A Normalized Particle Swarm Optimization Algorithm to Price Complex Chooser Option and Accelerating its Performance with GPU

by

Bhanu Pratap Sharma

A thesis submitted to
The Faculty of Graduate Studies of
The University of Manitoba
in partial fulfillment of the requirements
of the degree of

Master of Science

Department of Computer Science
The University of Manitoba
Winnipeg, Manitoba, Canada
September 2011

Thesis advisor

Dr. Ruppa Thulasiram

Dr. Parimala Thulasiraman

Author

Bhanu Pratap Sharma

A Normalized Particle Swarm Optimization Algorithm to Price Complex Chooser Option and Accelerating its Performance with GPU

Abstract

An option is a financial instrument which derives its value from an underlying asset. There are a wide range of options traded today. Some are simple and plain, like the European options, while others are very difficult to evaluate. Both buyers and sellers continue to look for efficient algorithms and faster technology to price options for profit. In this thesis, I will first map the PSO parameters to the parameters in the option pricing problem. Then, I extend this to study pricing of complex chooser option. Further, I design a parallel algorithm that avails of the inherent concurrency in PSO while searching for a optimum solution. For implementation of my algorithm I used graphics processor unit (GPU). Analyzing the characteristics of PSO and option pricing, I propose a strategy to normalize some of the PSO parameters that helps in better understanding the sensitivity of various parameters on option pricing results.

Acknowledgments

Firstly, I want to thank my advisors, Dr. Ruppa Thulasiram and Dr. Parimala Thulasiraman, for their able guidance during my research and study at University of Manitoba. My very sincere thanks to them for their support, encouragement and great effort they put into training me during the course of this thesis.

I would like to thank my father “Mr Ashok Sharma” and my brother “Keshav Sharma” for their emotional and moral support all through my life. Everything that I have achieved so far can be attributed to their love and sense of security they provided to me.

My deepest gratitude to “Mr. Malkiat Kamboz” and his family for supporting me in Winnipeg in every possible way. I would also like to thank “Ashish Phutela” , “Aman Brar” , “Bhavdeep Pabla” and “Jose Juan Mijares Chan” for sharing some memorable moments here in Canada.

Finally I would like to thank all the graduate students at the Department of Computer Science for making it a very friendly place to work. I want to thank them for all their help and support.

This thesis is dedicated to loving memory of my late mother , "Dr. Aruna Sharma". I remember her for all her kindness and wisdom.

Contents

List of Tables

List of Figures

Chapter 1

Introduction and Background

In recent years, the GPU architecture has been dominating the general purpose computing market. The current GPU GeForce GTX 550Ti architecture by Nvidia, with its 192 CUDA cores, is 28% faster than previous recently released architecture GeForce GTX 450. The number of cores together with the processing power of the architecture is increasing dramatically. With the introduction of new programming languages such as OpenCL by Kronos Group for GPUs and other accelerators, the accelerators can be efficiently used to solve large real world problems beside graphic applications.

In finance problems such as option pricing, value at risk and portfolio optimization demand efficient algorithms and high performance computing capabilities. Investors desire rapid solutions to beat the competition and any delay in information processing can be translated into huge losses. Therefore, financial applications can benefit from the tremendous parallelism available on GPU. In this thesis, I design an efficient bio-inspired option pricing algorithm to solve the problem in particular the complex chooser option and parallelize the algorithm on the GPU architecture.

1.1 Financial Option

A financial option is a contract (with an expiration date) between two parties. The holder of the option contract gets a right to exercise this option at his/her will during the contract period. The other party, known as the writer of the option is obligated to the decisions of the holder. There are two types of options: call or put option. A call (put) option is a contract that gives the right to its holder to buy (sell) a pre-specified underlying asset at a pre-determined contract price (the strike price). Depending on when the contract is exercised, there are two well-known styles of options also called as vanilla options: European and American. In European style, an option can be exercised only on the date of expiration whereas in an American style, an option can be exercised on any date before the expiry date. The complex chooser option is another style of option categorized as exotic options. In this research, I consider the complex chooser option for various reasons. This option has not been studied due to its complex nature. Also, this product combines various other options in its fold and hence it's a step away from building a financial portfolio. Also, it is quite challenging to price this option.

A complex chooser option Rubinstein [1991] gives the holder the right to choose an option type and style among various styles, during the pre-determined time period of the option contract (generally within the first 25% of the contract). An example in using the complex chooser option is in optimizing a portfolio. A portfolio is a collection of options and other financial instruments owned by a single investor or an organization. Pricing the portfolio involves evaluating all the options and suggesting the investor an appropriate exercise time to optimize the profitability of the portfolio. Due to the flexibility of this option, a complex chooser option has attracted many investors in the recent years. However, as the investor has an opportunity to "choose" between a call and a put, the complex chooser option is generally more expensive than simple options and is computationally more challenging because double decisions need to be made with complex chooser option.

1.2 Background Literature

One classical mathematical model for option pricing was developed by Black and Scholes Black and Scholes [1973] and Merton Merton [1973]. This model assumes constant volatility for the underlying asset during the contract period of the option. It produces an exact solution in closed-form for an European option. The Black-Scholes-Merton model can be extended to get an approximate closed-form solution for a complex chooser option. However, this approximate solution becomes totally unreliable under volatile and dynamic market conditions. Therefore, advanced algorithms and computing platforms are always in demand for pricing options. For my thesis I use the current technology of GPUs to price complex chooser option.

In cases where an exact solution cannot be found, numerical techniques have been developed such as binomial lattice Cox et al. [1979], the finite-differencing technique Tavella and Randall [2000], Monte Carlo Boyle [1977] simulation, fast Fourier Transform approach Carr and Madan [1999] and others for general option pricing. In option pricing, accuracy is one of the important issues any technique has to satisfy.

The accuracy of numerical methods can be improved through increased computation time for a given instance of a solution as shown by Heston and Zhou [2000].It is not an exhaustive optimal solution for the problem. To this effect Barua et al. [2005]; Rahmayil et al. [2004]; Thulasiram et al. [2001]; Thulasiram and Thulasiraman [2003]; Jauvion and Nguyen [2008]; Podlozhnyuk [2008]; Solomon et al. [2010] have developed parallel algorithms on high performance computers and have gained improved performance.

The strive for accuracy and better models for option pricing in a volatile market has led researchers to study heuristic approaches involving evolutionary Chidambaran et al. [1999]; Yin et al. [2007]; Keber and Schuster [2002] and swarm intelligence Brabazon and O'Neil [2006]; Kumar et al. [2008b]; Jha et al. [2009]; Prasain et al. [2010a,b] techniques. These techniques start with an initial search space and can work simultaneously to find a solution. During the course of the execution, through co-operation of individual or using genetic operators, they are able to limit the search space in which the solution is found.

In this thesis, I use particle swarm optimization (PSO) algorithm for solving the option pricing problem. Due to the robustness and simplicity of PSO, the technique has been been used in various applications Meneses et al. [2009]; Jin and Samii [2005]; Schutte et al. [2004]. The general PSO algorithm, as described by Kennedy and Eberhart Kennedy and Eberhart [1995], works by introducing a number of particles into the solution space (a continuous space where each point represents one possible solution) and moving these particles throughout the search space, searching for an optimal solution. In Jha et al. [2009]; Prasain et al. [2010a], an initial attempt to map the PSO algorithm for pricing European call option was made with certain assumptions that are not practicable for real market scenarios. For my research, I take a different approach than earlier studies Prasain et al. [2010a,b]. I design a normalized PSO algorithm (Chapter 5) and show that it is more suitable for real market conditions.

Due to tremendous amount of parallelism available in the PSO algorithm, efforts have been made in developing and designing parallel PSO algorithms Schutte et al. [2004]; Chang et al. [2005]; Venter and Sobieszczanski-Sobieski [2006]; Prasain et al. [2010b]; Solomon et al. [2011]. In recent years, several works Li et al. [2007]; Jauvion and Nguyen [2008]; Mussi et al. [2010]; Solomon et al. [2011] have capitalized on the performance benefits of using a GPU and have studied the parallelization of the PSO algorithm on these accelerators. However, the GPU architecture is constrained in regards to the types of algorithms that work well. The structured, predictable nature of very regular, synchronous data-parallel algorithms work well with the SIMD-esque GPU architecture.

The PSO algorithm is an iterative, synchronous algorithm. It may require several iterations to reach the optimal solution. While there is synchronization and communication between iterations, the particles can work independently within iteration. This feature of the PSO algorithm, makes it suitable for parallelization on the GPU architecture.

The contribution of my research is as follows: (i) Design a normalized parallel PSO algorithm for pricing complex chooser option; (ii) compare the algorithm to the existing approximate Black-Scholes model for complex chooser option; (iii)Design a parallel the algorithm for multi-core architecture, implement it on a GPU architecture and study its performance.

1.3 GPU and CUDA Programming Model

This section briefly describes the GPU architecture and the CUDA programming model used in this research.

