PARALLEL COMPUTING IN ASIAN OPTION PRICING

by

Halis Sak

B.Sc., in Mechanical Engineering, Middle East Technical University, 2000

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science
in
Industrial Engineering

T.C. YÜKSEKÖĞRETİM KURULU
DOKÜMANTASYON MERKEZİ

139465

Boğaziçi University

2003

PARALLEL COMPUTING IN ASIAN OPTION PRICING

APPROVED BY:

Assist. Prof. İlkay Boduroğlu .....
(Thesis Supervisor)

Prof. Süleyman Özekici .....

Assoc. Prof. Can Özturan .....

DATE OF APPROVAL: 23.6.2003

ACKNOWLEDGEMENTS

I am grateful to Prof. Süleyman Özekici for introducing me to the field of computational finance. Without his guidance and patience, I would probably be lost in this thesis study. I am also grateful to Assist. Prof. İlkay Boduroğlu for his guidance in the subject of parallel computing. It was a pleasure for me to conduct my research under their supervision.

I have been working as a teaching assistant at İstanbul Kültür University for one and half years. I want to thank Assoc. Prof. Tülin Yazgaç for her support as an employer during this period of time.

I want to thank ASMA staff, Assoc. Prof. Can Özturan, Burak Gürdağ and others for helping me execute our codes on ASMA. Whenever I got stuck on ASMA, Burak Gürdağ was there to solve the problem. So, I want to thank him once more.

I am debtful to Ph.D. Orhan Feyzioğlu for his invaluable help in formatting my thesis. Without his guidance on latex, I would never have prepared my thesis on time for graduation.

Finally, I want to thank my brothers, parents and to all who really care about me.

ABSTRACT

PARALLEL COMPUTING IN ASIAN OPTION PRICING

We discuss the use of parallel computing in Asian option pricing and evaluate the efficiency of various algorithms. We only focus on "backward-starting fixed strike" continuously averaged Asian options. We implement a one-state-variable partial differential equation (P.D.E.) approach (Rogers and Shi (1995) and Alziary et al. (1997)) to price an Asian option. We also implement the same methodology to price a standard European option as the accuracy check for Asian option. We solve this parabolic P.D.E. by using both explicit and Crank-Nicolson implicit finite-difference methods. Then, we look for algorithms designed for implementing these computations in parallel. Finally, we evaluate all the algorithms by comparing the numerical results with respect to accuracy and wall-clock time of code executions. Codes are executed on Advanced System for Multi-Computer Applications (ASMA) Linux PC cluster. ASMA is located in the Department of Computer Engineering in Boğaziçi University, Turkey.

ÖZET

ASYA OPSİYONU FİYATLANDIRILMASINDA PARALEL HESAPLAMANIN KULLANILMASI

Bu tezde Asya opsiyonu fiyatlandırılmasında paralel hesaplamanın kullanılması incelenmiş ve çeşitli algoritmaların mukayeseli verimlilik analizi yapılmıştır. Sadece geriden-başlamalı, sabit-son-fiyatlı ve sürekli-ortalanan Asya opsiyonları ele alınmıştır. Rogers ve Shi (1995) ile Alziary v.d. (1997) makalelerinde anlatılan bir durum değişkenli kısmi diferansiyel denklem metodunu kullanarak Asya opsiyonları fiyatlandırılmıştır. Ayrıca, standart Avrupa opsiyonunu da Asya opsiyonunun fiyatlandırılmasında yapılan hatayı kontrol etmek amacıyla aynı yöntemlerle fiyatlandırılmıştır. Elde edilen kısmi diferansiyel denklem doğrudan hesaplama ve Crank-Nicolson sonlu-farklar yöntemleri ile çözülmüştür. Sonraki aşamada bu hesaplamaları paralel olarak yapabilecek algoritmalar araştırılmıştır. En sonunda, bütün algoritmalar elde edilen sonuçların doğruluğu ve kodların çalışma zamanları dikkate alınarak değerlendirilmiştir. Kodlar Boğaziçi Üniversitesi Bilgisayar Mühendisliği Bölümüne bağlı olan Gelişmiş Çoklu-Bilgisayar Uygulamaları (ASMA) projesinin Linux PC öbeğinde çalıştırılmıştır.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS/ABBREVIATIONS

C_t	Option price at time t
C_t^a	Asian Call option price at time t
C_t^e	European Call option price at time t
$\hat{C}_t^a$	First derivative of relative Asian option price w.r.t. t
$\hat{C}_x^a$	First derivative of relative Asian option price with w.r.t. x
$\hat{C}_{xx}^a$	Second derivative of relative Asian option price w.r.t. x
$\hat{C}_t^e$	First derivative of relative European option price w.r.t. t
$\hat{C}_y^e$	First derivative of relative European option price w.r.t. y
$\hat{C}_{yy}^e$	Second derivative of relative European option price w.r.t. y
$\check{C}_t^a$	First derivative of relative Asian option price w.r.t. t
$\check{C}_y^a$	First derivative of relative Asian option price w.r.t. y
$\check{C}_{yy}^a$	Second derivative of relative Asian option price w.r.t. y
$\check{C}_t^e$	First derivative of relative European option price w.r.t. t
$\check{C}_z^e$	First derivative of relative European option price w.r.t. z
$\check{C}_{zz}^e$	Second derivative of relative European option price w.r.t. z
e	Exponential coefficient
E	Percent error done in European option pricing
E^Q	Expectation under measure Q
E_t^Q	Expectation under measure Q at time t
F_t	Filtration at time t
K	Strike price
M	Number of time intervals used
N	Number y intervals used
p	Number of processors used
P_i	Processor i
P_t^a	Asian Put option price at time t
Q	Risk neutral measure
Q^S	A measure
r	Interest rate

S_t	Stock price at time t
S_0	Initial stock price
t	Time
$\triangle t$	Finite-difference time increment
$\bar{t}_A$	Average time spent on pricing of Asian option
T	Maturity time
V_t	Value of an option at time t
$\triangle y$	Finite-difference y increment
β_{in}	Power law exponents of in-degree distribution
β_{out}	Power law exponents of out-degree
$\rho(B)$	Spectral radius of the Jacobian iteration matrix
σ	Volatility
Φ	Cumulative standard normal function
ASMA	Advanced System for Multi-Computer Applications
LB	Lower bound price of Asian options by Rogers and Shi (1995)
max	Maximum of
min	Minimum of
PDE	Initial price of Asian options by Zvan et al. (1997)
P.D.E.	Partial differential equation
PL	Power law
RNI	Initial price of Asian options by Lim (2002)
w.r.t.	With respect to

1. INTRODUCTION

An option is a derivative security which gives its holder the right to receive a contingent payoff within a specified period of time. Exotic options, which have more complicated payoffs than standard European and American options are extensively used in finance to meet specific needs of customers. As it is expected, they are not easy to price because of complicated payoffs. Thus, they generally do not have an explicit formula giving the price of the option. Asian options are one of the path-dependent exotic options. Path-dependent means that the payoff of an Asian option depends on the path followed by the underlying security.

