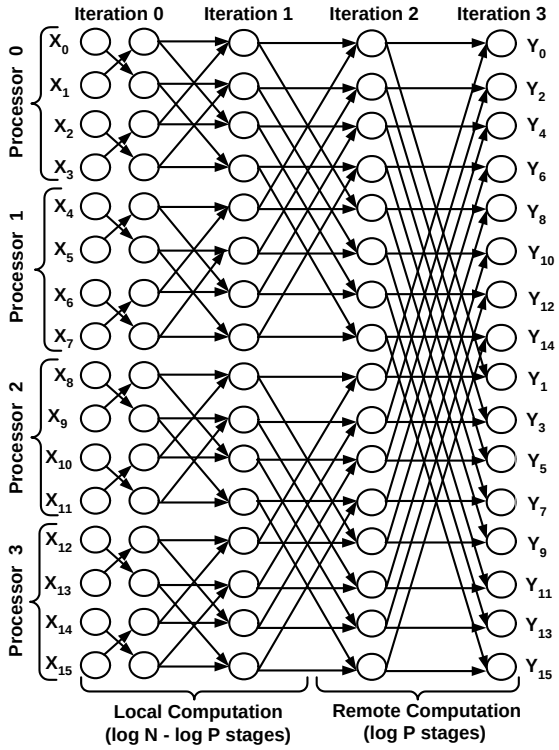

Figure 5.1: Cooley-Tukey Algorithm

The FFT is inherently a synchronous algorithm [GGKK03]. In general, a parallel algorithm for FFT with blocked data distribution [GGKK03] where $\frac{N}{P}$ data points are allocated to every processor involves communication for $\log P$ iterations and terminates after $\log N$ iterations. Each butterfly computation requires two data points which is a subset of the data-vectors. As mentioned in section 5.1, the total number of stages required to finish the FFT computation of N data points is $\log N$. The butterfly computation among N/P local data points in the same processor requires no communication. The number of stages required to finish the butterfly computations among the data points residing in the same processor is $\log N/P$, that is $\log N - \log P$. The rest $\log P$ $(\log N - (\log N - \log P))$

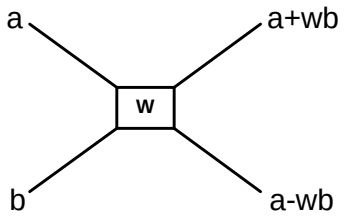

Figure 5.2: Butterfly Computation

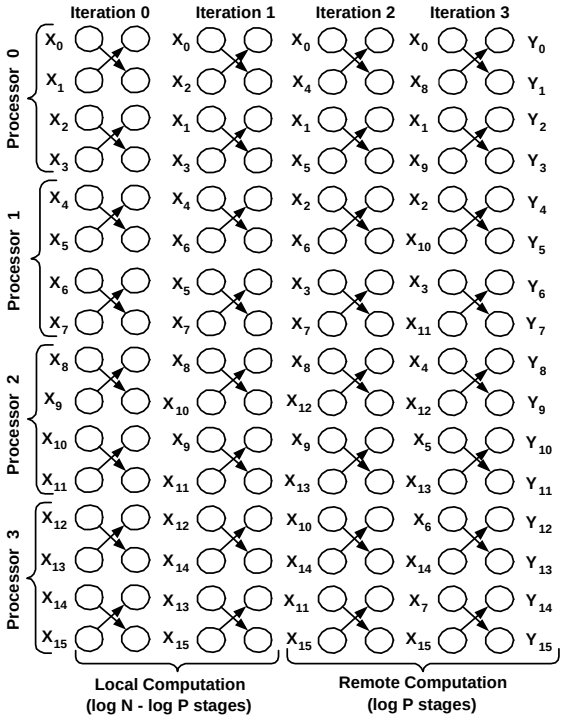

Figure 5.3: Data Swap Algorithm

number of stages requires communications among processors to collect the partner data point of each butterfly computation. The input data points are bit reversed before feeding them to the parallel FFT algorithm. If we assume shuffled input data at the beginning, the first ($\log N - \log P$) stages require no communication. That is, the data required for the butterfly computation, resides in each local processor. Therefore, during the first ($\log N - \log P$) iterations, a sequential FFT algorithm can be used inside each processor (called the local algorithm).

At the end of the $(\log N - \log P)th$ iteration, the latest computed values for $\frac{N}{P}$ data points exist in each processor. The last $\log P$ stages require remote communications (called the remote algorithm). The partners of each of the $\frac{N}{P}$ data points in processor i (P_i) required to perform the butterfly computation at each iteration reside in a different processor j (P_j) where $P_j(bin) = P_i(bin) \oplus 2^{k - \log \frac{N}{P}}$ (k is the current stage number where $k = 0 \ldots (\log N - 1)$, and $\oplus$ is the *eXclusive OR* operation). In a blocked data distribution, therefore, $\frac{N}{P}$ data items are communicated by each processor for $\log P$ stages. The message size in each case is $\frac{N}{P}$.

From figure 5.1, we can see that calculating Y_0 in processor 0 requires two data points, one of which resides in the local processor (0), and the other of which resides in processor 2. Thus, one communication is required to calculate Y_0. Similarly, calculating Y_1, Y_2, and Y_3 need 3 more communications with processor 2. Each processor requires four communications to calculate 4 FFT outputs. In total, sixteen communications are therefore required.

In our new data swap algorithm, depicted in figure 5.3, we apply the same blocked data distribution and the first ($\log N - \log P$) stages still require no communication. However, in the last $\log P$ stages that require communication, some data are swapped at each stage and the data continue to reside in each processor's local memory after swapping. Therefore, the identity of some of the data points stored in each processor

changes at each of the $\log P$ stages.

In figure 5.3, we can see that calculating the first two output data points in processor 0 requires two input data points with index 0 and 8 and data point with index 8 does not reside in the local processor. So we need one communication to bring data point 8 from processor 2. Similarly, calculating the next two output data points need one more communication. Therefore, in processor 0, we need two communications (not four) to calculate four output data points. Applying the same argument, each of the processors 1, 2, and 3 needs only two communications. In total, eight communications are required to calculate FFT of 16 data points. So in the new parallel FFT algorithm, the number of communications is reduced by half. The data points involved in calculating Y_0 at the different stages of figure 5.1 are the same nodes required to calculate Y_0 in figure 5.3 which proves that the calculated output values from both algorithm will be same and consistent.