The GPU core contains an array of Streaming Multiprocessors or SMs. Each SM is composed of several Scalar Processors or SPs. The SPs are the base unit that executes a thread in the SM. Each SM executes threads following a model similar to SIMD which NVIDIA NVIDIA [2008] refers to as SIMT, or Single Instruction Multiple Threads. For example, the latest architecture of GPU, the GT300 (or Fermi) consists of three billion transistors with 16 SMs each containing 32 cores. The hardware performs tasks such as thread creation, resource management and thread scheduling.

The GPU consists on an off-chip memory, shared memory, registers and texture or constant memory. Among them, the global memory is the slowest to access which adds significant overhead in overall performance if used frequently. All SMs have access to this memory. Techniques such as global coalescing can be used to help reduce the performance degradation of the slow memory access. That is, if threads follow certain access patterns[1], the number of global memory reads and writes can be significantly reduced, resulting in the negation of much of the performance issues related to memory latency. The shared memory is resident on each SM, and is accessible to threads exclusively executed on the SMs. NVIDIA NVIDIA [2008] claims shared

[1]These patterns are described by NVIDIA NVIDIA [2008] in the CUDA Programming Manual

memory is as fast as accessing a register if no bank conflicts exist[2]. Therefore, the importance of using shared memory as a developer-controlled cache is not to be underestimated. Constant/texture memory acts as caches and is faster than the global memory but slower than the shared memory.

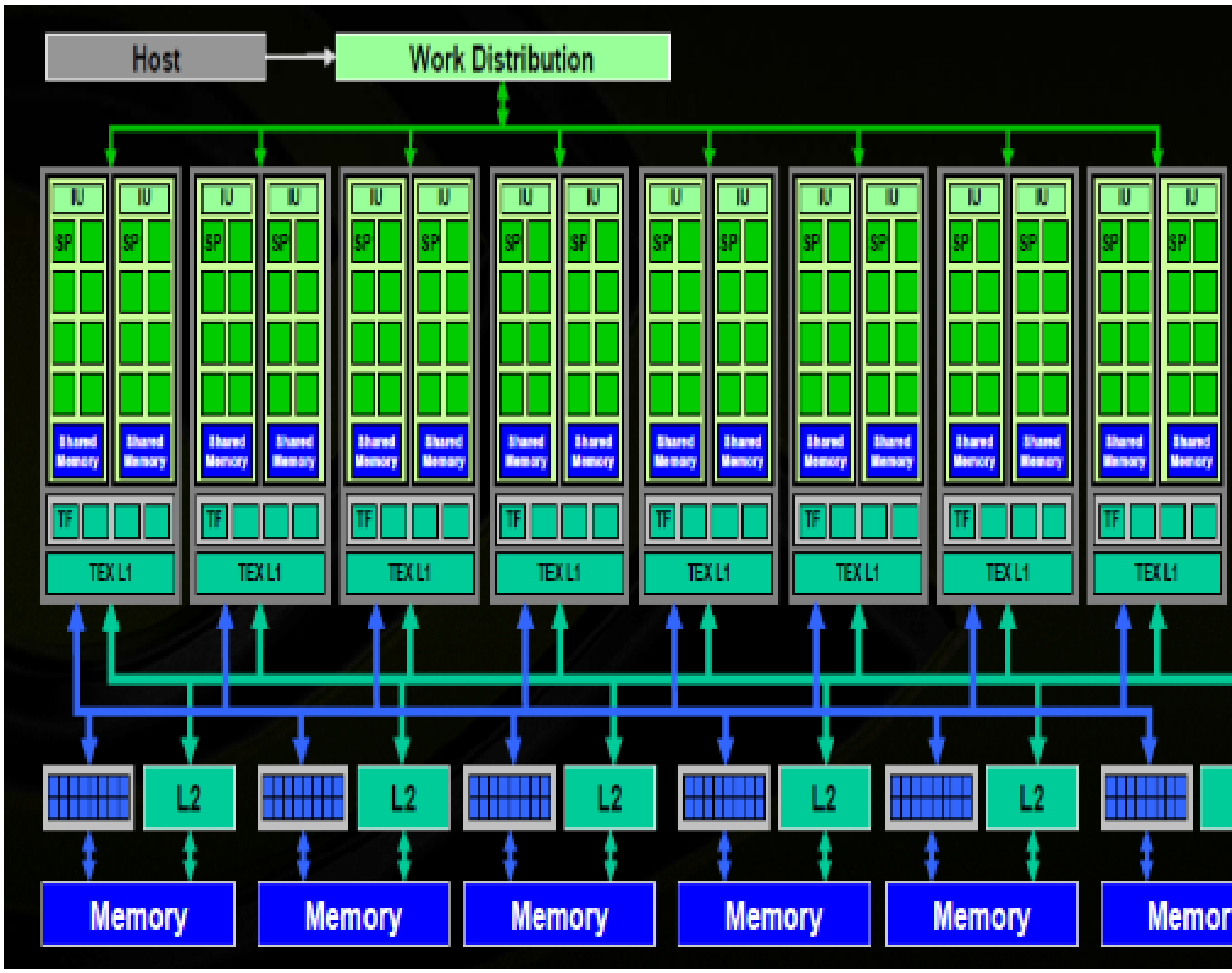

Figure 1.1: GPU Architecture

[2]Shared memory is split in to 16 32-bit wide banks, multiple requests for data from the same bank arriving at the same time are serialized.

The CUDA threading model exposes a hierarchy of thread groupings. At the highest level there exists the thread grid, which encapsulates all threads executing the application. Groupings of threads within the grid form thread blocks. Threads within blocks are assigned a thread ID unique only among threads in the same block.

As a further organization, threads within each block are ordered into 32-thread *warps*. Each thread within a warp is given the same instruction to execute as all the other threads within that warp. When branching occurs, threads in a warp which have diverged are marked as inactive and do not execute any useful work until instructions from their path of the branch are issued to the warp. This is, in effect, the SIMT model of execution. Different warps within a thread block can be executing different instructions from one another, but threads within a warp must execute the same instruction or no instruction at all.

Chapter 2

Conventional option pricing techniques (Related Work)

In this chapter I will discuss some basic option pricing techniques that are in use in the finance community.

2.1 Black-Scholes-Merton model

In early 1970s the economists Fischer Black and Myron Scholes revolutionized the option trading by providing a closed form solution for European call and put options. Black-Scholes model is based on the solution of partial differential equations, mostly used for heat flow problems. The mathematician, Robert C. Merton provided the initial mathematical model for option pricing. For this work Robert C. Merton and Myron Scholes received the Nobel Memorial Prize in Economics in 1997.

Although the initial Black-Scholes-Merton model was developed for European call and put options only, many mathematicians have extended this to various other options. Approximate solutions for various options like complex chooser, arithmetic mean , time switch options etc. are based on the Black-Scholes-Merton model.

The classical Black-Scholes-Merton formula for a call option is given by Hull [2007]; Black and Scholes [1973].

$$C(S,t) = N(d_1) \times S - N(d_2) \times K \times e^{-r(T-t)} \tag{2.1}$$

where,

$$d_1 = \frac{\ln(S/K) + (r + \sigma^2/2)(T-t)}{\sigma\sqrt{(T-t)}}, and \tag{2.2}$$

$$d_2 = d_1 - \sigma\sqrt{(T-t)} \tag{2.3}$$

The Black-Scholes formula for put option is

$$P(S,t) = N(-d_2) \times K \times e^{-r(T-t)} - N(-d_1) \times S \tag{2.4}$$

In these equations, S is the underlying asset price, K is the strike price in the contract, r is the interest rate, σ is volatility and t is the expiration time. $N(d)$ represents the normal distribution function on d.

2.2 Binomial and trinomial lattice

The Black-Scholes-Merton model makes assumptions like constant market volatility and the model completely fails for practical options since volatility is not constant in the finance market; and special cases like market crashes. In 1979 Cox-Ross-Rubenstein (CRR) Cox et al. [1979] proposed a discrete time approach for option pricing.

Option pricing using this method involves a tree data structure as shown in figure 2.1. S is the stock price, T is time to maturity, σ is the market volatility and Δt is time between two time steps. u represents the factor by which the stock price increases and d represents the the factor by which the stock price decreases.

The value of u and d can be calculated from volatility (σ), as explained by Cox-Ross-Rubenstein (CRR) Cox et al. [1979] using the following equations.

$$u = e^{\sigma\sqrt{\Delta t}}$$

$$d = e^{-\sigma\sqrt{\Delta t}} = \frac{1}{u}$$

It can be noticed in the above figure that at each time step, the stock price can either go up or down. This represents the real market scenario more correctly.

A natural extension of the binomial lattice is the trinomial lattice. At each node the stock price can go up, down or remain constant. The difference can be seen in the figure 2.2

Because trinomial lattice considers an extra case, i.e. when the stock price remains constant, this method is closer to real market scenario than the binomial lattice.