Our aim in this thesis study is to demonstrate how parallel computing can be used in pricing of a European-style Asian option for which early exercise is not possible. We only focus on "backward-starting fixed strike", continuously averaged Asian options. There has been many different approaches to price an Asian option. These are given in the literature review part in detail. As one of these approaches, Rogers and Shi (1995) and Alziary *et al.* (1997) propose a one-state-variable partial differential equation (P.D.E.) to be solved by numerical methods such as finite-difference methods, Monte Carlo simulation, etc. This approach is quite appropriate for our aim, because it is easier to solve the one-state-variable P.D.E.. We prefer to use finite-difference methods to solve this one-state-variable P.D.E. because it requires fewer computations than Monte Carlo simulation to attain similar accuracy (Alziary *et al.*,1997).

Rogers and Shi (1995) uses the Numerical Algorithms Group's (NAG) routine (D03PAF) to solve the parabolic P.D.E.. NAG develops and provides software to solve complex mathematical problems. On the other hand, Alziary *et al.* (1997) uses explicit finite-difference method to solve the same parabolic P.D.E.. We solve this parabolic P.D.E. by using both explicit and Crank-Nicolson implicit finite-difference methods. Then, we look for algorithms designed for implementing these computations in parallel. If an algorithm does not exist in literature as in the case of explicit finite-difference method, it is not difficult to come up with one. On the other hand, for implicit finite-

difference method what we do is to implement the parallel algorithms designed for solving system of linear equations to our specific problem. Finally, we compare all the algorithms and access the efficiency of parallel computing. In this comparison, our criterion is the wall-clock times of algorithms and accuracy of the computed values. As the accuracy check, we use the percent error done in European option pricing if we had used the same finite-spacings used in Asian option pricing like Alziary *et al.* (1997). One more check is provided by a comparison of prices with the evaluated prices by Rogers and Shi (1995) and Zvan *et al.* (1997).

We first give a detailed literature survey about different approaches to price an Asian option and parallel algorithms in solving system of linear equations in Section 2. Then, in Section 3, a general framework is given to reveal the assumptions in forming the one-state-variable P.D.E. approach. Then, basics of this one-state-variable P.D.E. approach is introduced in Section 4. In Section 5, numerical implementation details of different algorithms are introduced. Finally, in Section 6, the comparison of all of the algorithms are presented.

2. LITERATURE REVIEW

Financial market instruments can be categorized into two different types. On one side the 'underlying' stocks: shares, bonds, commodities, foreign currencies; and on the other side their 'derivatives'. Derivatives' payoff depends on underlying stock's behavior in time. Although they may be used for the risk reducing purposes, they may also magnify the risks. There is no limitation on designing novel derivatives to meet specific needs of clients. In this respect, there is a great number of derivative securities currently sold by investment banks such as futures contract, European option, American option, Asian option, Lookback option, etc. Although some derivatives can be priced analytically such as European call option, for others there may not be an exact analytical solution. Path-dependent options for which the payoff depends on the path followed by the underlying security are from the second class according to this criterion. Our interest in this thesis study is on one of the path-dependent options; namely Asian option.

Arithmetic Asian or average price (rate) options' payoff depends on the average price of the underlying security over a pre-determined time period. Asian options have a great number of applications in currency, equity, interest rate, commodity, energy and insurance markets. There are many reasons why Asian options have a wide variety of applications in different markets. First of all, corporations examine the average price realized over the accounting period for the security when preparing long-term financial estimations. So Asian options are natural risk reducing tools. In addition to that Asian options are less expensive than others, since the volatility of an average price of a commodity is smaller than the volatility of price itself. As an example for the world-wide corporations' usage of Asian options; Microsoft uses average-rate foreign exchange options to hedge its revenues from the foreign countries. Second reason is that it is not an easy thing to manipulate the price of a commodity during its time period. Thus many commodity contracts are Asian-style such as crude oil futures and options on the NYMEX and copper and aluminium options on the London Metals Exchange.

As indicated before, Asian options are path-dependent options. This makes it difficult to price analytically. The price of an Asian option at a point in time is dependent on both the price of the underlying asset at that time and the average of the price up to that time. In addition to that, the arithmetic average is not lognormally distributed when Black-Scholes-Merton pricing framework is used (i.e., the underlying security follows the geometric Brownian motion). In reality, the distribution can not be characterized analytically.

In the literature, different approaches are implemented to price an Asian option. These can be categorized under three headings; analytical pricing formulae, analytical approximations, and numerical methods. Linetsky (2001) states that three analytical pricing formulae are available for continuously-averaged Asian options. Yor (1992) expresses Asian option price as a triple integral. Linetsky (2001) reports that this triple integral is hard to evaluate numerically so it is not a practical approach. Geman and Yor (1992) derive a closed-form expression for the Laplace transform of the Asian call option price. This Laplace transform has to be inverted numerically by applying a suitable numerical Laplace inversion algorithm. In a recent contribution, Dufresne (2000) obtains an alternative formula that expresses the Asian option price as an infinite series of Laguerre polynomials with each coefficient of the series given by a single integral that needs to be computed numerically.

Various analytical approximations are suggested to approximate the distribution of the arithmetic average. Lognormal approximation with matched first and second moments of Turnbull and Wakeman (1992) and the reciprocal Gamma approximation of Milevsky and Posner (1998) are two of them. In general, the problem with analytical approximations is that in many cases no reliable error bounds or estimates are available, thus, the approximation is uncontrolled. The approximation may work well for some parameter values, while it may fail for others.

Finally, numerical methods, such as Monte Carlo simulation and numerical finite-difference P.D.E. methods are applied to price an Asian option. Boyle and Emanuel (1980), Kemna and Vorst (1990), Boyle *et al.* (1997) are the papers that implements

Monte Carlo simulation in Asian option pricing. In option pricing, usage of numerical finite-difference P.D.E. methods starts with Schwartz (1977) and Brennan and Schwartz (1978), which introduces the application of explicit and implicit numerical finite-difference methods on the standard Black-Scholes equation. Then, Courtadon (1982) proposes a more accurate finite-difference approximation for the valuation of options; namely Crank-Nicolson (or mixed) finite-difference method. Hull and White (1990) suggest a modification in the application of explicit finite-difference method. Kemna and Vorst (1990) establish a P.D.E. with two state variables (the stock price and the average) which characterize price of a "fixed-strike" Asian option. However, the pricing of Asian options is complicated by using two state variables because of extra boundary conditions and computationally difficult numerical methods to solve P.D.E.. Independently, Rogers and Shi (1995) and Alziary *et al.* (1997) propose a one-state-variable P.D.E., easy to implement to price fixed and floating strike European-style Asian options. Ingersoll (1987) is first to use this strategy to price floating-strike Asian options. Both papers then use numerical finite-difference P.D.E. methods to solve this one-state-variable P.D.E. Rogers and Shi (1995) also gives lower and upper bounds for the Asian option price. In addition to that Alziary *et al.* (1997) gives additional information on the hedging strategy. Vecer (2000, 2001) are among the papers implementing numerical finite-difference P.D.E. methods for Asian option pricing.

Although computers' speed increases day by day, we still want to see what efficiencies we get by using parallel computers in the computations. In parallel programming, usage of functions to make the communications between parallel processors is popular even if high order languages for parallel programming has been created. We use Message Passing Interface functions to make the communications between parallel processors for our parallel codes. Bunnin *et al.* (2000) discuss the usage of parallel computing in option pricing. It implements Monte-Carlo simulation method to price discretely sampled Asian option in parallel. Furthermore, Pseudespectral Chebyshev method is applied to price standard European call option in parallel as the second example.