We take advantage of the the fact that communication between processors is point to point and swap the data in a similar manner. However, in this case, only $\frac{N}{2P}$ data items are communicated by each processor at every stage. It is worth noting that the data swapping between processors at each location allows both the upper and lower part of the butterfly computations to be performed locally by each processor. This improvement enhances data locality and thereby provides performance increase in the new FFT algorithm compared to the Cooley-Tukey algorithm.

If the inputs for the butterfly computation are complex numbers where the real and imaginary parts are also high precision floating point numbers, then the calculated output values will also be high precision complex numbers. Taking the approximation of the calculated output for each butterfly computation at each stage of the parallel FFT computation may deviate from the accurate result when the number of communications increases with the increase in total number of data points.

5.3 Pseudo Code

Pseudo code for the mathematical model:

```
Begin
1. Initialize parameters {S, T, r, lambda, N,
   alpha, eta, delta, delta_x} for equation
   (4.7) and (4.10) in chapter 4

2. Compute the constants
      uni_distribution = 1/S
      log_terminalspot = log10(asset_price)
      p = (N * lambda)/2
      e_term = exp((alpha + 1)*log_terminalspot)
      nominator = exp(-r * T) * uni_distribution;

4. Computer N values of the real (fft_input.real) and
   imaginary parts (fft_input.imaginary) used as
   inputs to the FFT equation (4.11) in chapter 4

      for j = 1 to N Do
        Vj = eta * (j-1)
        delta = e_term * ((alpha+1) * cos(log_terminalspot
        * Vj) + Vj * sin(log_terminalspot * Vj)) - (alpha
        + 1)

        delta_x = e_term * ((alpha + 1) *
```

```
        sin(log_terminalspot *  Vj) -  Vj * cos(log_terminalspot
        * Vj)) +  Vj;

        denominator = (sqr(alpha + 1) + sqr(Vj)) *
        (sqr((sqr(alpha) + alpha - sqr(Vj))) +
        sqr(2 * alpha + 1) * sqr(Vj))

        psi_T_Vj.real = (nominator * ((sqr(alpha)
        + alpha - sqr(Vj)) * delta + (2 * alpha + 1)
        * Vj * delta_x ))/denominator

        psi_T_Vj.imaginary = (nominator * ((sqr(alpha)
        + alpha - sqr(Vj)) * delta_x  - (2 * alpha + 1)
        * Vj * delta))/denominator

        fft_input.imaginary = eta * (psi_T_Vj.imaginary
        * cos(p * Vj) + psi_T_Vj.real * sin(p * Vj))

        fft_input.real = eta * (psi_T_Vj.real * cos(p
        * Vj) - psi_T_Vj.imaginary * sin(p * Vj))
End
```

The N numbers of the FFT input calculated at step 4 are then fed into our new data swap algorithm.

Pseudo code for the Data Swap Algorithm:

```
Begin
1. localsize=TotalDataSize/NumofProcessors /* assumed to
     devide evenly*/

2. The FFT input data calculated are bit
reversed in the master processor.

3. The master processor distributes the data
evenly among all other (worker) processors and
RealLocalA and ImaginaryLocalA arrays contain the real
imaginary parts of each of N/P local data points.

4. Calculate log(TotalDataSize) and log(ProcessorNumber)

5. for i = 1 to log(TotalDataSize)
 If we are in the first log N - log P stages
 /* local algorithm*/
   p = 0
   for j = 0 to localsize - 1
     if ((j & pow(2, i)) == 0)
       RealLocalTempA[p] = RealLocalA[j ^ pow(2, i)]
       ImaginaryLocalTempA[p] = ImaginaryLocalA[j ^ pow(2, i)]
     else
       RealLocalTempA[p] = RealLocalA[(j + 1) ^ pow(2, i)]
       ImaginaryLocalTempA[p] = ImaginaryLocalA[(j + 1) ^ pow(2, i)]
```

```
    p = p + 1
    j = j + 2
    /*Appropiate data are available in the local processor.
      Now we will do the butterfly computation*/
    for j = 0 to localsize - 1
      if ((j & pow(2, i)) == 0)
        do butterfly calculation with the data points
        stored in RealLocalTempA[p], RealLocalA[j] and
        ImaginaryLocalA[j], ImaginaryLocalTempA[p]
      else
        do butterfly calculation with the data points
        stored in RealLocalTempA[p], RealLocalA[j + 1]
        and ImaginaryLocalA[j + 1], ImaginaryLocalTempA[p]
    p = p + 1
    j = j + 2
  /*If the data is not residant in the local processor
  for the last log P stages, we have to communicate
  with other processors to find the data that are needed
  for the butterfly computation*/
else
  if ((MyRank & pow(2, i - localindex)) == 0)
     dest = MyRank ^ pow(2, i - localindex)
     send all data points with odd number index
     from RealLocalA and ImaginaryLocalA arrays
  else
     source = MyRank ^ pow(2, i - localindex)
```

```
      receive data points from the data sending
      processor and stored in RealLocalTempA and
      ImaginaryLocalTempA arrays

   if ((MyRank & pow(2, i-localindex)) != 0)
      dest = MyRank ^ pow(2, i - localindex)
      send all data points with even number index
      from RealLocalA and ImaginaryLocalA arrays
   else
      source = MyRank ^ pow(2, i - localindex)
      receive data points from the data sending
      processor and stored in RealLocalTempA
      and ImaginaryLocalTempA arrays

 /*Now we have the right data in all local processors
   so we can now do the butterfly computations locally*/
   p = 0
   for j = 0 to j < localsize - 1
      if ((MyRank & pow(2, i - localindex)) == 0)
       do butterfly calculation with the data points
       stored in RealLocalTempA[p], RealLocalA[j] and
ImaginaryLocalA[j], ImaginaryLocalTempA[p]
      else
        do butterfly calculation with the data points
        stored in RealLocalTempA[p], RealLocalA[j + 1]
        and ImaginaryLocalA[j + 1], ImaginaryLocalTempA[p]
```

```
        p = p + 1
        j = j + 2
End
```

Chapter 6

Results

In this section, we present the analytical results first followed by the experimental results.

6.1 Analytical Results

The first $(\log N - \log P)$ iterations are local computations requiring no communication. Each processor computes the butterfly computation on its own $\frac{N}{P}$ local data items. However, the next $\log P$ iterations require remote communications among the processors. At a particular iteration i of the $\log P$ iterations, $\frac{N}{P}$ data points are exchanged in the Cooley-Tukey algorithm (figure 5.1) and $\frac{N}{2P}$ data points are swapped (communicated) in our swap algorithm (figure 5.3) for each processor.