Binomial and trinomial lattice are frequently used for option pricing, mostly because they are very easy to implement and parallelize. However, they have some limitations, first, both are very rigid; that is, the stock prices can go up or down by a fixed amount and they can not be implemented with dynamic volatility. Both

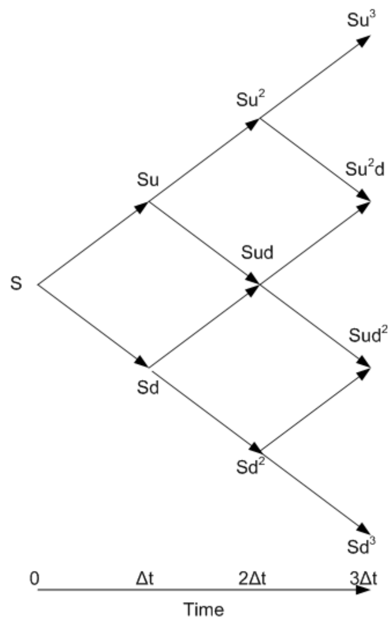

Figure 2.1: Binomial Price Tree

Figure 2.2: Trinomial Price Tree

these limitations divert binomial and trinomial lattice from real market scenario. To overcome these limitations, heuristics have been used in the finance community for option pricing, some of which are explained in the next section

2.3 Heuristic techniques for option pricing

Heuristic techniques have been used on various aspects of financial applications Hutchinson et al. [1994]; Chen et al. [2006]. For option pricing, heuristic has been used in finding approximate solutions or formulas for calculating implied volatilities. Implied volatility is the volatility of an asset that is calculated using the value of the option. There are no closed form solutions for calculating implied volatilities. Therefore, analytical approximations Bharadia et al. [1995, 1996]; Chance [1996]; Keber [1999] and recently heuristics have been considered Keber and Schuster [2003]. The focus of my research is not in deriving formulas for implied volatility, but rests is designing an efficient algorithm for pricing the options. Therefore, this section describes heuristics used in the literature to price options with varying volatilities.

An important advantage of heuristics or metaheuristics approaches over other numerical techniques used in pricing options are techniques ability to incorporate a known approximate solution initially. An algorithm would evolve over many iterations starting from this initial solution. This initial approximate solution could be the solution obtained from Black-Scholes-Merton model. In Chidambaran et al. [1999], the authors propose a genetic programming (GP) approach to price options. The authors incorporate an approximate solution of the Black-Scholes-Merton model initially into the gene pool for evolving future generations. The authors claim that GP produces better approximations than Black-Scholes-Merton model when the underlying assets follow a jump diffusion process.

In GP technique, the mutation and crossover probability rates are generally fixed. In Yin et al. [2007], the authors dynamically alter these rates in each GP run. The authors include the dynamic volatility in their algorithm to match the real market scenario. They claim that this adaptive algorithm captures the market in real time and produces better approximations.

Kumar et al. Kumar et al. [2008b], developed an ant colony optimization (ACO) Dorigo et al. [1996] based algorithm to price options. Their dynamic iterative algorithm inherently captures the market volatility during the life of the option. Initially, all ants start searching the solution space from an initial node. The objective of the ants is to find the best node (time or profit) to exercise an option. Ants move towards the global best node by choosing a path to the next node that has high concentration of the pheromone. After a few iterations, more ants are injected at the best node to explore the solution space further. Kumar et al. Kumar et al. [2008a] showed that the algorithm performs better than binomial-lattice algorithm. The algorithm, however, does not optimize on exercise.

Jha et al. Jha et al. [2009] used Particle Swarm Optimization (PSO) for pricing European options. It was a rudimentary attempt of using PSO for option pricing and the authors used very small number of particles. There are two limitations in their approach. First, the authors assume constant market volatility. Second, they do not compute the best possible exercise time. This restriction would eliminate evaluation of American options. Also, Jha et al. Jha et al. [2009] perform all their experiments in MATLAB using PSO as a black box which is very difficult to manipulate with real market data.

Prasain et al. Prasain et al. [2010a] and Prasain Prasain et al. [2010b] efficiently map the PSO algorithm to the option pricing problem. They also incorporate dynamic market volatility into the PSO algorithm and estimate the best exercise time. This algorithm accurately represents the real market conditions and it can be used to evaluate American options as well. However, their algorithm stagnates at a solution which may not necessarily be optimal. This, I believe is due to the constants and random numbers generated by the PSO algorithm.

For my research, I propose a normalization scheme for some of these PSO parameters to better suit my application, the complex chooser option. Note that to the best of my knowledge there is no work published that uses PSO for pricing complex chooser option. In the next chapter I will explain the complex chooser option in details.

Chapter 3

Complex Chooser Option

In this section I explain the complex chooser option in detail. First, I explain the inception and meaning of the complex chooser option followed by the mathematical model used to evaluate the complex chooser option.

3.1 Options for the undecided

The complex chooser option also known as the “option for the undecided” was introduced by Rubinstein in 1991 Rubinstein [1991]. This option gives the holder the opportunity to choose between the two types of options call or put. In other words, the holder purchases the option now, but after a pre-determined period of time he/she decides if the option would be a call or a put. The risk involved in option trading is reduced to a small extent by letting the holder decide between call or put through the chooser option. Therefore, the investors who cannot decide about the type and style of options at the time of contract can be lured to chooser option.

Chooser options traded in the past were quite simple; they would just comprise of two types: a call and a put. For European investors the chooser options would comprise of European call and European put, while for the American investors those would be American call and American put. Also, regardless of which style (European/American) is choosen, the basic parameters (stock price, strike price, expiration time) do not change for chooser option. Rubinstein Rubinstein [1991] later introduced the complex chooser option, which comprises of options with different stock prices, strike prices and expiration times. The complex chooser option can therefore be considered as a portfolio of many options.

Pay-off for a call or a put option is given by $f_c = Max(S - K, 0)$, $f_p = Max(K - S, 0)$, respectively where S and K are the underlying asset price and strike price of the contract respectively. f_c and f_p are the payoff from a call or a put option.

In simple terms, the pay-off of a chooser option can be written as

$$f_{chooser} = Max[f_c, f_p; t] \tag{3.1}$$

where t is the pre-determined time before which the investor has to decide between a call or a put option for the chooser option he/she holds.

3.2 Evaluation using Black-Scholes-Merton formula

Black-Scholes-Merton formula can be used to evaluate the complex chooser option. The classical Black-Scholes-Merton formula for a call option is given by Hull [2007]; Black and Scholes [1973].

$$C(S,t) = N(d_1) \times S - N(d_2) \times K \times e^{-r(T-t)} \tag{3.2}$$

where,

$$d_1 = \frac{\ln(S/K) + (r + \sigma^2/2)(T-t)}{\sigma\sqrt{(T-t)}}, and \tag{3.3}$$

$$d_2 = d_1 - \sigma\sqrt{(T-t)} \tag{3.4}$$

The Black-Scholes-Merton formula for put option is

$$P(S,t) = N(-d_2) \times K \times e^{-r(T-t)} - N(-d_1) \times S \tag{3.5}$$

In these equations, S is the underlying asset price, K is the strike price in the contract, r is the interest rate, σ is volatility and T is the expiration time. $N(d)$ represents the normal distribution function on d. For simplicity the Black-Scholes-Merton formula for call option can be written as a function:

$c =$ Black-Scholes ("c", S, K, t, σ, r) where "c" represents call option.

Similarly, the put option can be represented as

$p =$ Black-Scholes ("p", S, K, t, σ, r) where "p" represents put option.

We will also use the terms $N(d1)$ and $N(-d1)$ for call and put options, respectively, in calculations later in this section. Therefore for simplicity, we represent them as the functions:

$d_c =$ Delta ("c", S, K, t, σ, r) where "c" represents call option and,

$d_p =$ Delta ("p", S, K, t, σ, r) where "p" represents put option

The option price (w)of a complex chooser option as given by Rubinstein Rubinstein [1991] is given by

$$w = Se^{(b-r)T_c}M(d_1, y_1; \rho_1) - K_c e^{-rT_c} M(d_2, y_1 - \sigma\sqrt{T_c}; \rho_1) - Se^{(b-r)T_c}M(d_1, -y_2; \rho_2) + K_p e^{-rT_p} M(-d_2, \tag{3.6}$$

where T_c and T_p are the time to maturity on the call and put respectively. b is the interest rate adjustment for dividend paying stock. $M's$ are the Cumulative Bivariate Normal Distribution function. In my experiments I consider non-dividend paying stock and hence the value of b is 0.

$$d_1 = \frac{\ln(S/I) + (b + \sigma^2/2)t}{\sigma\sqrt{(t)}} \tag{3.7}$$

$$d_2 = d_1 - \sigma\sqrt{(t)} \tag{3.8}$$

$$y_1 = \frac{\ln(S/K_c) + (b + \sigma^2/2)T_c}{\sigma\sqrt{(T_c)}} \tag{3.9}$$

$$y_2 = \frac{\ln(S/K_p) + (b + \sigma^2/2)T_p}{\sigma\sqrt{(T_p)}} \tag{3.10}$$

$$\rho_1 = \sqrt{t/T_c} \tag{3.11}$$

$$\rho_2 = \sqrt{t/T_p} \tag{3.12}$$

I in the above equations is called the critical value chooser. The Black-Scholes-Merton formula for call and put options given earlier are used to find this critical value chooser. Algorithm 1 evaluates critical chooser value. K_c and K_p represent the strike prices and T_c and T_p represent the expiration times for call and put options, respectively. This is basically Newton-Raphson method, which is generally used to approximate roots of the real valued functions.