Converting the explicit finite-difference code to parallel code is an easy task in contrast to the implicit one, in which we solve system of linear equations. For our problem, the system of linear equations structure is tridiagonal as we show in the next sections. Stone (1973) proposes the recursive doubling algorithm as an efficient parallel algorithm for the solution of a tridiagonal system of linear equations. Then, Wang (1981) introduces the partition algorithm for the same problem. Then, Parallel Factorization algorithm is proposed in Amodio and Brugnano (1992). Finally, Mattor *et al.* (1995) improves the Parallel Factorization algorithm as the best algorithm ever proposed for solving large tridiagonal matrix problems in parallel. It is a coarse grained algorithm because there are no send-receive relationships between processors until the coupling of solutions in all of the processors. Lin (2001) uses a unified graph model to represent most of the algorithms, such as Cyclic Elimination, Cyclic Reduction and Recursive Doubling algorithms for the parallelization of system of linear equations.

3. FEATURES OF ASIAN OPTIONS

In this section, features of Asian options are introduced. In addition to that underlying assumptions and models are given to provide a base for pricing methods. A continuous time economy where uncertainty is represented by a probability space (A, F, P) is considered on time interval $[0, T]$. Continuous filtration $(F_t)_{0 \leq t \leq T}$ represent the flow of information on time interval $[0, T]$. The economy is formed out of two assets, a stock S, and a zero coupon bond maturing at time T of constant interest rate r. The stock price follows a geometric Brownian motion. There exists a probability Q under which the discounted price of the stock, $(e^{-rt}S_t)$, is a martingale because of the assumption that absence of arbitrage opportunities exist. Since we suppose that the market is complete, the probability Q is unique and called the risk neutral probability (Duffie (1985)). Under these assumptions, Harrison and Kreps (1979) and Harrison and Pliska (1981) states that

The stock's price follows the stochastic differential equation

$$dS_t = S_t r dt + S_t \sigma d\omega_t$$

where ω_t is a Wiener process (standard Brownian motion) under the measure Q.

The price of any option is the discounted expectation of its terminal payoff (at date T) under Q.

Two different types of Asian options are to be written using this general framework on the parameters of stock's price S, maturity time T and the time interval $[t_0, T]$ over which the average value of the stock is considered.

The "fixed-strike" Asian call option's payoff at date T is

$$\left(\frac{1}{T}\int_{t_o}^{T} S_u du - K\right)^+,$$

where K is the given strike price and $a^+ = \max(0, a)$ for any real number a.

The "floating-strike" Asian call option's payoff at date T is

$$\left(S_T - \frac{1}{T}\int_{t_o}^{T} S_u du\right)^+.$$

Thus, if we want to price these options we should calculate the following expectations under the risk neutral probability Q:

$$C_t^a \stackrel{def}{=} E_t^Q\left[e^{-r(T-t)}\left(\frac{1}{T}\int_{t_o}^{T} S_u du - K\right)^+\right],$$

and

$$C_t^a \stackrel{def}{=} E_t^Q\left[e^{-r(T-t)}\left(S_T - \frac{1}{T}\int_{t_o}^{T} S_u du\right)^+\right],$$

where we use the notation $E_t^Q[X] = E^Q[X \mid F_t]$ for any random variable X. Moreover, E^Q denotes the fact that the expectation is taken with respect to the measure Q.

Since $\int_{t_o}^{T} S_u du$ is a mix of lognormal random variables, these expectations are hard to evaluate. Our aim is to price European-style Asian options which can be exercised only at maturity date. There is one more classification for Asian options according to whether or not the time to maturity is less than or greater than the length of the averaging period ($0 \leq t_0 \leq T$). First case is called "backward-starting" where ($0 \leq t_0 \leq t \leq T$) and second case is called "forward-starting" where ($0 \leq t \leq t_0 \leq T$). In our thesis study, we only focus on the case where the time to maturity is less than or equal to the length of the averaging period (that is "backward-starting" Asian option case). We price an European-style, "backward-starting fixed-strike" Asian call

option. Alziary *et al.* (1997) derives a put-call parity equation using Fubini's theorem for "fixed-strike" Asian call option. To simplify the notation they take t_0 as zero. If C_t^a(resp. P_t^a) denote the price at date t of a "fixed-strike" Asian call (resp. put) option then

$$P_t^a = C_t^a - \frac{S_t}{Tr}\left(1 - e^{-r(T-t)}\right) + e^{-r(T-t)}\left(K - \frac{1}{T}\int_0^t S_u du\right). \tag{3.1}$$

If the known part of the average is greater than the exercise price (i.e., $K - \frac{1}{T}\int_0^t S_u du < 0$), then Asian put is worthless ($P_t^a = 0$). So Equation (3.1) directly gives the price for Asian call option as

$$C_t^a = \frac{S_t}{Tr}\left(1 - e^{-r(T-t)}\right) - e^{-r(T-t)}\left(K - \frac{1}{T}\int_0^t S_u du\right) \tag{3.2}$$

when $P_t^a = 0$.

4. THE P.D.E. APPROACH

We use the one-state-variable P.D.E. approach proposed by Rogers and Shi (1995) and Alziary *et al.* (1997) to price a European-style, "backward-starting fixed-strike" Asian call option. We now introduce the basics of this approach before we discuss the numerical methods to solve these partial differential equations.

4.1. Pricing an Asian Option

For "fixed-strike" Asian Call option, the pricing equation is stated as

$$C_t^a = E_t^Q \left[e^{-r(T-t)} \left(\frac{1}{T} \int_0^T S_u du - K \right)^+ \right]$$

in previous section. After including a new variable S_T and changing the appearance, it becomes

$$C_t^a = e^{-r(T-t)} E_t^Q \left[S_T \left(\frac{K - \frac{1}{T}\int_0^T S_u du}{S_T} \right)^- \right] \tag{4.1}$$

where $a^- = \min(0, a)$ for any real number a.

Note that

$$E^Q [S_T] = E^Q [S_T \mid F_0] = e^{rT} E^Q \left[e^{-rT} S_T \mid F_0 \right]$$

Since $e^{-rt} S_t$ is martingale under measure Q,

$$E^Q \left[e^{-rT} S_T \mid F_0 \right] = S_0$$

and

$$E^Q[S_T] = e^{rT}S_0.$$

Let Q^S be defined by its Radon-Nikodym derivative with respect to the risk neutral measure Q such that

$$\frac{dQ^S}{dQ} = \frac{S_T}{e^{rT}S_0}.$$

We know that

$$E^{Q^S}[X_T \mid F_t] = \frac{E^Q[\xi_T X_T \mid F_t]}{\xi_t} \tag{4.2}$$

where

$$\begin{aligned}\xi_t &= E^Q\left[\frac{dQ^S}{dQ} \mid F_t\right] \\ &= E^Q\left[\frac{S_T}{e^{rT}S_0} \mid F_t\right] \\ &= \frac{1}{S_0}E^Q\left[e^{-rT}S_T \mid F_t\right].\end{aligned}$$