Let the *startup time* (t_s), be the time required for a processor to prepare a message (adding header, trailer, and error correction information). Then, for $\log P$ iterations, $t_s \log P$ time is required. Let's assume the startup times for the Cooley-Tukey algorithm and swap algorithm are $t_s^{'}$ and $t_s{''}$ respectively where $t_s^{'} > t_s{''}$ since each processor in the Cooley-Tukey algorithm exchanges twice as much data as the swap algorithm. If

the channel bandwidth is r words per second, then the *per-word transfer time,* $t_w = \frac{1}{r}$, is required for a word or data point to traverse the link between processors. So the total transfer time is $t_w \frac{N}{P} \log P$ for $\log P$ number of interprocessor communications for the Cooley-Tukey algorithm and $t_w \frac{N}{2P} \log P$ interprocessor communications for the swap algorithm. Once the data is swapped, each processor performs the upper and lower portion of the butterfly computations on its local data. Assume each butterfly computation requires time t_c. Then the total computational cost (ignoring communication cost) for the whole FFT algorithm is $t_c \frac{N}{P} \log N$.

Therefore, the *parallel runtime,* $T_{Cooley-Tukey}$ and T_{Swap} of the Cooley-Tukey algorithm and the swap algorithm respectively are as follows where T_{comp} is the computation time and T_{comm} is the communication time:

$$\begin{aligned} T_{Cooley-Tukey} &= T_{comp} + T_{comm} \\ &= t_c \frac{N}{P} \log N + t_s^{'} \log P + t_w \frac{N}{P} \log P \qquad (6.1) \\ &\quad \text{for the Cooley-Tukey algorithm.} \end{aligned}$$

$$\begin{aligned} T_{Swap} &= T_{comp} + T_{comm} \\ &= t_c \frac{N}{P} \log N + t_s^{''} \log P + t_w \frac{N}{2P} \log P \qquad \text{for the Swap algorithm.} \quad (6.2) \end{aligned}$$

Using $T_{Cooley-Tukey}$ and T_{Swap}, the speedup and efficiency calculations of both the algorithms are as follows:

Speedup Comparison

The expression for the speedup for both algorithms are shown below:

$$S_{Cooley-Tukey} = \frac{T_{serial(Cooley-Tukey)}}{T_{parallel(Cooley-Tukey)}}$$

$$
\begin{aligned}
&= \frac{t_c N \log N}{t_c \frac{N}{P} \log N + t_s' \log P + t_w \frac{N}{P} \log P} \\
&= \frac{P t_c N \log N}{t_c N \log N + P t_s' \log P + t_w N \log P} \\
&= \frac{P T_c}{t_c N \log N + (\log P)(P t_s' + t_w N)} \quad \text{where } T_c = t_c N \log N \\
&= \frac{P T_c}{T_c + (\log P) C_b} \quad \text{where } C_b = P t_s' + t_w N. \qquad (6.3)
\end{aligned}
$$

$$
\begin{aligned}
S_{Swap} &= \frac{T_{serial(Swap)}}{T_{parallel(Swap)}} \\
&= \frac{t_c N \log N}{t_c \frac{N}{P} \log N + t_s'' \log P + t_w \frac{N}{2P} \log P} \\
&= \frac{P t_c N \log N}{t_c N \log N + P t_s'' \log P + t_w \frac{N}{2} \log P} \\
&= \frac{P T_c}{t_c N \log N + (\log P)(P t_s'' + t_w \frac{N}{2})} \quad \text{where } T_c = t_c N \log N \\
&= \frac{P T_c}{T_c + (\log P) C_s} \quad \text{where } C_s = P t_s'' + t_w \frac{N}{2}. \qquad (6.4)
\end{aligned}
$$

$$
\begin{aligned}
C_b &= P t_s' + t_w N \\
&= P(t_s'' + \delta) + t_w \frac{N}{2} + t_w \frac{N}{2} \qquad \text{as } t_s' > t_s'' \\
&= P t_s'' + t_w \frac{N}{2} + P\delta + t_w \frac{N}{2} \\
&= C_s + P\delta + t_w \frac{N}{2}.
\end{aligned}
$$

Therefore,

$$C_b > C_s. \qquad (6.5)$$

Applying condition (6.5) to equations (6.3) and (6.4), gives

$$S_{Cooley-Tukey} < S_{Swap}. \qquad (6.6)$$

Hence, the swap algorithm produces a better speedup performance than the Cooley-Tukey algorithm as will be shown in the experimental results section.

Efficiency Comparison

The expressions for efficiency for both algorithms are shown below

$$\begin{aligned} E_{Cooley-Tukey} &= \frac{S_{Cooley-Tukey}}{P} \\ &= \frac{T_c}{T_c + (\log P)C_b}. \end{aligned} \tag{6.7}$$

$$\begin{aligned} E_{Swap} &= \frac{S_{Swap}}{P} \\ &= \frac{T_c}{T_c + (\log P)C_s}. \end{aligned} \tag{6.8}$$

Using condition (6.5) in equations (6.7) and (6.8) we can say

$$E_{Cooley-Tukey} < E_{Swap}. \tag{6.9}$$

6.2 Experimental Results

Section 6.2.1 shows the calculated option values using our new data swap algorithm and in section 6.2.2, we study the performance of our algorithm with respect to the parallel Cooley-Tukey algorithm.

6.2.1 Call Value

Figure 6.1 shows how the data swap algorithm calculates the call values from the input data set generated from the BTT-CM equation. As mentioned in chapter 4, the BTT-CM

equation calculates $\psi_T(v_j)$ for different values of v_j (strike price), where $v_j = \eta(j-1)$ and $j = 1 \ldots N$. η is the strike price spacing and for a fixed strike price range (for example 0-300) larger values of N gives smaller values of η. With smaller strike price spacing, the data swap algorithm calculates more numbers of intermediate call values in the specified region of the strike price. This depicts the change in the call price for smaller change in the strike price. In section 6.2.2, we observe that our data swap algorithm performs better over the Cooley-Tukey algorithm when the data sizes (N) increases. Therefore, the data swap algorithm can capture the call prices for various strike prices faster than the Cooley-Tukey algorithm.