Algorithm 1 Function CriticalValueChooser(S, K_c, K_p, T, T_c,T_p, r, σ)

```
1: $S_v$ = S
2: $c_i$ = BlackScholes (“c”, S, $K_c$, $T_c - T$,$\sigma$, r)
3: $p_i$ = BlackScholes (“p”, S, $K_p$, $T_p - T$,$\sigma$, r)
4: $d_c$ = Delta (“c”, S, $K_c$, $T_c - T$,$\sigma$, r)
5: $d_p$ = Delta (“p”, S, $K_p$, $T_p - T$,$\sigma$, r)
6: $y_i = c_i$ - $p_i$
7: $d_i = d_c$ - $d_p$
8: $\epsilon = 0.000001$
9: while ($y_i > \epsilon$) do
10:    $S_v = S_v$ - $y_i/d_i$
11:    $c_i$ = BlackScholes (“c”, S, $K_c$, $T_c - T$,$\sigma$, r)
12:    $p_i$ = BlackScholes (“p”, S, $K_p$, $T_p - T$,$\sigma$, r)
13:    $d_c$ = Delta (“c”, S, $K_c$, $T_c - T$,$\sigma$, r)
14:    $d_p$ = Delta (“p”, S, $K_p$, $T_p - T$,$\sigma$, r)
15:    $y_i = c_i$ - $p_i$
16:    $d_i = d_c$ - $d_p$
17: end while
CriticalValueChooser(I) = $S_v$
```

With the critical value calculated, d_1 and d_2 in Equations 3.7 and 3.8 can be computed,which in turn can be used to compute w, the approximate value of a complex chooser option from Equation 3.6.

In this chapter I explained the complex chooser option and its mathematical derivation. Another important component of my thesis is the particle swarm optimization, which I will explain in the next chapter.

Chapter 4

Particle Swarm Optimization

Particle swarm optimization is a population-based technique conceptualized from the observation of flock of birds(figure 4.1) or school of fish(figure 4.2) migrating or looking for food. Initially, each bird will look in its own neighbourhood for a potential food source.

Figure 4.1: Flock of birds

At the end of the first iteration, the birds decide on a location that might lead to a food source. Each bird will then compare its result with other birds of the flock and will make a decision to either fly to another location or stay in its current location.

As more and more iterations are performed, the probability that the flock of birds will come closer to the food source is higher. This is the basic concept of PSO.

Figure 4.2: School of fish

The example mentioned above is homologous to the option pricing problem. There could be various possible solutions for a given option. Our aim is to find a solution that best satisfies the holder of the option. Therefore, a good mapping has to be developed between PSO and option pricing. Hari Parasin Parasin [2010] has done an initial work in this direction. Discussion on this is provided in the related work (section 2) motivating the current work.

There are four steps in a PSO algorithm Kennedy and Eberhart [1995]: initialization, position update, evaluation, and termination.

4.1 Initialization

Initialize N particles randomly in a solution space. Equation 4.1 describes initial position, and Equation 4.2 gives the initial velocity.

$$\overrightarrow{X_0^i} = \overrightarrow{X_{min}} + r_1 \times (\overrightarrow{X_{max}} - \overrightarrow{X_{min}}) \tag{4.1}$$

$$\overrightarrow{V_0^i} = \overrightarrow{X_i} \tag{4.2}$$

For a particle i, $\overrightarrow{X_0^i}$ is the initial position, $\overrightarrow{X_{min}}$ is the minimum allowable position in the sample space, $\overrightarrow{X_{max}}$ is the maximum allowable position in the sample space, r_1 is an uniform random number within the range of [0,1], and $\overrightarrow{V_0^i}$ is the initial velocity.

4.2 Position Update

During the execution of the algorithm, each particle i monitors four values: its current position ($\overrightarrow{X_i}$), the best position it reached in previous cycles ($\overrightarrow{P_i}$), its flying velocity ($\overrightarrow{V_i}$), and the swarm best position ($\overrightarrow{P^g}$). These four values are represented in vectors as:$\overrightarrow{X_i} = (x_{i1}, x_{i2}, x_{i3}, \ldots, x_{iD})$; $\overrightarrow{P_i} = (p_{i1}, p_{i2}, p_{i3}, \ldots, p_{iD})$; $\overrightarrow{V_i} = (v_{i1}, v_{i2}, v_{i3}, \ldots, v_{iD})$; $\overrightarrow{P^g} = (p_{g1}, v_{g2}, v_{g3}, \ldots, v_{gD})$, and are used to calculate the next velocity Kennedy and Eberhart [1995]:

$$\overrightarrow{V_{t+1}^i} = \omega \times \overrightarrow{V_t^i} + c_1 \times r_1 \times (\overrightarrow{P_t^i} - \overrightarrow{X_t^i}) + c_2 \times r_2 \times (\overrightarrow{P_t^g} - \overrightarrow{X_t^i}) \tag{4.3}$$

In Equation 4.3, r_1 and r_2 are *two* uniform random numbers within the range of $[0, 1]$, c_1 is self or local confident factor, and c_2 is swarm or global confident factor. The inertia weight ω balances the trade off between global and local search during optimization process. It decreases the velocity value at each time step to increase PSO exploitation. It is usually set in the range between [0,1].

The particle's next position is calculated using the equation 4.4 Shi and Eberhart [1998].

$$\overrightarrow{X_{t+1}^i} = \overrightarrow{X_t^i} + \overrightarrow{V_{t+1}^i} \tag{4.4}$$

4.3 Evaluation

The evaluation step depends on the application under consideration and its fitness function. Local and global updates are done if particles find better fitness values. The algorithm terminates after certain number of iterations predefined by the user for the respective application.

4.4 Limitations of PSO

As mentioned earlier, Particle swarm optimization is a population-based technique conceptualized from the observation of flock of birds or school of fish. The mathematical model derived from these observations are very generic and cannot be used directly for a given application. The mapping of PSO equations to a given application is the real challenge. Once this mapping is successfully achieved it might hold valid for the set of parameters currently considered in the application. That is, if a new parameter is added to the application or existing one is removed, the PSO model may not work well.

Another major drawback of PSO is that the number of particles to be used is not well defined. The optimal number of particles to be used has to be found through a trial and error method with using various experiments.

In this chapter I explained the conventional PSO algorithm. However, for my thesis I changed this conventional PSO algorithm and designed a normalized PSO algorithm (NPSO) which I will explain in the next chapter.

Chapter 5

Normalized Particle Swarm Optimization (NPSO)

In this chapter I will explain the improvements made to the PSO algorithm 4.4 Shi and Eberhart [1998] proposed by Prasain et al. [2010b] for the option pricing problem. First, I will briefly explain the mapping of the basic PSO algorithm to the option pricing problem that Prasain et al. [2010b] considered. The position vector ($\overrightarrow{X_t^i}$) and the velocity vector ($\overrightarrow{V_t^i}$) in PSO refer to change in stock price, and the new stock price at time t respectively. $\overrightarrow{P_i}$ and $\overrightarrow{P^g}$ refer to the particle's local best option value and the global best option value achieved by the swarm.

In Equation 4.3, there are two random numbers (r_1 and r_2) and two constant parameters(c_1 and c_2). Random numbers influence the movement of the particles. In the literature, random number generators are used to compute r_1 and r_2. In my work, I use a random number generator to select r_1 in the range [0,1]. However, I set $r_2 = \sigma$ and justify this follows : In the velocity equation 4.3, r_2 is linked to the the global best value, *gbest*. That is, I use a fraction of the value computed by $g_{best} - x_i(t-1)$. For the option pricing problem the particles are moving towards the common goal of finding a solution for one single investor. This implies, all particles should globally experience the same market fluctuations. If each particle selects a different random number, each particle will have a different experience which is incorrect from a pratical point of view. Therefore, I associate r_2 with volatility, the parameter that influences the market fluctuation.

The constants, c_1 and c_2 weigh the contribution of the cognitive and social components, respectively Shi and Eberhart [1998]. In the literature, various values have been used for c_1 and c_2, without providing any explanations for their selection. In Prasain et al. [2010b], the authors conclude that for option pricing, the PSO algorithm has difficulty is finding the optimal solution in certain scenarios. Through experiments, I realize that the above constraints in c_1 and c_2 constants may have some effect on the results.