Since $e^{-rt}S_t$ is martingale under measure Q, $E^Q\left[e^{-rT}S_T \mid F_t\right] = e^{-rt}S_t$. Then,

$$\xi_t = \frac{e^{-rt}S_t}{S_0}$$

and

$$\xi_T = \frac{e^{-rT}S_T}{S_0}.$$

If we turn back to the Equation (4.1),

$$
\begin{aligned}
C_t^a &= e^{-r(T-t)} E_t^Q \left[\frac{S_T S_0 e^{rT}}{S_0 e^{rT}} \left(\frac{K - \frac{1}{T}\int_0^T S_u du}{S_T} \right)^- \right] \\
&= e^{-r(T-t)} S_0 e^{rT} E_t^Q \left[\xi_T \left(\frac{K - \frac{1}{T}\int_0^T S_u du}{S_T} \right)^- \right] \\
&= e^{rt} S_0 E_t^Q \left[\xi_T \left(\frac{K - \frac{1}{T}\int_0^T S_u du}{S_T} \right)^- \right] \\
&= \frac{e^{rt} S_0 S_t}{S_t} E_t^Q \left[\xi_T \left(\frac{K - \frac{1}{T}\int_0^T S_u du}{S_T} \right)^- \right] \\
&= \frac{S_t}{\xi_t} E_t^Q \left[\xi_T \left(\frac{K - \frac{1}{T}\int_0^T S_u du}{S_T} \right)^- \right].
\end{aligned}
$$

By using Equation (4.2), this equation can be written as

$$
C_t^a = S_t E_t^{Q^S} \left[\left(\frac{K - \frac{1}{T}\int_0^T S_u du}{S_T} \right)^- \right].
$$

If we define a new variable $x_t = \left[K - (1/T)\int_0^t S_u du\right]/S_t$ then

$$
C_t^a / S_t = E_t^{Q^S}[x_T^-]. \tag{4.3}
$$

The next step is to characterize $E_t^{Q^S}\left[x_T^-\right]$ (i.e., the relative price of Asian option) by a one-state-variable P.D.E.

It is obvious that $x_t = f(S_t, z_t)$, where $z_t = \int_0^t S_u du$. So, taking $y_1 = S_t$ and

$y_2 = z_t$, we can write

$$\frac{\partial f}{\partial y_1} = \hat{C}^a(x_t, t), \frac{\partial f}{\partial y_2} = S_t,$$
$$\frac{\partial^2 f}{\partial y_1^2} = \frac{\partial^2 f}{\partial y_2^2} = 0,$$
$$\frac{\partial}{\partial y_1}\frac{\partial f}{\partial y_2} = \frac{\partial}{\partial y_2}\frac{\partial f}{\partial y_1} = 1.$$

and

$$dS_t dS_t = S_t^2 \sigma^2 dt, d_{z_t} dS_t = 0.$$

By Itô's formula;

$$dx_t = \frac{-1}{S_t^2}\left[K - \left(\frac{1}{T}\right)\int_0^t S_u du\right](S_t r dt + S_t \sigma d\omega_t) - \frac{1}{S_t T} S_t dt + \frac{1}{2}\frac{2}{S_t^3}\left[K - \left(\frac{1}{T}\right)\int_0^t S_u du\right] S_t^2 \sigma^2 dt$$
$$dx_t = \left(-\frac{1}{T} - r x_t + x_t \sigma^2\right) dt - \sigma x_t d\omega_t.$$

The price C_t^a at date t of a "fixed-strike" Asian call option is given by

$$V_t = C_t^a = S_t \hat{C}^a(x_t, t)$$

where $\hat{C}^a(x_t, t) = E_t^{Q^S}[x_T^-]$ from Equation(4.3).

It is obvious that $V_t = f(S_t, \hat{C}^a(x_t, t))$, so taking $y_1 = S_t$ and $y_2 = \hat{C}^a(x_t, t)$, we obtain

$$\frac{\partial f}{\partial y_1} = \hat{C}^a(x_t, t), \frac{\partial f}{\partial y_2} = S_t,$$
$$\frac{\partial^2 f}{\partial y_1^2} = \frac{\partial^2 f}{\partial y_2^2} = 0,$$
$$\frac{\partial}{\partial y_1}\frac{\partial f}{\partial y_2} = \frac{\partial}{\partial y_2}\frac{\partial f}{\partial y_1} = 1.$$

By Itô's formula

$$dV_t = \hat{C}^a(x_t, t)dS_t + S_t d\hat{C}^a(x_t, t) + \frac{1}{2}2(1)dS_t d\hat{C}^a(x_t, t).$$

After simplifications we get

$$dV_t = \hat{C}^a(x_t, t)dS_t + S_t d\hat{C}^a(x_t, t) + dS_t d\hat{C}^a(x_t, t) \tag{4.4}$$

where x_t satisfies

$$dx_t = \left(-\frac{1}{T} - rx_t + x_t\sigma^2\right)dt - \sigma x_t d\omega_t.$$

Note that $\hat{C}^a(x_t, t) = f(x_t, t)$, so taking $y_1 = x_t$ and $y_2 = t$, we obtain

$$\frac{\partial f}{\partial y_1} = \hat{C}^a_x(x_t, t), \frac{\partial f}{\partial y_2} = \hat{C}^a_t(x_t, t),$$
$$\frac{\partial^2 f}{\partial y_1^2} = \hat{C}^a_{xx}(x_t, t),$$
$$dx_t dx_t = \sigma^2 x_t^2 dt.$$

By Itô's formula

$$d\hat{C}^a(x_t, t) = \hat{C}^a_x(x_t, t)dx_t + \hat{C}^a_t(x_t, t)dt + \frac{1}{2}\hat{C}^a_{xx}(x_t, t)\sigma^2 x_t^2 dt$$

$$= \hat{C}^a_x(x_t,t)\left[\left(-\frac{1}{T} - rx_t + x_t\sigma^2\right)dt - \sigma x_t d\omega_t\right] + \hat{C}^a_t(x_t,t)dt + \frac{1}{2}\hat{C}^a_{xx}(x_t,t)\sigma^2 x_t^2 dt$$

$$= \left[\hat{C}^a_x(x_t,t)\left(-\frac{1}{T} - rx_t + \sigma^2 x_t\right) + \hat{C}^a_t(x_t,t) + \frac{1}{2}\hat{C}^a_{xx}(x_t,t)\sigma^2 x_t^2\right]dt - \hat{C}^a_x(x_t,t)\sigma x_t d\omega_t$$

Then, if we turn back to Equation (4.4),

$$\begin{aligned} dV_t = \quad & \hat{C}^a(x_t,t)\left(S_t r dt + S_t\sigma d\omega_t\right) + \\ & + S_t\left[\left[\hat{C}^a_x(x_t,t)\left(-\frac{1}{T} - rx_t + x_t\sigma^2\right) + \hat{C}^a_t(x_t,t) + \frac{1}{2}\hat{C}^a_{xx}(x_t,t)\sigma^2 x_t^2\right]dt \right. \\ & \left. - \hat{C}^a_x(x_t,t)\sigma x_t d\omega_t\right] - \hat{C}^a_x(x_t,t)\sigma^2 S_t x_t dt. \end{aligned}$$