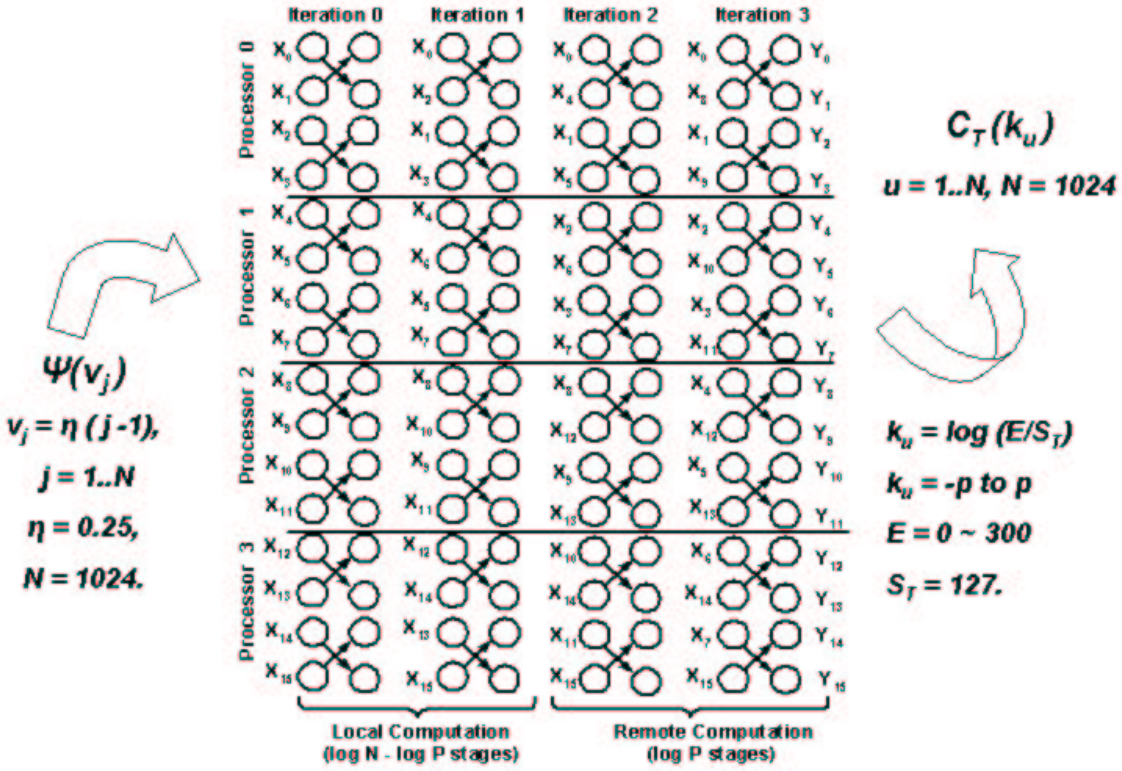

Figure 6.1: Input and Output to the Data Swap Algorithm

The data swap algorithm calculates N number of call values ($C_T(k_u)$ where k_u is the

$\log$ of the ratio of the strike price E and the terminal spot price S_T (market price of the underlying asset). The call values obtained from the data swap algorithm are normalized values with respect to the terminal spot price. In other words, call values for a given strike price, which is the $\log$ of the ratio of the strike and terminal spot price, are normalized with terminal spot price as a base. When the $strike\ price < spot\ price$ the call option is in-the-money (the value of k_u will be negative) and when strike and spot price are equal the call option is at-the-money (the value of k_u will be zero). Whereas the call option is out-of-the-money (the value of k_u will be positive), if $strike\ price > spot\ price$. When the call option is in-the-money, the investor would prefer to exercise the option (purchasing the option) at the strike price and immediately sell the asset in the market at the terminal spot price. Thus, the holder can profit. When the call option is at-the-money, the profit or loss is zero. But for the call option out-of-the-money, the investor would not prefer to exercise it as the spot price of the asset in the market is less than the exercise price (strike price) of the contract.

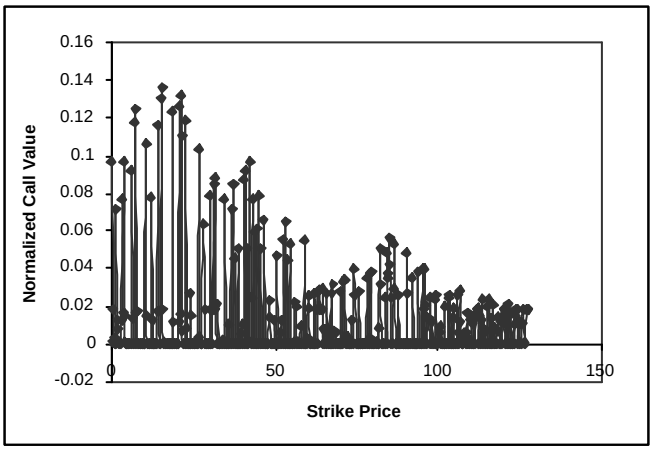

Figure 6.2: Computed Call Values

Figure 6.2 depicts the calculated in-the-money call values for different strike prices

using the data swap algorithm. In the experiment of call value computation, strike price can be any value between 0 and 300. Our data swap algorithm can calculate (figure 6.1) call values for in-the-money, at-the-money and out-of-the-money call options. We are considering in-the-money call where the terminal spot price is always greater than the strike price. Therefore, figure 6.2 plots a portion of the calculated call values (in-the-money) from the output values of the data swap algorithm.

For this particular experiment, (with $\eta = 0.25$, $\gamma = 0.02454$, and $N = 1024$) the terminal spot price is 127 and to calculate the in-the-money call, the strike price ranges from 0 to 150. The plot shows that the normalized option value is decreasing with the increase of strike price. If X is the strike price and S_T is the terminal spot price of the underlying asset, the European call value is $\max(S_T - X, 0)$. With the increase of strike price from 0 toward 127, $(S_T - X)$ is expected to decrease, which can be seen in figure 6.2. For larger values of N we can get more number of call values computed for the strike price range from 0 to 127, which makes the plot as a continuous function.

6.2.2 Performance Results

The experiments on the performance study of the data swap algorithm were conducted on a 20 node SunFire 6800 high performance computing system running MPI at the University of Manitoba . The Sunfire consists of Ultra Sparc III CPUs, with 1050 MHz clock rate and 40 gigabytes of cumulative shared memory runs the Solaris 8 operating system. The data generated by the BTT-CM equation in chapter 4 are used for the FFT input.