To this respect, in this work, I consider a normalization procedure to these constants. In finding the local best, each particle communicates with the nearest neighbour. Note that the local best value of a particle changes at each time. This change should be reflected in the velocity equation, which determines the change in stock prices at each time step. The particles share their local information to all particles in the swarm and compute the global best. This global best value allows particles to move closer to the optimal solution. I hypothesize that using these two parameters in the normalization will allow a particle to move closer to the optimal solution. Therefore, I normalize c_1 as the ratio of particle i's local best and the swarm's global best. Recall that c_2 provides the swarm's confident factor. All the particles are collaborating towards a single goal of identifying a node that provides the best option value. In other words, the particles are working for one investor. Therefore, a single particle can safely rely on the collective confidence factor (c_2) of the swarm. Hence, I set, $c_2 = 1$ and $c_2 = 1 > c_1$. The normalized equation used in this research for the velocity and position formulas at time $(t+1)$ for a particle i are given below:

$$V_{t+1}^i = \omega * V_t^i + \frac{l_{best}}{g_{best}} * r_1(l_{best} - X_t^i) + \sigma * (g_{best} - X_t^i) \tag{5.1}$$

$$X_{t+1}^i = X_t^i + V_{t+1}^i \tag{5.2}$$

In PSO, ω, balances the exploration-exploitation of the particles movements. This is usually set in the interval [0,1]. In Shi and Eberhart [1998], $\omega = 0.5 + rand/2$, where rand is a random number. This parameter uses the velocity of the particle's previous iteration and decreases it by some fraction to increase exploitation. Prasain et al. Prasain et al. [2010b] set $\omega = \sigma + k$ where $k$is some constant. In option pricing, volatility (σ) is an important parameter that captures the dynamics of the market. I found through various experiments that when the market is highly volatile, the assumptions made by Prasain et al. Prasain et al. [2010b] about ω is inefficient. I argue this as follows.

The aim of ω is to restrict the high fluctuations in the movement of particles. On the other hand, σ reflects the real market fluctuation which is highly volatile. That is, it represents the fluctuations in the market, and maybe variable. Therefore, σ should not be mapped to a restrictive parameter, ω. This would produce incorrect results in the real market scenario. Also, in Prasain et al. [2010b], the authors map

ω to volatility with a very high number, $(0.5 + rand/2)$. In general, volatility is not that high in real market scenarios. Therefore, in my work, ω is randomly generated within the range [0,1] and I do not map ω to volatility.

In the PSO, algorithm, the velocity is usually set to a some maximum value ($\overrightarrow{V_{max}}$) to avoid a search explosion state. For option pricing, this refers to the maximum change in stock prices. Also, the position vectors are bounded by $\overrightarrow{X_{max}}$ and $\overrightarrow{X_{min}}$ to limit the search space. In my thesis, I avoid using $\overrightarrow{V_{max}}$, while the use of $\overrightarrow{X_{min}}$ and $\overrightarrow{X_{max}}$ are limited to the initialization step. After the initialization, boundary conditions are not used to encourage particles to explore the sample space as much as possible. This is another major improvement in using PSO for option pricing where very strict boundary conditions were used by Prasain et al. Prasain et al. [2010b] .

The normalized PSO algorithm (NPSO) for each particle is explained in Algorithm 2. The algorithm stops when the particles reach the expiration time or exceed the number of iterations.

5.1 Sensitivity analysis of the normalized PSO algorithm

In this section I do sensitivity analysis of my proposed NPSO model that works as a test for proof-of-concept. In the following section I compare my NPSO algorithm with the traditional Black-Scholes-Merton model. This works as a validation of my proposed NPSO model.

In general, it is inferred from the market analysis that with a long dated option

Algorithm 2 Algorithm:Normalized PSO

Randomly disperse particles into solution space

```
1:  for ($I = 0 \neq$ NumberofIterations) do
2:      for (all particles in swarm) do
3:          if Time < Expiration time then
4:              Compute fitness of current location
5:              Update $l_best$ if necessary
6:              Update $g_best$ if necessary
7:              Calculate $c_1 = \frac{l_{best}}{g_{best}}$
8:              Update velocity using Equation 5.1
9:              Update position using Equation 4.4
10:             Update Time
11:         else
12:             Terminate
13:         end if
14:     end for
15: end for
```

(lengthy expiration time) or with a highly volatile underlying asset, the value of the option is generally expected to be high. I did a one-at-a-time sampling Saltelli et al. [2008] on three financial parameters, i.e. volatility (σ), strike price (K) and expiration time (T). In finance, such analysis is generally characterized by Greek terms Θ (for expiration time) and υ (vega, for volatility).

The computed option values of my analysis are listed in table 5.1. In Table 5.1, I vary the volatility by keeping dampening factor (ω), expiration time (T) and strike price (K) constant. T is measured in years, K is set to \$45.

In Table 5.2, I increase the expiration time (T) by keeping ω, σ and K constant. I see that my normalized PSO model produces an increase in option values as should be the case in both cases.

In Table 5.3, I vary the K, by keeping ω, σ and T constant. I use a call option for this experiment. I see that increasing strike price decreases the option values as expected.

This sensitivity analysis confirms that my normalized PSO algorithm works without any loss of generality.

Table 5.1: Varying Volatility, $\omega = 0.5$, T = 1, K = 40

Simulation #	σ (%)	Option Value
1	20	2.94
2	40	3.20
3	60	3.40
4	80	3.70
5	90	3.90

Table 5.2: Varying Expiration Time, $\omega = 0.75$, $\sigma = 60\%$, K = 40

Simulation #	T	Option Value
1	1	7.10
2	1.5	7.8
3	2	7.02
4	2.5	8.20
5	3	9.00

Table 5.3: Varying Strike Price, $\omega = 0.25$, $\sigma = 50\%$, T = 1.5

Simulation #	K	Option Value
1	30	10.40
2	20	19.00
3	10	27.00
4	5	31.00

5.2 Comparison of NPSO with Black-Scholes-Merton model

In this section I compare the results of my normalized PSO algorithm (NPSO) with the approximate Black-Scholes-Merton model for European call and put options. I price European call and put option with varying number of particles (N) and volatilities (σ).

As the volatility increases, the expected change in the underlying asset price increases. This implies that there is a larger solution space to search. Therefore, I hypothesized that I had to increase the number of particles to accommodate this search. As can be seen in Table 5.4 and Table 5.5 increasing volatility together with increase in number of particles, produces option values close to approximate Black-Scholes-Merton model. That is, I observe that $N \propto \sigma$, or $N = C * \sigma$, where C is a constant of proportionality where $0.7 < c < 2$. I get an error of less than two percent in almost all the cases, a significant improvement over the results obtained by Prasain et al. Prasain et al. [2010b].

Table 5.4: Option values for European call

σ	N	NPSO	CBSM	Error %
10	20	3.84	3.86	0.51
20	30	4.76	4.61	3.4
30	30	5.586	5.58	0.10
40	40	6.63	6.61	0.43
50	50	7.79	7.66	1.7
60	60	8.58	8.71	1.4
70	60	9.56	9.76	1.9
80	70	10.76	10.79	0.23
90	70	11.69	11.81	0.98

Table 5.5: Option values for European put

σ	N	NPSO	CBSM	Error %
10	20	2.35	2.24	4.8
20	30	3.56	3.38	5.3
30	30	4.47	4.56	1.8
40	40	5.58	5.75	2.8
50	50	6.88	6.9	0.6
60	60	8.19	8.1	1.1
70	60	9.10	9.26	1.6
80	70	10.32	10.4	0.75
90	70	11.36	11.52	1.39

5.3 GPU implementation of NPSO for option pricing

In this section, I will explain the GPU implementation of NPSO for American, European, and Complex chooser options then, I extend this study to optimize a portfolio.

The initialization phase is done on the CPU. This includes uniformly distributing the particles in the solution space, calculating the initial positions and velocities

(Equation 4.1, Equation 4.2) for all the particles and finding the initial local and global best values. The initial local best values are set as the stock prices (initial positions). The initial global value is the maximum of the initial local best values. Using these values, c_1 is calculated as $\frac{l_{best}}{g_{best}}$ as explained in the previous chapter. After the initialization phase, the control is passed to the GPU where the particles process several iterations until the expiration time (user-defined) is met.

Bastos-Filho et al. Bastos-Filho et al. [2009] indicate that the quality of random numbers used does not affect the accuracy of the PSO algorithm. We implemented linear congruential generator Durst [1989], which is used in both the CPU and GPU for generating random numbers for r_1, r_2 and ω.

One thread block is used to evaluate one option. For simple options such as American and European, I use one thread block. I use one thread for each PSO particle. Therefore, the number of threads created and used in each block is equal to the number of PSO particles used in a particular simulation. For complex chooser and portfolio options, multiple thread blocks are used. In the case of complex chooser option, two thread blocks independently price a call and a put option. The option yielding a better profit will be selected as the option for the chooser option contract and further priced. Similarly, for the portfolio, one thread block is used for each option. A portfolio option may include several options. Therefore, I use several thread blocks for each option. We select the best option, locking time (chooser date) and exercise time for the investor.