After simplifications we get

$$\begin{aligned} dV_t = & \left(\hat{C}^a(x_t,t)S_t r + S_t\left[\hat{C}^a_x(x_t,t)\left(-\frac{1}{T} - rx_t + x_t\sigma^2\right) + \hat{C}^a_t(x_t,t)\right.\right. \\ & \left.\left. + \frac{1}{2}\hat{C}^a_{xx}(x_t,t)\sigma^2 x_t^2\right]\right)dt - \hat{C}^a_x(x_t,t)\sigma^2 S_t x_t dt + \left(\hat{C}^a(x_t,t)S_t\sigma - S_t\hat{C}^a_x(x_t,t)\sigma x_t\right)d\omega_t. \end{aligned} \tag{4.5}$$

Since there is a self-financing trading strategy (ϕ_t, ψ_t) replicating an Asian option such that

$$dV_t = \phi_t dS_t + \psi_t dB_t$$

where $B_t = e^{rt}$ and $V_t = B_t\psi_t + \phi_t S_t = S_t\hat{C}^a(x_t,t)$. We can write

$$\begin{aligned} dV_t &= \phi_t\left(S_t r dt + S_t\sigma d\omega_t\right) + \psi_t r B_t dt \\ &= \left(\phi_t S_t r + \psi_t r B_t\right)dt + \phi_t S_t\sigma d\omega_t \\ &= rS_t\hat{C}^a(x_t,t)dt + \phi_t S_t\sigma d\omega_t. \end{aligned} \tag{4.6}$$

Since Equations (4.5) and (4.6) are equal

$$\phi_t S_t \sigma = \hat{C}^a(x_t, t) S_t \sigma - S_t \hat{C}^a_x(x_t, t) \sigma x_t$$

and we obtain

$$\phi_t = \hat{C}^a(x_t, t) - \hat{C}^a_x(x_t, t) x_t.$$

Then, it is easy to see that

$$\psi_t = \frac{1}{B_t}\left(-\phi_t S_t + S_t \hat{C}^a(x_t, t)\right) = \frac{1}{B_t} S_t \hat{C}^a_x(x_t, t) x_t.$$

And, finally

$$\hat{C}^a(x_t, t) S_t r + S_t \left[\hat{C}^a_x(x_t, t)\left(-\frac{1}{T} - r x_t + x_t \sigma^2\right) + \hat{C}^a_t(x_t, t) + \frac{1}{2}\hat{C}^a_{xx}(x_t, t)\sigma^2 x_t^2\right]$$

$$-\hat{C}^a_x(x_t, t)\sigma^2 S_t x_t = S_t \hat{C}^a(x_t, t) r$$

which implies

$$\left[\hat{C}^a_x(x_t, t)\left(-\frac{1}{T} - r x_t + x_t \sigma^2\right) + \hat{C}^a_t(x_t, t) + \frac{1}{2}\hat{C}^a_{xx}(x_t, t)\sigma^2 x_t^2\right] - \hat{C}^a_x(x_t, t)\sigma^2 x_t = 0.$$

So the resultant P.D.E. is

$$\hat{C}^a_x(x, t)\left(-\frac{1}{T} - r x\right) + \hat{C}^a_t(x, t) + \frac{1}{2}\hat{C}^a_{xx}(x, t)\sigma^2 x^2 = 0 \tag{4.7}$$

with the boundary condition

$$\hat{C}^a(x,T) = x^-$$

which holds in the domain $D = \{(x,t) : x \epsilon R, 0 \leq t \leq T\}$.

In Section 3, it was stated that if the known part of the average is greater than the exercise price (i.e.,$K - \frac{1}{T}\int_0^t S_u du < 0$) (or equivalently $x_t \leq 0$) at the pricing date, explicit solution is available as given in Equation (3.2). So, if we simply include the new variable x_t into Equation (3.2) then

$$\hat{C}_t^a = \frac{1}{Tr}\left(1 - e^{-r(T-t)}\right) - e^{-r(T-t)}x_t \tag{4.8}$$

Equation (4.8) will be useful in the numerical implementation of P.D.E. Alziary *et al.* (1997) implement the P.D.E. on $\mathbb{R}^+$ with Equation (4.8) as a boundary condition at $x_t = 0$.

4.2. Pricing a European Option

We follow the same procedure with the Asian option to get the P.D.E. for the European option stated without proof in Alziary *et al.* (1997).

Let's start with simple definition of European call option pricing

$$\begin{aligned} C_t^e &= E_t^Q\left[e^{-r(T-t)}(S_T - K)^+\right] \\ &= e^{-r(T-t)}E_t^Q\left[S_T(1 - \frac{K}{S_T})^+\right] \end{aligned}$$

$$\begin{aligned} C_t^e &= e^{-r(T-t)} E_t^Q \left[\frac{S_T}{e^{rT} S_0} e^{rT} S_0 (1 - \frac{K}{S_T})^+ \right] \\ &= S_0 e^{rt} E_t^Q \left[\frac{S_T}{e^{rT} S_0} (1 - \frac{K}{S_T})^+ \right] \\ &= \frac{S_0 e^{rt}}{S_t} S_t E_t^Q \left[\xi_T (1 - \frac{K}{S_T})^+ \right] \\ &= \frac{S_t}{\xi_t} E_t^Q \left[\xi_T (1 - \frac{K}{S_T})^+ \right]. \end{aligned}$$

So, if we remember Equation (4.2), then

$$C_t^e = S_t E_t^{Q^S} \left[(1 - \frac{K}{S_T})^+ \right].$$

If we define a variable $y_t = \frac{K}{S_t}$, then

$$C_t^e = S_t E_t^{Q^S} \left[(1 - y_T)^+ \right].$$

It is obvious that $y_t = f(S_t)$, so taking $y_1 = S_t$, we can write

$$\begin{aligned} \frac{\partial f}{\partial y_1} &= \frac{-K}{S_t^2} \\ \frac{\partial^2 f}{\partial y_1^2} &= \frac{2K}{S_t^3} \\ dS_t dS_t &= S_t^2 \sigma^2 dt. \end{aligned}$$

By Itô's formula,

$$dy_t = \frac{-K}{S_t^2} dS_t + \frac{1}{2} \frac{2K}{S_t^3} S_t^2 \sigma^2 dt$$

$$\begin{aligned} dy_t &= \frac{-K}{S_t^2}(S_t r dt + S_t \sigma d\omega_t) + \frac{1}{2}\frac{2K}{S_t^3} S_t^2 \sigma^2 dt \\ &= (\frac{K}{S_t}\sigma^2 - \frac{K}{S_t} r)dt - \frac{K}{S_t}\sigma d\omega_t \\ &= (\sigma^2 y_t - r y_t)dt - \sigma y_t d\omega_t. \end{aligned}$$

The price C_t^e at date t of a European call option is given by

$$V_t = C_t^e = S_t \hat{C}^e(y_t, t).$$

It is obvious that $V_t = f(S_t, \hat{C}^e(y_t, t))$, so taking $x_1 = S_t$ and $x_2 = \hat{C}^e(y_t, t)$, we can write

$$\frac{\partial f}{\partial x_1} = \hat{C}^e(y_t, t), \frac{\partial f}{\partial x_2} = S_t,$$

$$\frac{\partial^2 f}{\partial x_1^2} = \frac{\partial^2 f}{\partial x_2^2} = 0,$$

and

$$\frac{\partial}{\partial x_1}\frac{\partial f}{\partial x_2} = \frac{\partial}{\partial x_2}\frac{\partial f}{\partial x_1} = 1.$$