Figure 6.3 and table 6.1 show the execution time, for different numbers of processors of the FFT Cooley-Tukey Algorithm. As the data size increases, there is a decrease in execution time. For a data size of 2^{20} on 16 processors, there is $\log 2^{20} - \log 2^4 = 16$ local

computations and 4 remote computations. At each iteration $\frac{N}{2P} = \frac{2^{20}}{2^5} = 2^{15}$ data points are swapped on 16 processors. On a two processor machine, there are $\log 2^{20} - \log 2 = 19$ local computations and only 1 remote communication. However, there is a significant decrease in execution time using 16 processors. This is attributed to the fact that in MPI, the packing and unpacking of $\frac{N}{2P} = 2^{18}$ data elements for 2 processors a requires significant amount of time.

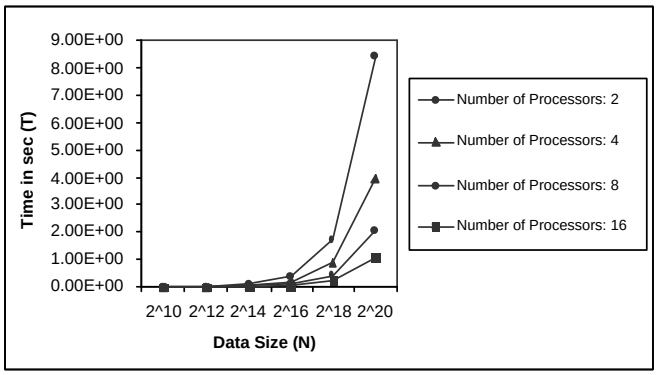

Figure 6.3: Execution Time of the Swap Algorithm for Various Problem Sizes

Data Size \ Number of Processors	2	4	8	16
2^{10}	3.16E-03	1.95E-03	1.33E-03	4.40E-03
2^{12}	1.86E-02	1.00E-02	8.53E-03	6.80E-03
2^{14}	7.74E-02	3.83E-02	2.33E-02	1.53E-02
2^{16}	3.69E-01	1.84E-01	9.42E-02	5.44E-02
2^{18}	1.74E+00	8.60E-01	4.38E-01	2.53E-01
2^{20}	8.40E+00	4.01E+00	2.02E+00	1.08E+00

Table 6.1: Execution Time of the Swap Algorithm for Various Problem Sizes

When we compare the swap algorithm to the Cooley-Tukey Algorithm in figure 6.4 and in table 6.2 on 16 processors, the swap algorithm performs 15% better than Cooley-Tukey Algorithm with a data size of 2^{20}. This corroborates the analytical results in equation (6.1) and equation (6.2), where $T_{swap} < T_{Cooley-Tukey}$.

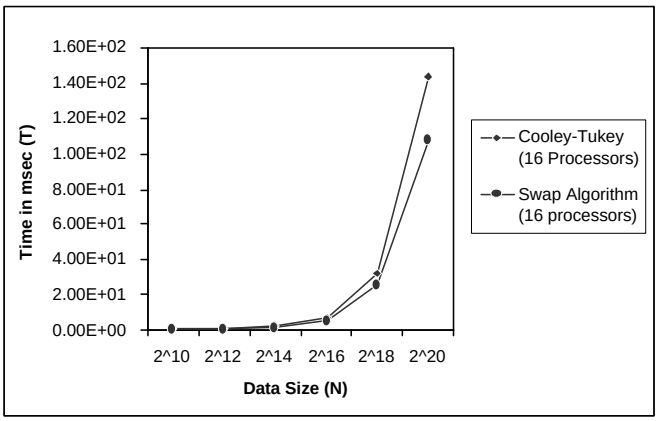

Figure 6.4: Comparison of the Execution Time of the Swap and Cooley-Tukey Algorithms

Data Size / 16 Processors	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}
Cooley-Tukey	5.13E-01	6.86E-01	2.54E+00	7.20E+00	3.22E+01	1.44E+02
Swap	4.40E-01	6.80E-01	1.53E+00	5.44E+00	2.53E+01	1.08E+02

Table 6.2: Comparison of the Execution Time of the Swap and Cooley-Tukey Algorithms

The speedup on 16 processors for a data size of 2^{19} is approximately 15 as shown in figure 6.5. As the data size decreases, however, the speedup on 16 processors is about 5. When $N = 2^{12}$, $t_w 2^9$ data items are swapped while for $N = 2^{16}$, $t_w 2^{13}$ items

swapped. From equation (6.4), since the speedup is proportional to N, the speedup on 16 processors for a larger data size produces better results.

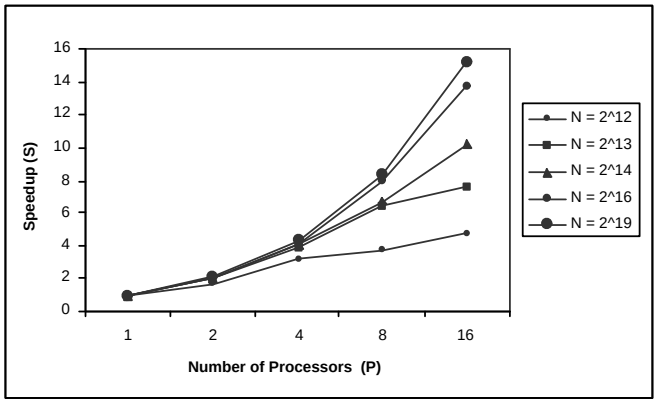

Figure 6.5: Speedup of the Swap Algorithm with respect to Number of Processors

Data Size \ Number of Processors	1	2	4	8	16
2^{12}	1	1.71E+00	3.18E+00	3.72E+00	4.68E+00
2^{13}	1	2.02E+00	3.86E+00	6.40E+00	7.62E+00
2^{14}	1	2.03E+00	4.09E+00	6.71E+00	1.02E+01
2^{16}	1	2.02E+00	4.05E+00	7.93E+00	1.38E+01
2^{19}	1	2.14E+00	4.29E+00	8.35E+00	1.52E+01

Table 6.3: Speedup of the Swap Algorithm with respect to Number of Processors

From the speedup, we calculated the efficiency of the swap algorithm for various processors on a fixed data size as presented in figure 6.6 and table 6.4. The efficiency for 16 processors is close to 1. For 4, 8, and 16 processors the efficiency is 90% for data sizes of 2^{14}, 2^{16}, 2^{19} respectively. Also for 8 and 16 processors the efficiency is 50% for sizes

of 2^{12} and 2^{13} respectively. These results illustrate that as the data size and the number of processors are increased, the swap algorithm exhibits good scalability.