Each thread is assigned a time stamp representing a physical time between the start and end of the contract period. The threads update the velocity and position using Equation 5.1. Each thread calculates its local best value and compares with the local best values in the previous iteration and replaces it, if necessary. The global best value is the local minimum of all the local best values. This is done using a parallel reduction. If the minimum value obtained is better than the current global best position, the thread replaces the global best position. Each particles stores the local best values and global bvalue in the shared memory of the GPU. Constants K, S, T are stored in registers.

In this chapter I explained the NPSO algorithm and the GPU implementation. In next chapter I will explain the experimental results.

Chapter 6

Experimental results

In the chapter 5, I validated my model by comparing the results from my normalized PSO algorithm with that from Black-Scholes-Merton model. In this chapter, I perform several experiments with static volatilty, dynamic volatility for different options and extend this for portfolio optimization. I discuss the option pricing results and compare the execution times of the GPU implementation with that of sequential implementation for different styles of options. As discussed in chapter 5, the number of particles used in these experiments is proportional to volatility unless otherwise mentioned.

6.1 Static Volatility

In these experiments, the volatility is a constant set statically at the beginning of the experiment. Table 6.1 shows the increase in option values as we increase the volatility, for American Call (AMCall), American Put (AMPut) and Complex

Chooser (CC) options. Higher volatility implies the better exercise opportunity and hence the increase in the option value.

Table 6.1: American Call,Put, Complex Chooser

σ	AMCall	AMPut	CC
10	3.8	3.82	4.34
20	5.3	5.02	5.83
30	6.3	5.7	6.64
40	7.7	6.7	7.7
50	8.7	7.33	9.52
60	9.5	8.91	11.38
70	10.4	9.6	12.08
80	11.13	9.95	12.98
90	11.9	10.89	13.3

Figure 6.1 and Figure 6.2 represent the execution time results of American call and put options, respectively. As I increase the volatility, I see that the execution time also increases due to the increase in number of computations. However, I get a speedup of 40. This was because I used the shared memory and the global reduction operations done in the shared memory are always faster than the reduction operation performed in the global memory. As each particle is assigned to one thread, all threads can work in parallel. When the particles need to find the global best, a reduction operation is performed in the shared memory. In sequential version of the algorithm, calculations for one particle are done at a time. As a result of this drawback the parallel algorithm works around 40 times faster when compared the sequential algorithm.

Figure 6.3 shows the execution time of the complex chooser option. As can be seen, the GPU implementation outperforms the sequential implementation. Compared to Figure 6.1 and Figure 6.2 there is a slight increase in execution time when I increase volatility to 90%. This is to be expected since in the complex chooser option, there

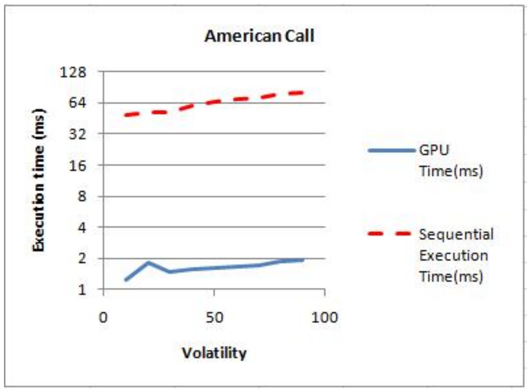

Figure 6.1: American Call Option: σ vs. Execution time

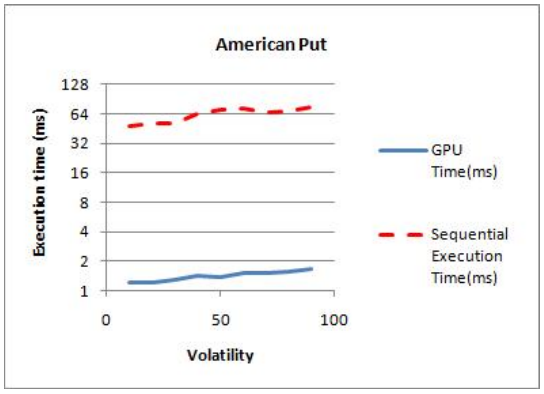

Figure 6.2: American Put Option: σ vs. Execution time

are more function to be evaluated as explained below. First, for each option, there is a separate thread block that computes the normalized particle optimization algorithm, which is done in parallel. Second, the values of various options are compared among themselves to select the best results to be provided to the investor. Hence, the speedup obtained is slightly less at around 36.

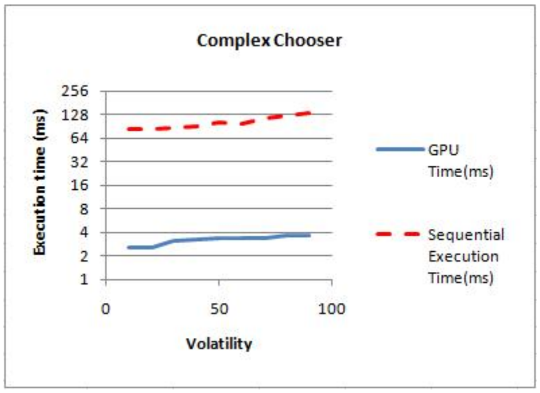

Figure 6.3: Complex Chooser Option: σ vs. Execution time

6.2 Dynamic Volatility

To capture the real market scenario to a greater extent, I varied the volatility dynamically at runtime. Since the volatility generally lies between 10% to 90%, I set the number of particles to reflect the average volatility (around 50%). Therefore, I used 45 particles, although changing the number of particles from 35-50 had insignificant effect on the pricing results. For volatility of 50% I used around 40-50 particles.

I evaluated five different options (European Call (EUCall), European Put (EUPut), American Call (AMCall), American Put (AMPut) and Complex Chooser (CC) options) with four different expiration time (3, 6, 9 and 12 months). As expiration time (T) increases, option prices should also increase. This is shown in Table 6.2.

Table 6.2: Option prices for European Call/Put, American Call/Put, Complex Chooser

T	EUCall	EUPut	AMCall	AMPut	CC
0.25	7.4	7.7	6.9	6.9	8.2
0.5	9.4	8.7	8.7	8.1	9.3
0.75	11.7	10.7	11.4	9.2	11
1.0	12.2	13.1	12.7	9.6	13.8

It can be seen that for all the options, as the T increases the option value increases. With increase in expiration time the solution space expands and there is more opportunity for the options to yield better value. Figures 6.4 and Figure 6.5 show the execution times of European call and put options under dynamic volatility. Figure 6.6 and Figure 6.7 show the execution times results of American call and put options under dynamic volatility. The GPU outperforms the sequential implementation significantly. The average speedup for exercising the option within one year is 40 for European call and put. For American call and put options the average speedup is about 45 and 42, respectively.

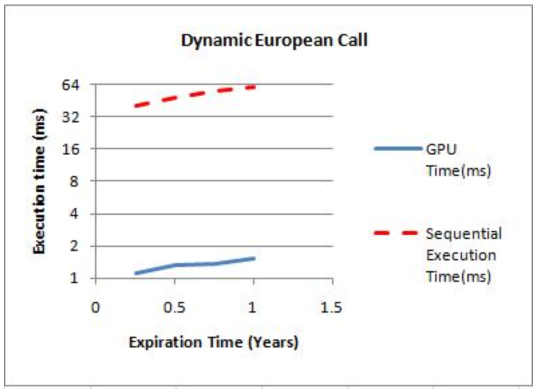

Figure 6.4: European Call Option: Expiration Time vs. Execution Time

Figure 6.8 shows the results of execution time for complex chooser option with dynamic volatility. The sequential execution time is approximately 97ms while the GPU execution time is about 3.4ms. therefore the average speedup attained is 29.

It can be seen from above results that GPUs outperform CPUs in all the cases. One of the main reason for this is that GPUs work very well on data parallel problems. To exploit this feature of GPU I designed my algorithm in such a way that GPU

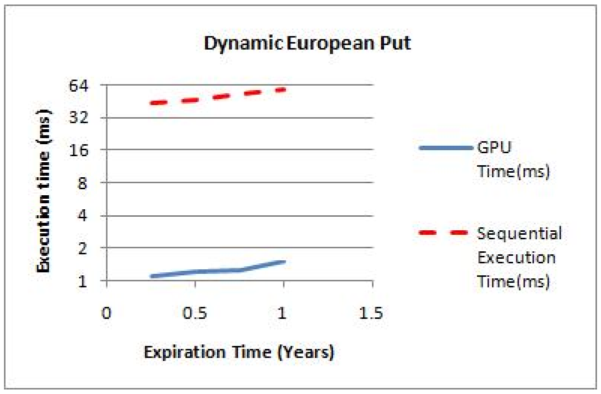

Figure 6.5: European Put Option: Expiration Time vs. Execution Time

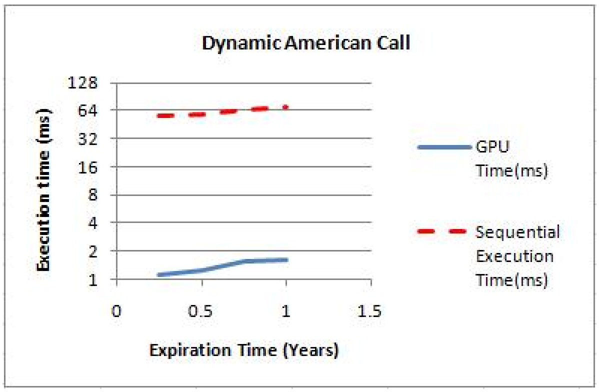

Figure 6.6: American Call Option: Expiration Time vs. Execution Time

resources could be utilized as much as possible. Secondly, I used shared memory of GPUs for reduction operation instead of global memory, which is much faster than the global memory.