By Itô's formula

$$dV_t = \hat{C}^e(y_t, t) dS_t + S_t d\hat{C}^e(y_t, t) + dS_t d\hat{C}^e(y_t, t) \tag{4.9}$$

where y_t satisfies

$$dy_t = (\sigma^2 y_t - r y_t)dt - \sigma y_t d\omega_t$$

Note that $\hat{C}^e(y_t, t) = f(y_t, t)$, so taking $x_1 = y_t$ and $x_2 = t$, we can write

$$\frac{\partial f}{\partial x_1} = \hat{C}^e_y(y_t, t), \frac{\partial f}{\partial x_2} = \hat{C}^e_t(y_t, t),$$

$$\frac{\partial^2 f}{\partial x_1^2} = \hat{C}^e_{yy}(y_t, t)$$

and

$$dy_t dy_t = \sigma^2 y_t^2 dt.$$

By Itô's formula

$$\begin{aligned} d\hat{C}^e(y_t, t) &= \hat{C}^e_y(y_t, t)dy_t + \hat{C}^e_t(y_t, t)dt + \frac{1}{2}\hat{C}^e_{yy}(y_t, t)\sigma^2 y_t^2 dt \\ &= \hat{C}^e_y(y_t, t)\left[(\sigma^2 y_t - r y_t)dt - \sigma y_t d\omega_t\right] + \hat{C}^e_t(y_t, t)dt + \frac{1}{2}\hat{C}^e_{yy}(y_t, t)\sigma^2 y_t^2 dt \\ &= \left[\hat{C}^e_y(y_t, t)(\sigma^2 y_t - r y_t) + \hat{C}^e_t(y_t, t) + \frac{1}{2}\hat{C}^e_{yy}(y_t, t)\sigma^2 y_t^2\right] dt - \sigma y_t \hat{C}^e_y(y_t, t)d\omega_t. \end{aligned}$$

If we turn back to Equation (4.9), then

$$dV_t = \hat{C}^e(y_t, t)dS_t + S_t d\hat{C}^e(y_t, t) + dS_t d\hat{C}^e(y_t, t)$$

$$
\begin{aligned}
dV_t =& \hat{C}^e(y_t,t)\left(S_t r dt + S_t \sigma d\omega_t\right) + \\
&+ S_t\left[\left(\hat{C}^e_y(y_t,t)(\sigma^2 y_t - r y_t) + \hat{C}^e_t(y_t,t) + \frac{1}{2}\hat{C}^e_{yy}(y_t,t)\sigma^2 y_t^2\right)dt - \sigma y_t \hat{C}^e_y(y_t,t)d\omega_t\right] \\
&- \sigma^2 y_t S_t \hat{C}^e_y(y_t,t)dt.
\end{aligned}
$$

After simplifications we get

$$
\begin{aligned}
dV_t =& \left[\hat{C}^e(y_t,t)S_t r + S_t\left(\hat{C}^e_y(y_t,t)(\sigma^2 y_t - r y_t) + \hat{C}^e_t(y_t,t) + \frac{1}{2}\hat{C}^e_{yy}(y_t,t)\sigma^2 y_t^2\right)\right. \\
&\left. -\sigma^2 y_t S_t \hat{C}^e_y(y_t,t)\right]dt + \left[S_t\sigma\hat{C}^e(y_t,t) - \sigma y_t S_t \hat{C}^e_y(y_t,t)\right]d\omega_t. \qquad (4.10)
\end{aligned}
$$

Since there is a self-financing trading strategy (ϕ_t, ψ_t) replicating a European option, we can write

$$
dV_t = r S_t \hat{C}^e(x_t,t)dt + \phi_t S_t \sigma d\omega_t \qquad (4.11)
$$

by using the same procedure used in Asian option pricing.

Since Equations (4.10) and (4.11) are equal,

$$
\hat{C}^e(y_t,t)S_t r + S_t\left[\hat{C}^e_y(y_t,t)(\sigma^2 y_t - r y_t) + \hat{C}^e_t(y_t,t) + \frac{1}{2}\hat{C}^e_{yy}(y_t,t)\sigma^2 y_t^2\right] -
$$

$$
-\sigma^2 y_t S_t \hat{C}^e_y(y_t,t) = S_t \hat{C}^e(y_t,t) r
$$

which implies

$$
\left[\hat{C}^e_y(y_t,t)(\sigma^2 y_t - r y_t) + \hat{C}^e_t(y_t,t) + \frac{1}{2}\hat{C}^e_{yy}(y_t,t)\sigma^2 y_t^2\right] - \sigma^2 y_t \hat{C}^e_y(y_t,t) = 0.
$$

So the resultant P.D.E. is

$$-ry\hat{C}_y^e(y,t) + \hat{C}_t^y(y,t) + \frac{1}{2}\sigma^2 y^2 \hat{C}_{yy}^e(y,t) = 0 \tag{4.12}$$

with boundary condition

$$\hat{C}^e(y,T) = (1-y)^+$$

which holds in the domain $D = \{(y,t) : 0 \leq y < \infty, 0 \leq t \leq T\}$.

Equation (4.12) has of course the following explicit solution

$$\hat{C}^e(y_t,t) = E_t^{Q^S}\left[(1-\frac{K}{S_T})^+\right] = \Phi(d_1) - y_t e^{-r(T-t)}\Phi(d_1 - \sigma\sqrt{T-t})$$

with

$$d_1 = \frac{1}{\sigma\sqrt{T-t}} \ln \frac{1}{y_t e^{-r(T-t)}} + \frac{1}{2}\sigma\sqrt{T-t}$$

where Φ is the cumulative standard normal function. The only difference in Equations (4.7) and (4.12) is the $1/T$ coefficient, which is a result of the "arithmetic average nature" of Asian options in Equation (4.7) (Alziary *et al.*, 1997).

5. NUMERICAL IMPLEMENTATION

We apply the same numerical approach to solve the one-state-variable P.D.E. as in Alziary *et al.* (1997). That is, we first reduce the unbounded domain to a bounded one by changing variables in the P.D.E.. Although Alziary *et al.* (1997) applies only the explicit finite-difference method to solve the P.D.E. after the variable change, we apply both explicit and Crank-Nicolson implicit finite-difference methods. In addition, we look for algorithms to parallelize these methods since our aim is to use parallel computing in Asian option pricing. So far so good, but how can we compare the accuracy of computed prices using different methods for an Asian option which does not have an explicit pricing formula. As shown in the previous section, Alziary *et al.* (1997) implements the same approach to find a one-state-variable P.D.E. (Equation (4.12)) for the European option, which has an analytic pricing formula. Then, they solve it and declare that the error occurred in Asian option pricing is close to the error occurred in European option pricing because of the similarity in Equations (4.7) and (4.12). Our accuracy check is to use this logic and compute per cent error done in European option pricing for the same finite-spacings with Asian option.

5.1. Change of Variables for an Asian Option

First, using the fact that the solution is known in the case $x \leq 0$, Alziary *et al.* (1997) consider Equation (4.7) only for $x \geq 0$ with the following boundary conditions

$$\hat{C}^a(0,t) = \frac{1}{Tr}(1 - e^{-r(T-t)}),$$

$$\lim_{x \to \infty} \hat{C}^a(x,t) = 0.$$

The boundary condition at infinity comes obviously from the definition of the Asian call.