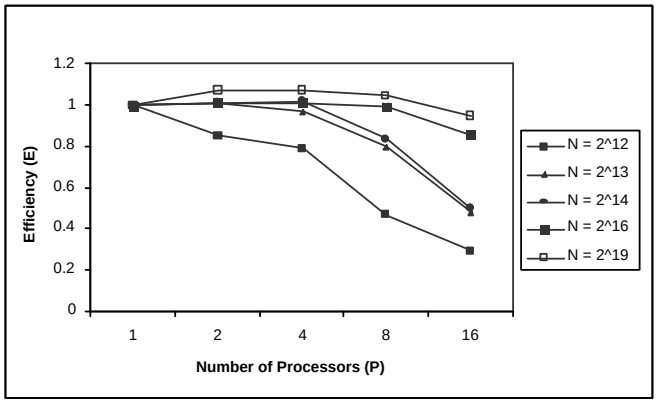

Figure 6.6: Efficiency of the Swap Algorithm

Data Size \ Number of Processors	1	2	4	8	16
2^{12}	1	8.54E-01	7.94E-01	4.66E-01	2.92E-01
2^{13}	1	1.01E+00	9.66E-01	8.00E-01	4.77E-01
2^{14}	1	1.01E+00	1.02E+00	8.39E-01	4.96E-01
2^{16}	1	1.01E+00	1.01E+00	9.92E-01	8.59E-01
2^{19}	1	1.07E+00	1.07E+00	1.04E+00	9.50E-01

Table 6.4: Efficiency of the Swap Algorithm

Finally, figure 6.7 and table 6.5 compares the speedup of both the swap and Cooley-Tukey Algorithms. The speedup of the swap algorithm for data sizes 2^{16} and 2^{19} for large number of processors produces better results than the Cooley-Tukey Algorithm. The experimental results correlates with the analytical results in equation (6.6).

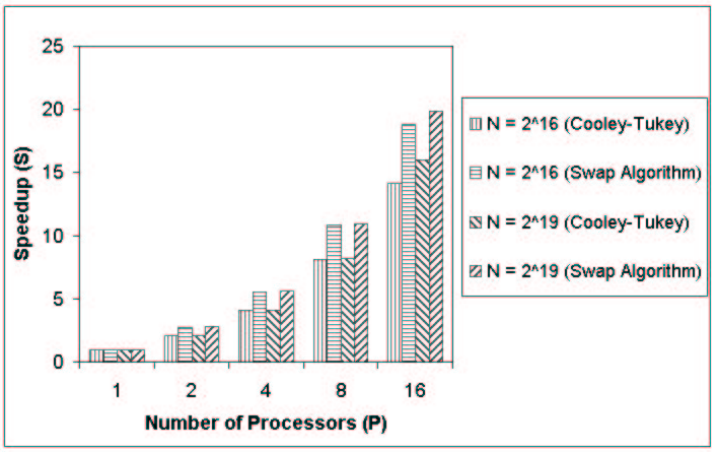

Figure 6.7: Comparison of the Speedup between the Cooley-Tukey and Swap Algorithms

In the Sunfire at the University of Manitoba, MPI is built in the system which gives fast interconnections. In the Cooley-Tukey algorithm, the numbers of communications are more than that in the data swap algorithm. As we have noted, the data swap algorithm performs better on this machine. We have also experimented both the algorithms on a cluster with non-dedicated interconnections (a cluster in the Department of Computer Science, the University of Manitoba, commonly known as “gull” cluster). Due to the lack of dedicated interconnection, as the number of communications increases the total time required for the communications is much higher for the Cooley-Tukey algorithm. Therefore, on the non-dedicated cluster, the speed increase using the data swap algorithm over the Cooley-Tukey algorithm is far pronounced than that on the sunfire system and hence we are not reporting our results from the gull cluster.

Data Size / Number of Processors	Cooley-Tukey 2^{16}	Swap 2^{16}	Cooley-Tukey 2^{19}	Swap 2^{19}
1	1	1	1	1
2	2.05E+00	2.77E+00	2.06E+00	2.79E+00
4	4.11E+00	5.54E+00	4.14E+00	5.60E+00
8	8.09E+00	1.08E+01	8.23E+00	1.09E+01
16	1.42E+01	1.88E+01	1.60E+01	1.98E+01

Table 6.5: Comparison of the Speedup between the Cooley-Tukey and Swap Algorithms

Chapter 7

Conclusions

Our research yields two significant results that comprise this thesis. As mentioned in chapter 3, the scope of this thesis is to continue one of the earlier works [TT03] in two directions: improving the study of the option pricing problem using the FFT mathematically and computationally. This work has provided new insights into the mathematics of FFT for option pricing problems. Our BTT-CM model improves the mathematical model of the Fourier transform for option pricing facilitating tractability for parallel computing, providing efficient and accurate solutions, and improving the mathematical model so as to design an efficient FFT algorithm. Feeding the results generated from the BTT-CM model into the FFT algorithm facilitated a smooth transition in using the FFT algorithm for computing call values. We also exploited the principle of data locality in the butterfly network of the Cooley-Tukey algorithm to reduce the communication latency to obtain better performance.

Next, we implemented the new parallel algorithm on a distributed memory machine and studied the performance improvement attained. We have shown analytically how our new parallel FFT algorithm performs better than the Cooley-Tukey algorithm due to

reduced communication time achieved by swapping data points before computations to improve data locality. Compared to the traditional Cooley-Tukey algorithm, the current algorithm with data swapping performs better by more than 15% for large data sizes as confirmed by experimentation.

In summary, without loss of generality, we have first identified appropriate values for the parameters and generated the input data set for the FFT computation using the modified mathematical model (BTT-CM equation). A basic parallel implementation of the FFT on a distributed platform, using MPI for message passing, was carried out first. The communication latency was reduced by improving data locality, a main challenge in developing a parallel FFT algorithm. We have integrated the mathematical model to the new FFT algorithm and studied the performance results. These conclusion are reported in two publications [BTT04a, BTT04b].

In the rapidly changing market place, these improvements could mean a lot for an investor or financial institution.