6.3 Portfolio Management

I extend my experiments to optimize a portfolio under dynamic market conditions. The portfolio value is the total estimated cost of the holdings of the investor.

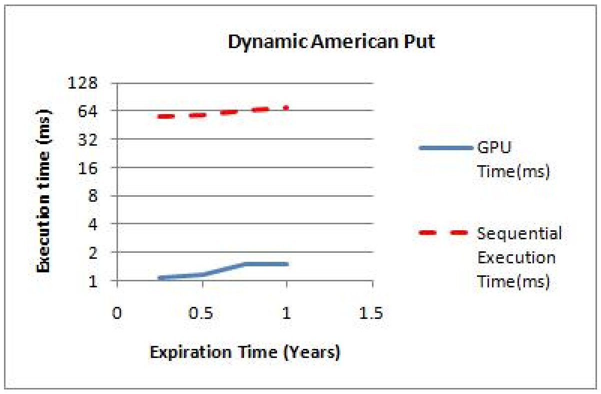

Figure 6.7: American Put Option: Expiration Time vs. Execution Time

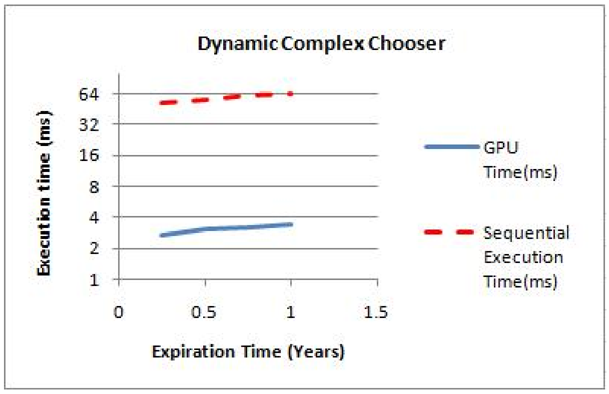

Figure 6.8: Complex Chooser Option: Expiration Time vs. Execution Time

Minimizing this cost is a continual optimization problem, which is computationally intense. I consider European Call and Put, American Call and Put options in my portfolio. I vary the chooser time (t^*) for various expiration times (T), 3, 6, 9 and 12 months. Among the different option styles (European call and put, American call and put) the option that optimizes the portfolio is selected. The results of this selection is shown in Table 6.3. The chooser time or locking time is the time when the investor makes the selection of an option. For example, at chooser or locking time the investor will select one of European call, European put, American call and American put. After this time the investor cannot change the type of option. Note that, if the investor selects European option at locking or chooser time, he can only exercise this option at the end of the expiration time. For American option, the investor has the option to exercise anytime before expiration time. This is reflected in the exercise time (ET) in Table 6.3. As can be seen from this table, selection of American option gives opportunity to exercise early. The implication of this ability to exercise early means that the liability of holding a stock is greatly reduced. As a rule of thumb its beneficial to have some money now than having the same amount of money in some future date. In other words, with American options in portfolio, the risk of huge loss is mediated to some extends. That is, the complex chooser option is not only beneficial in terms of investment decision on options but also could be used to optimize a portfolio as well.

Table 6.3: Portfolio Optimization

T	Option Type	Chooser Time	Exercise Time	Portfolio Value
3	EUCall	0.55	3	17.1
6	EUPut	1.2	6	23.1
9	AMCall	2.1	7.4	23.64
12	AMCall	1.9	6.8	24.1
15	AMCall	3.5	9.7	23.9
18	EUCall	4.3	18	25
21	EUCall	4.4	21	26.9
24	EUCall	5.1	24	28.5

Figure 6.9 shows the execution timing results of the portfolio option. We get a speedup of 40 for exercising the option in 2 years (24 months). The sequential execution time at 24 months is 1.4s and GPU execution time reduces significantly to 35ms.

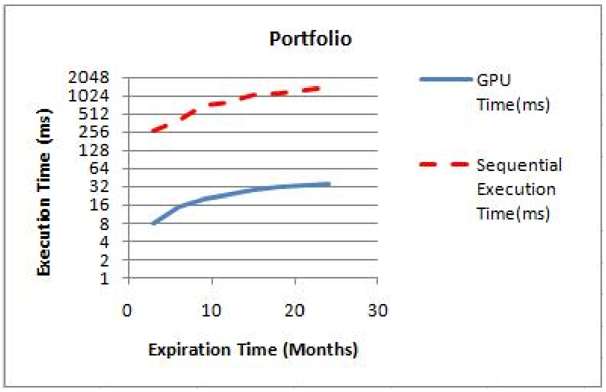

Figure 6.9: Portfolio Option: Expiration Time vs. Execution Time

6.4 Comparison with Hari's Parasin [2010] work

I don't compare my results with Hari's Parasin [2010] work because of the following reasons.

I found out that there were some errors in mapping PSO for option pricing. ω in the PSO model represents the dampening factor, which is used to keep the movement of particles restricted and prevent them from going out of the solution space. Hari mapped this to σ which is the market volatility and which represents the rate at which the stock price might change. So this was a mapping error which could result in incorrect option values.

Hari did not change the number of PSO particles as the solution space increased. I have explained earlier that we have to increase the number of particles as the the solution space increase for an optimal solution.

Due to these limitations Hari got a significant percent error as compared to Black-Scholes-Merton model, therefore I didn't compare my results with Hari's work.

6.5 Comparison of GPU execution time with MPI/OpenMI execution time

Since around 2008 NVIDIA's CUDA platform has been used for wide applications like image processing, statistics and option pricing etc. GPUs are very good for data parallel problems and they have been proved to be much better than MPI/OpenMP platforms by a large number of researches from different fields of applications. Due to this overwhelming growing importance of GPUs, the researches are shifting from MPI/OpenMP platforms to GPU. Knowing this fact very well through my literature survey, I did not spend efforts comparing my results with MPI/OpenMP.

6.6 Comparison with other Black-Scholes-Merton approximations

There are some mathematical models built on the Black-Scholes-Merton model. Understanding of these approximation models requires a lot of mathematical background, which I do not have at this time. So I did not use these models to verify my results.

In this chapter, I explained all the experiments performed. In general it can be seen that in static volatility, the option price increases with increase in market volatility. In dynamic volatility it can be seen that as the expiration time increases the option value increases. These two results support the fundamental principles of the option pricing. Secondly, we can see that with the use of GPUs significant speedups can be attained, which is always good for the investors in the highly competitive and dynamic markets.

Chapter 7

Conclusion

For my research I have designed a new normalized particle swarm optimization (NPSO) parallel algorithm to price an exotic financial option and accelerated its performance on GPU. As a proof-of-concept, I have done a series of sensitivity analysis to show that my algorithm generates pricing results that describe the trend of expected behavior without loss of any generality. I have validated my algorithm by comparing the results from my experiments to closed-form solutions due to Black-Scholes-Merton model for simple options. I have then parallelized my algorithm to implement on a GPU and studied its performance for pricing complex chooser option. I got a speedup of up to 40 in my experiments. This has been possible since in my model I have eliminated many parameters that adds to the overhead costs. Also, my NPSO algorithm allows incorporating volatility of the underlying asset dynamically, and hence this study captures real time market conditions. Moreover, the particles in my NPSO algorithm terminate computation after hitting the expiration time in three or four consecutive iterations. That is, I do not have to specify a boundary condition for particle movement. In addition, I show that my NPSO model for complex chooser option can be used for optimizing simple financial portfolio as well.

With the experience from the current study, I believe that my NPSO algorithm can be used to price any exotic financial option. My immediate future work is to extend this study for some few common exotic options such as swing options, lookback options etc. A portfolio generally comprises of many financial instruments such as stocks, bonds and options etc. Continually adjusting the portfolio for risk management is a real time task faced by many money managers in practice. My experience with PSO and accelerating PSO algorithm on GPU for simple portfolio has given me the impetus to take up risk management problem in the near future.