Then, Alziary *et al.* (1997) define a new state variable $y = e^{-x}$ such that $\check{C}^a(y,t) = \hat{C}^a(-\ln y, t)$ for $x \in [0,\infty]$, $y \in [0,1]$.

So, let's introduce the new variable into Equation (4.7) to get

$$y = e^{-x} \Rightarrow dy = -ydx \text{ and } x = -\ln y,$$

$$\hat{C}^a_x(x,t)\frac{dx}{dy} = \check{C}^a_y(y,t) \Rightarrow \hat{C}^a_x(x,t) = -y\check{C}^a_y(y,t)$$

and

$$\begin{aligned}
\hat{C}^a_{xx}(x,t) &= \frac{d}{dx}\hat{C}^a_x(x,t) \\
&= -y\frac{d}{dy}\left(-y\check{C}^a_y(y,t)\right) \\
&= y\left(\check{C}^a_y(y,t) + y\check{C}^e_{yy}(y,t)\right) \\
&= y\check{C}^a_y(y,t) + y^2\check{C}^a_{yy}(y,t).
\end{aligned}$$

So Equation (4.7) changes into

$$-y\check{C}^a_y(y,t)(-\frac{1}{T} - r(-\ln y)) + \left[y\check{C}^a_y(y,t) + y^2\check{C}^a_{yy}(y,t)\right]\frac{1}{2}\sigma^2\left(-\ln y\right)^2 + \check{C}^a_t(y,t) = 0$$

and the P.D.E. becomes

$$\check{C}^a_t(y,t) + \frac{1}{2}\sigma^2\left(\ln y\right)^2 y^2\check{C}^a_{yy}(y,t) + \left[(\frac{1}{T} - r\ln y)\,y + \frac{1}{2}\sigma^2 y\,\left(\ln y\right)^2\right]\check{C}^a_y(y,t) = 0 \quad (5.1)$$

with the following boundary conditions

$$\begin{aligned}
&\check{C}^a(y,T) = 0 \\
&\check{C}^a(1,t) = \frac{1}{Tr}(1 - e^{-r(T-t)}) \\
&\check{C}^a(0,t) = 0
\end{aligned}$$

and the new P.D.E. holds on the bounded domain for $0 \leq y \leq 1$ and $0 \leq t \leq T$.

5.2. Change of Variables for a European Option

Since y is always greater than zero because of the definition, Alziary *et al.* (1997) consider Equation (4.12) only for $y \geq 0$ with the following boundary conditions:

$$\hat{C}^e(y,T) = (1-y)^+,$$

$$\lim_{y\to\infty} \hat{C}^e(y,t) = 0,$$

$$\hat{C}^e(0,t) = 1.$$

The boundary condition at infinity and zero comes obviously from the definition of the European call.

Then, Alziary *et al.* (1997) define a new state variable $z = e^{-y}$ such that $\check{C}^e(z,t) = \hat{C}^e(-\ln z, t)$ for $y \in [0,\infty]$, $z \in [0,1]$.

So, let's introduce the new variable into Equation (4.12)

$$z = e^{-y} \Rightarrow dz = -zdy \text{ and } y = -\ln z,$$

$$\hat{C}^e_y(y,t)\frac{dy}{dz} = \check{C}^e_z(z,t) \Rightarrow \hat{C}^e_y(y,t) = -z\check{C}^e_z(z,t)$$

and

$$\begin{aligned}\hat{C}^e_{yy}(y,t) &= \frac{d}{dy}\hat{C}^e_y(y,t) \\ &= -z\frac{d}{dz}\left(-z\check{C}^e_z(z,t)\right) \\ &= z\left(\check{C}^e_z(z,t) + z\check{C}^e_{zz}(z,t)\right) \\ &= z\check{C}^e_z(z,t) + z^2\check{C}^e_{zz}(z,t)\end{aligned}$$

So Equation (4.12) changes into

$$-z\check{C}^e_z(z,t)(-r(-\ln z)) + \left[z\check{C}^e_z(z,t) + z^2\check{C}^e_{zz}(z,t)\right]\frac{1}{2}\sigma^2\left(-\ln z\right)^2 + \check{C}^e_t(z,t) = 0$$

and the P.D.E. becomes

$$\check{C}^e_t(z,t) + \frac{1}{2}\sigma^2\left(\ln z\right)^2 z^2\check{C}^e_{zz}(z,t) + \left[-rz\ln z + \frac{1}{2}\sigma^2 z\left(\ln z\right)^2\right]\check{C}^e_z(z,t) = 0 \qquad (5.2)$$

with the following boundary conditions

$$\begin{aligned}\check{C}^e(z,T) &= (1+\ln z)^+ \\ \check{C}^e(0,t) &= 0 \\ \check{C}^e(1,t) &= 1\end{aligned}$$

and the new P.D.E. holds on the bounded domain $0 \le z \le 1$ and $0 \le t \le T$.

5.3. Explicit Finite-Difference Method

To implement the explicit finite-difference method, two increments $\triangle t$, and $\triangle y$ of the variables t and y are chosen. A grid is then constructed for considering values of $\check{C}^a$ when y is equal to:

$$0, \triangle y, 2\triangle y, ..., n_y \triangle y = 1,$$

and time is equal to

$$0, \triangle t, 2 \triangle\ t, ..., n_t \triangle\ t = T$$

We will denote $i \triangle\ t$ by t_i, $j \triangle y$ by y_j, and the approximation of $\check{C}^a(j \triangle y, i \triangle t)$ by $\check{C}^a_{ij}$. The partial derivatives of $\check{C}^a$ with respect to y at node $(i-1, j)$ are approximated as

$$\frac{\partial \check{C}^a}{\partial y} = \frac{\check{C}^a_{i,\,j+1} - \check{C}^a_{i,\,j}}{\triangle y}, \tag{5.3}$$

$$\frac{\partial^2 \check{C}^a}{\partial y^2} = \frac{\check{C}^a_{i,\,j+1} + \check{C}^a_{i,\,j-1} - 2\check{C}^a_{i,\,j}}{(\triangle y)^2}, \tag{5.4}$$

and the time derivative is approximated as

$$\frac{\partial \check{C}^a}{\partial t} = \frac{\check{C}^a_{i,\,j} - \check{C}^a_{i-1,\,j}}{\triangle t}. \tag{5.5}$$