Chapter 8

Future Work

Immediate future work based on the presented research would be to test the scalability of the new FFT algorithm on a cluster with more processors. This is not currently possible at the University of Manitoba due to the absence of such cluster. We are trying to access some national facilities to do this.

Our study is based on some assumptions concerning volatility (constant) and interest rate. The FFT approach is more flexible than other techniques because with the change of the parametric conditions it does not change the computational time significantly. Moreover, in the FFT technique, one can add more realistic factors such as stochastic volatilities and stochastic interest rates to represent a more realistic characteristics of market dynamics. These assumptions could be relaxed with some effort as another immediate example of future work.

To avoid the communication latency inherent in the distributed environments for the use of FFT in real time option pricing, a real challenge is in implementing the algorithm on shared memory and distributed shared memory architectures. In shared memory architectures the communication latency is less pronounced depending on the design of

the algorithm and hence, we can expect better performance. However, synchronization remains an issue for FFT implementation on shared memory architectures. From our investigation so far we find that the following are a few of the likely challenges for the implementation of the algorithm on shared memory machines:

1. Thread creation, thread granularity, thread boundary are some of the important issues needed to be considered for the implementation on shared memory architecture using OpenMP.

2. Developing a performance model in a shared address space is more difficult than in a message passing interface. Implicit naming, replication, and coherence make modeling performance more complicated, so performance will be quite dependent on:

   - how the synchronization of the butterfly computation at each stage of the FFT computation can be achieved using semaphore and critical sections?
   - performance study will have to consider not only the number of processors and problem size but also number of threads, context switch time and overhead, and data dependencies;
   - how workload over processors varies with the number of threads given a fixed data size and a fixed number of processors?
   - finding the size of the thread (fine, medium or coarse grained) that will fit best with the FFT computation for the option pricing problem. Spawning and synchronizing threads incurs a fair amount of overhead, So finding the optimal number of threads for the parallel FFT computation depends on the problem size as well as the number of available processors;

The parallel FFT algorithm that we have implemented for a distributed memory architecture can also be implemented for a distributed shared memory architecture.

Bibliography

[Amm89] H.M. Amman. *Supercomputing in Economics: A New Field of Research?* High Performance Computing, (Ed: J.L. Delhaye, E. Gelenbe) Elsevier Science Publishers B.V. (North-Holland), 1989.

[Bat97] D. Bates. Jumps and Stochastic Volatility: Exchange Rate Processes Implicit in the PHLX Deutschemark Options. *Mathematical Finance*, 7:413–426, 1997.

[BM97] G. Bakshi and D. Madan. A Simplified Approach to the Valuation of Options. memo, University of Maryland, 1997.

[BM00] G. Bakshi and D. Madan. Spanning and Derivative Security Valuation. *Journal of Financial Economics*, 44(1):123–165, 2000.

[BN97] O. E. Barndorff-Nielsen. Processes of Normal Inverse Gaussian Type. *Finance and Stochastics*, 2(1):41–68, 1997.

[Boy77] P.P. Boyle. Options: A Monte Carlo Approach. *Journal of Financial Economics*, 4:323–338, May 1977.

[BS73] F. Black and M. Scholes. The Pricing of Options and Corporate Liabilities. *Journal of Political Economy*, 81:637–654, January 1973.

[BS77] M. Brennan and E. Schwartz. The Valuation of American Put Options. *Journal of Finance*, pages 323–338, 1977.

[BTT04a] S. Barua, R.K. Thulasiram, and P. Thulasiraman. Fast Fourier Transform for Option Pricing: Improved Mathematical Modeling and Design of Efficient Parallel Algorithm. In *Proceedings (CD-ROM) of the International Conference on Computational Science and its Applications*, pages 686–695, Assisi, Italy, May 14–17 2004.

[BTT04b] S. Barua, R.K. Thulasiram, and P. Thulasiraman. Improving Data Locality in Parallel Fast Fourier Transform Algorithm for Pricing Financial Derivatives. In *Proc. (CD-ROM) of Parallel and Distributed Scientific and Engineering Computing (PDSEC)*, pages 235–240, New Mexico, USA, April 26–30 2004.

[Cer04] A. Cerny. *Mathematical Techniques in Finance - Tools for Incomplete Markets- Chapter 7*. Princeton University Press, Princeton, NJ, 2004.

[Cha97] D.M. Chance. Option Pricing Using Finite Difference Methodsi - Revised October 2003, 1997. Technical note 97-02, Department of Finance, Louisiana State University.

[CJEV98] P. Chalasani, S.h Jha, F. Egriboyun, and A. Varikooty. A Refined Binomial Lattice for Pricing American Asian Options. In *Proceedings of the Eighth Annual Derivative Securities Conference, Boston (loose-bound volume - no page number)*, April 1998.

[Cla98] I.J. Clark. Option Pricing Algorithms for the Cray T3D Supercomputer. *Proceedings of the first National Conference on Computational and Quantitative Finance (loose-bound volume - no page number)*, September 1998.

[CLRS01] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*. McGraw Hill, Boston, MA, 2001.

[CLW77] J.W. Cooley, P.A. Lewis, and P.D. Welch. *The Fast Fourier Transform and its Application to Time Series Analysis*. Wiley, New York, 1977. In Statistical Methods for Digital Computers.

[CM99] P. Carr and D.B. Madan. Option Valuation using the Fast Fourier Transform. *The Journal of Computational Finance*, 2(4):61–73, 1999.

[CRR79] J.C. Cox, S.A. Ross, and M. Rubinstein. Option Pricing: A Simplified Approach. *Journal of Financial Economics*, 7:229–263, September 1979.

[CS92] R. Chen and L. O. Scott. Pricing Interest Rate Options in a Two-Factor Cox-Ingersoll-Ross Model of the Term Structure. *Review of Financial Studies*, 5:613–636, 1992.

[CTIT04] A. Chhabra, P. Thulasiraman, M.T. Islam, and R.K. Thulasiram. An OpenMP implementation of FTCS method for reduced Black-Scholes Equation . In *Proceedings of the International Symposium on High Performance*

Computing Systems and Applications (HPCS), pages 205–207, Winnipeg, MB, Canada, May 16-19 2004.

[DH00] M.A.H. Dempster and S.S.G Hong. Spread Option Valuation and the Fast Fourier Transform. Technical Report WP 26/2000, Judge Institute of Management Studies, Cambridge, England, 2000.