As an immediate extension of this work, I will use other approximate Black-Scholes-Merton equations to verify my results. I will also use my NSPO algorithm for other exotic options. Secondly, for the future work I will compare my results with the real market data.

Bibliography

S. Barua, R. K.Thulasiram, and P. Thulasiraman. High performance computing for a financial application using fast Fourier transform. In *The 11th International European Parallel Computing Conference, (EuroPar 2005)*, volume 3648, pages 1246–1253, Lisbon, Portugal, 2005.

C. J. A. Bastos-Filho, J. D. Andrade, M. R. S. Pita, and A. D. Ramos. Impact of the quality of random numbers generators on the performance of particle swarm optimization. In *Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics*, San Antonio, Texas, October 2009.

M. A. J. Bharadia, N. Christofides, and G. R. Salkin. Computing the Black Scholes implied volatility: Generalization of a simple formula. *Advances in Futures and Options Research*, 8:15–29, 1995.

M. A. J. Bharadia, N. Christofides, and G. R. Salkin. A quadratic method for the calculation of implied volatility using the Garman-Kohlhagen model. *Financial Analysts Journal*, 52:61–64, 1996.

F. Black and M. Scholes. The pricing of options and corporate liabilities. *Journal of Political Economy*, 81(3):637–654, January 1973.

P. Boyle. Options: A Monte Carlo approach. *Journal of Financial Economics*, 4: 223–238, 1977.

A. Brabazon and M. O'Neil. *Biologically Inspired Algorithms for Financial Modelling (Natural Computing Series)*. Springer-Verlag, 2006.

P. Carr and D. B. Madan. Option valuation using the fast Fourier transform. *The Journal of Computational Finance*, 2(4):61–73, 1999.

D. M. Chance. A generalized simple formula to compute the implied volatility. *The Financial Review*, 3:859–867, 1996.

J. Chang, S. Chu, J. Roddick, and J. Pan. A parallel particle swarm optimization algorithm with communication strategies. *Journal of Information Science and Engineering*, 21:809–818, 2005.

Y. Chen, B. Yang, and J. Dong. Time-series prediction using a local linear wavelet neural network. *Neurocomputing*, 69:449–465, 2006.

N. K. Chidambaran, C. W. J. Lee, and J. R. Trigueros. Option pricing via genetic programming. In *The 6th International Conference on Computational Finance*, pages 583–598. Leonard N. Stern School of Business, January 1999.

J. C. Cox, S. A. Ross, and M. Rubinstein. Options pricing: a simplified approach. *Journal of Financial Economics*, 7:229–263, 1979.

M. Dorigo, V. Maniezzo, and A. Colorni. Ant system–optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man and Cybernetics*, 26: 29–41, February 1996.

M. Durst. Using linear congruential generators for parallel random number generation. In *Proceedings of the 21st conference on winter simulation*, New York, NY, 1989.

S. Heston and G. Zhou. On the rate of convergence of discrete-time contingent claims. *Mathematical Finance*, 10:53–75, 2000.

J. Hull. *Options, Futures and Other Derivates.* Prentice Hall, 2007.

J. M. Hutchinson, A. W. Lo, and T. Poggio. A nonparametric approach to pricing and hedging derivative securities via learning networks. *Journal of Finance*, 49: 851–889, 1994.

G. Jauvion and T. Nguyen. Parallelized trinomial option pricing model on GPU with CUDA. http://www.arbitragis-research.com/cuda-in-computational-finance/ coxross-gpu.pdf /at download/file, August 2008.

G. K. Jha, H. Prasain, S. Kumar, R. K. Thulasiram, and P. Thulasiraman. Option pricing using particle swarm intelligence. In *The ACM Canadian Computer Science and Software Engineering Conference*, pages 267–272, May 2009.

N. Jin and Y. Samii. Parallel particle swarm optimization and finite-difference time-domain (PSO/FDTD) algorithm for multiband and wide-band patch antenna designs. *IEEE Transactions on Antenna and Propagation*, 53(11):3459–3468, 2005.

C. Keber. Genetically derived approximations for determining the implied volatility. *OR Specktrum*, 21:205–238, 1999.

C. Keber and M. G. Schuster. Option valuation with generalized ant programming. *The ACM Genetic and Evolutionary Computation Conference*, pages 74–81, 2002.

C. Keber and M. G. Schuster. Generalized ant programming in option pricing: determining implied volatilities based on american put options. In *IEEE Proceedings of Computational Intelligence in Financial Engineering*, pages 123–130, December 2003.

J. Kennedy and R. C. Eberhart. Particle swarm optimization. In *IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948, Perth, Australia, November 1995.

S. Kumar, R. K. Thulasiram, and P. Thulasiraman. A bioinspired algorithm to price options. In *The ACM C* Conference on Computer Science and Software Engienering*, pages 11–22, Montreal, May 2008a.

S. Kumar, R. K. Thulasiram, and P. Thulasiraman. *Ant Colony Optimization for Option Pricing*, volume 2 of *Natural Computing in Computational Finance*, chapter 4. Springer-Verlag, 2008b.

J. Li, D. Wang, S. Chi, and X. Hu. An efficient fine-grained parallel particle swarm optimization method based on GPU-acceleration. *International Journal of Innovation Comupting, Information and Control*, 3(B(6)):1707–1714, December 2007.

A. Meneses, M. Machado, and R. Schirru. Particle swarm optimization applied to the nuclear reload problem of a pressurized water reactor. *Progress in Nuclear Energy*, 51(2):319–326, 2009.

R. Merton. Theory of rational option pricing. *Bell Journal of Economics and Management Science*, 4(1):141–183, 1973.

L. Mussi, F. Daolio, and S. Cagnoni. Evaluation of parallel particle swarm optimization algorithms within the CUDA architecture. *Information Sciences*, September 2010.

NVIDIA. *Cuda 2.0 Programming Guide*. NVIDIA, June 2008.

H. Parasin. A parallel particle swarm optimization algorithm for option pricing. Master's thesis, University of Manitoba, Winnipeg, MB, June 2010.

V. Podlozhnyuk. Binomial option pricing model, April 2008. URL `http://developer.download.nvidia.com/compute/cuda/sdk/website/projects/binomialOptions/doc/binomialOptions.pdf`.

H. Prasain, R. . K. Thulasiram, and P. Thulasiraman. Mapping PSO and exploitation concurrency for option pricing on a homogeneous multi-core architecture. In *The ACM Genetic and Evolutionary Computation Conference*, Portland, OR, July 2010a.

H. Prasain, P. Thulasiraman, R. K. Thulasiram, and G. K. Jha. Performance evaluation of PSO-based algorithm for option pricing on homogeneous multi- core architecture. In *International Association of Science and Technology for Development*, Maui, HI, August 2010b.

S. Rahmayil, I. Shiller, and R. K. Thulasiram. Different estimators of the underlying asset's volatility and option pricing errors: parallel Monte-Carlo simulation. In *Proceedings of the International Conference on Computational Finance and its Applications*, pages 121–131, Bologna, Italy, 2004.

M. Rubinstein. Option for the undecided. *Risk Magazine*, 4(43), April 1991.

A. Saltelli, M. Ratto, T.Andres, F. Campolongo, J. Cariboni, D. Gatelli, M. Saisana, and S. Tarantola. *Global Sensitivity Analysis*. John Wiley Sons, 2008.

J. Schutte, J. Reinbolt, B. Fregly, R. Hafka, and A. George. Parallel global optimization with particle swarm algorithm. *International Journal of Numerical Methods in Engineering*, 61:2296–2315, 2004.

Y. Shi and R. Eberhart. A modified particle swarm optimizer. In *Proceedings IEEE World Congresss on Evolutionary Computation*, pages 69–73, Archorage, AK, USA, May 1998.

S. Solomon, R. K. Thulasiram, and P. Thulasiraman. Option pricing on the GPU. In *The 12th International Conference on High Performance Computing and Communications*, Melbourne, Australia, September 2010.

S. Solomon, P. Thulasiraman, and R. K. Thulasiram. Collaborative multi-swarm PSO for task matching using graphics processing units. In *The ACM Genetic and Evolutionary Computation Conference*, Dublin, Ireland, July 2011.

D. Tavella and C. Randall. *Pricing Financial Instruments: The Finite Differencing Method*. Wiley, April 2000.

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.

R. K. Thulasiram, L. Litov, H. Nojumi, C. Downing, and G. Gao. Multithreaded algorithms for pricing a class of complex options. In *IEEE/ACM International Parallel and Distribued Processing Symposium (IPDPS)*, San Francisco, CA, 2001.

G. Venter and J. Sobieszczanski-Sobieski. A parallel particle swarm optimization algorithm accelerated by asynchronous evaluations. *Journal of Aerospace Computing, Information, and Communication*, 3(3):123–137, March 2006.

Z. Yin, A. Brabazon, and C. O. Sullivan. Adaptive genetic programming for option pricing. *The ACM Genetic and Evolutionary Computation Conference*, pages 2588–2594, 2007.