[DHC+04] A. J. Davies, M. E. Honnor, H. Lai C, A. K. Parrott, and S. Rout. A Distributed Laplace Transform Algorithm for European Options. In *Proceedings of the International Conference on Computational Finance and its Applications (ICCFA)*, pages 157–166, Southampton, UK, April 2004. WIT Press.

[FLM+01] M.C. Fu, S.B. Laprise, D. Madan, Y. Su, and R. Wu. Pricing American Options: A Comparison of Monte Carlo Simulation Approaches. *Journal of Computational Finance*, 2:61–73, 2001.

[GC97] B. Gurdip and Z. Chen. An Alternative Valuation Model for Contingent Claims. *Journal of Financial Economics*, 44(1):123–165, 1997.

[GGKK03] A. Grama, A. Gupta, V. Kumar, and G. Karypis. *Introduction to Parallel Computing*. Pearson Educarion Limited, Edinburgh Gate, Essex, 2 edition, 2003.

[GS66] W.M. Gentleman and G. Sande. Fast Fourier Transform for Fun and Profit. In *Proceedings 1966 Fall Joint Computer Conference AFIPS 29*, pages 563–578, 1966.

[Hes93] 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.

[HL01] M.B. Haugh and A.W. Lo. Computational Challenges in Portfolio Management - Tomorrow's Hardest Problem. *Computing in Science and Engineering*, 3(3):54–59, May-June 2001.

[Hul02] J.C. Hull. *Options, Futures and Other Derivatives*. Prentice Hall, Upper Saddle River, NJ, 5th edition, 2002.

[KNR00] E. J. Kontoghiorghes, A. Nagurnec, and Berc Rustem. Parallel Computing in Economics, Finance and Decision-making. *Parallel Computing*, 26:207–209, 2000.

[MCC98] D. Madan, P. Carr, and E. Chang. The Variance Gamma Process and Option Pricing. *European Finance Review*, 2(1):79–105, 1998.

[RST04] S. Rahmail, I. Shiller, and R.K. Thulasiram. Different Estimators of the Underlying Asset's Volatility and Option Pricing Errors: Parallel Monte-Carlo Simulation. In *Proc. International Conference on Computational Finance and its Applications*, pages 121–131, Bologna, Italy, April 2004.

[Sco97] L.O. Scott. Pricing Stock Options in a Jump-Diffusion Model with Stochastic Volatility and Interest Rates. *Mathematical Finance*, 7(4):413–426, 1997.

[Sri02] A. Srinivasan. Parallel and Distributed Computing Issues in Pricing Financial Derivatives Through Quasi Monte Carlo. In *Proc. (CD-ROM) International Parallel and Distributed Processing Symposium (IPDPS02)*, Fort Lauderdale, FL, April 2002.

[TB04] R.K. Thulasiram and D. Bondarenko. Performance Evaluation of Parallel Algorithms for Pricing Multidimensional Financial Derivatives. *International Journal of Computational Science and Engineering (to be published)*, 2004.

[TDG00] R.K. Thulasiram, C. Downing, and G.R. Gao. A Multithreaded Parallel algorithm for pricing American Securities. In *Proceedings (CD-ROM) of the Computational Finance 2000 Conference*, London, UK, May/June 2000.

[Thu03a] R.K. Thulasiram. Computational Finance Lecture Notes, January 2003. Topic 5:PDEs and Finite Differencing Technique, Department of Computer Science, University of Manitoba.

[Thu03b] R.K. Thulasiram. Computational Finance Lecture Notes, January 2003. Topic 7: Monte Carlo Simulations, Department of Computer Science, University of Manitoba.

[TKA+00] K.B. Theobald, R. Kumar, G. Agrawal, G. Heber, R.K. Thulasiram, and G.R. Gao. Developing a Communication Intensive Application on the EARTH Multithreaded Architecture. In *Lecture Notes in Computer Science, Vol. 1900, Proceedings of the European Parallel Computing Conference*, pages 625–637, Munich, Germany, Aug–Sep 2000.

[TLN+01] R.K. Thulasiram, L. Litov, H. Nojumi, C.T. Downing, and G.R. Gao. Multithreaded Algorithms for Pricing a Class of Complex Options. In *Proceedings (CD-ROM) of the International Parallel and Distributed Processing Symposium(IPDPS)*, San Francisco, CA, April 2001.

[TR00] D. Tavalla and C. Randall. *Pricing Financial Instruments: The Finite Difference Method*. John Wiley and Sons, New York, NY, 2000.

[TT01] R.K. Thulasiram and P. Thulasiraman. A Parallel FFT Approach for Option Pricing. In *Proceedings of the SPIE Intl. Symp. on Commercial Applications for High Performance Computing (with ITCOM'01; Ed. H.J. Siegel)*, pages 181–192, Denver, CO, August 2001.

[TT03] R.K. Thulasiram and P. Thulasiraman. Performance Evaluation of a Multithreaded Fast Fourier Transform Algorithm for Derivative Pricing. *The Journal of Supercomputing*, 26(1):43–58, August 2003.

[Tuc97] A.B. Tucker. *Computer Science and Engineering Handbook*. CRC Press, Boca Raton, Florida, 1997.

[TZG04] R.K. Thulasiram, C. Zhen, and A. Gumel. A Second Order L_0 Stable Algorithm for Evaluating European Options. In *Proceedings of the International Symposium on High Performance Computing Systems and Applications (HPCS)*, pages 17–23, Winnipeg, MB, Canada, May 2004.

[Var96] H.R. Varian. *Computational Economics and Finance: Modeling and Analysis With Mathematica*. Springer Verlag, New York, NY, September 1996. Chapter 15.

[WHD95] P. Wilmott, S. Howison, and J. Dewynne. *The Mathematics of Financial Derivatives, A Student Introduction*. Cambridge Univesrity Press, Cambridge, UK, 1995.

[Zen99] S.A. Zenios. High-Performance Computing in Finance: The Last 10 Years and the Next. *Parallel Computing*, 25:2149–2075, December 1999.

[ZFV98] R. Zvan, P. Forsyth, and K.R. Vetzal. A General Finite Element Approach for PDE Option Pricing Models. *Proceedings of the first National Conference on Computational and Quantitative Finance (loose-bound volume - no page number)*, September 1998.

[Zhe02] C. Zhen. Numerical and Dynamical Studies of Some Reactions-Diffusion Models. M.Sc. Thesis, Department of Mathematics, The University of Manitoba, 2002.