Substituting Equations (5.3)—(5.5) into Equation (5.1) gives

$$\frac{\check{C}^a_{i,\,j} - \check{C}^a_{i-1,\,j}}{\triangle t} + \frac{1}{2}\sigma^2 (\ln y_j)^2 y_j^2 \left(\frac{\check{C}^a_{i,\,j+1} + \check{C}^a_{i,\,j-1} - 2\check{C}^a_{i,\,j}}{(\triangle y)^2} \right) +$$

$$+ \left[(\frac{1}{T} - r \ln y_j)\, y_j + \frac{1}{2}\sigma^2 y_j\ (\ln y_j)^2 \right] \left(\frac{\check{C}^a_{i,\,j+1} - \check{C}^a_{i,\,j}}{\triangle y} \right) = 0$$

$$\check{C}^a_{i,\,j} - \check{C}^a_{i-1,\,j} + \frac{1}{2}\sigma^2 (\ln y_j)^2 y_j^2 \triangle t \left(\frac{\check{C}^a_{i,\,j+1} + \check{C}^a_{i,\,j-1} - 2\check{C}^a_{i,\,j}}{(\triangle y)^2} \right) +$$

$$+ \left[(\frac{1}{T} - r \ln y_j)\, y_j + \frac{1}{2}\sigma^2 y_j\, (\ln y_j)^2 \right] \triangle t \left(\frac{\check{C}^a_{i,\,j+1} - \check{C}^a_{i,\,j}}{\triangle y} \right) = 0$$

$$\begin{aligned}
\check{C}^a_{i-1,\,j} = & \left[\frac{1}{2}\sigma^2 (\ln y_j)^2 y_j^2 \frac{\triangle t}{\triangle y^2} + \left((\frac{1}{T} - r \ln y_j)\, y_j + \frac{1}{2}\sigma^2 y_j\, (\ln y_j)^2 \right) \frac{\triangle t}{\triangle y} \right] \check{C}^a_{i,\,j+1} + \\
& + \left[1 - \sigma^2 (\ln y_j)^2 (y_j)^2 \frac{\triangle t}{(\triangle y)^2} - \left[(\frac{1}{T} - r \ln y_j)\, y_j + \frac{1}{2}\sigma^2 y_j\, (\ln y_j)^2 \right] \frac{\triangle t}{\triangle y} \right] \check{C}^a_{i,\,j} + \\
& + \left[\frac{1}{2}\sigma^2 (\ln y_j)^2 (y_j)^2 \frac{\triangle t}{(\triangle y)^2} \right] \check{C}^a_{i,\,j-1}
\end{aligned}$$

We can simplify the equation such that

$$\check{C}^a_{i-1,\,j} = P_{j,j-1}\check{C}^a_{i,\,j-1} + P_{j,j}\check{C}^a_{i,\,j} + P_{j,\,j+1}\check{C}^a_{i,\,j+1} \tag{5.6}$$

for j=1,..,n_y-1 and where

$$P_{j,j-1} = \frac{1}{2}\sigma^2 (\ln y_j)^2 (y_j)^2 \frac{\triangle t}{(\triangle y)^2},$$

$$P_{j,j} = 1 - \sigma^2 (\ln y_j)^2 (y_j)^2 \frac{\triangle t}{(\triangle y)^2} - \left[(\frac{1}{T} - r \ln y_j)\, y_j + \frac{1}{2}\sigma^2 y_j\, (\ln y_j)^2 \right] \frac{\triangle t}{\triangle y},$$

$$P_{j,j+1} = \frac{1}{2}\sigma^2 (\ln y_j)^2 (y_j)^2 \frac{\triangle t}{(\triangle y)^2} + \left[(\frac{1}{T} - r \ln y_j)\, y_j + \frac{1}{2}\sigma^2 y_j\, (\ln y_j)^2 \right] \frac{\triangle t}{\triangle y},$$

with boundary conditions as

$$\check{C}^a_{n_t,j} = 0 \text{ for } 0 \leq j \leq n_y$$
$$\check{C}^a_{i-1,0} = 0$$
$$\check{C}^a_{i-1,n_y} = \frac{1}{Tr}(1 - e^{-r(T-(i-1)\triangle t)})$$

for $1 \leq i \leq n_t$.

For the European option, we repeat the same procedure to obtain

$$\check{C}^e_{i-1,\,j} = P_{j,j-1}\check{C}^e_{i,\,j-1} + P_{j,j}\check{C}^e_{i,\,j} + P_{j,\,j+1}\check{C}^e_{i,\,j+1} \tag{5.7}$$

for j=1,..,n_y-1 where

$$P_{j,j-1} = \frac{1}{2}\sigma^2 (\ln y_j)^2 (z_j)^2 \frac{\triangle t}{(\triangle z)^2},$$

$$P_{j,j} = 1 - \sigma^2 (\ln z_j)^2 (z_j)^2 \frac{\triangle t}{(\triangle z)^2} - \left[-r \ln z_j z_j + \frac{1}{2}\sigma^2 z_j (\ln z_j)^2\right] \frac{\triangle t}{\triangle z},$$

$$P_{j,j+1} = \frac{1}{2}\sigma^2 (\ln z_j)^2 (z_j)^2 \frac{\triangle t}{(\triangle z)^2} + \left[-r \ln z_j z_j + \frac{1}{2}\sigma^2 z_j (\ln z_j)^2\right] \frac{\triangle t}{\triangle z},$$

with boundary conditions as

$$\check{C}^e_{n_t,j} = (1 + \ln(j \triangle z))^+ \text{ for } 0 \leq j \leq n_z$$
$$\check{C}^e_{i-1,0} = 0$$
$$\check{C}^e_{i-1,n_z} = 1$$

for $1 \leq i \leq n_t$.

Using Equations (5.3) and (5.4) is standard. However, instead of using a centered estimate of the first partial derivative of $\check{C}^a$ with respect to y, Alziary *et al.* (1997) use a right estimate. Indeed, with the usual estimate it is not possible to satisfy a stability condition on the whole domain $y \in [0, 1]$.

First, let us show that the explicit finite-difference approximation of Equation (5.6) is obviously consistent with the P.D.E. (5.1). It is not difficult to do the same for the European option to show the consistency of Equation (5.7) with the P.D.E. (4.12).

By using Taylor series expansion we get

$$\check{C}^a_{i-1,\,j} = \check{C}^a_{i,\,j} - \Delta t \left(\frac{\partial \check{C}^a}{\partial t}\right)_{i,\,j} + \frac{1}{2}(\Delta t)^2 \left(\frac{\partial^2 \check{C}^a}{\partial t^2}\right)_{i,\,j} - \frac{1}{6}(\Delta t)^3 \left(\frac{\partial^3 \check{C}^a}{\partial t^3}\right)_{i,\,j} + \ldots$$

$$\check{C}^a_{i,\,j-1} = \check{C}^a_{i,\,j} - \Delta y \left(\frac{\partial \check{C}^a}{\partial y}\right)_{i,\,j} + \frac{1}{2}(\Delta y)^2 \left(\frac{\partial^2 \check{C}^a}{\partial y^2}\right)_{i,\,j} - \frac{1}{6}(\Delta y)^3 \left(\frac{\partial^3 \check{C}^a}{\partial y^3}\right)_{i,\,j} + \ldots$$

$$\check{C}^a_{i,\,j+1} = \check{C}^a_{i,\,j} + \Delta y \left(\frac{\partial \check{C}^a}{\partial y}\right)_{i,\,j} + \frac{1}{2}(\Delta y)^2 \left(\frac{\partial^2 \check{C}^a}{\partial y^2}\right)_{i,\,j} + \frac{1}{6}(\Delta y)^3 \left(\frac{\partial^3 \check{C}^a}{\partial y^3}\right)_{i,\,j} + \ldots$$

Then, substitute these into Equation (5.6)

$$\check{C}^a_{i-1,\,j} = P_{j,j-1}\check{C}^a_{i,\,j-1} + P_{j,j}\check{C}^a_{i,\,j} + P_{j,\,j+1}\check{C}^a_{i,\,j+1}$$

$$\check{C}^a_{i,\,j} - \Delta t \left(\frac{\partial \check{C}^a}{\partial t}\right)_{i,\,j} + \frac{1}{2}(\Delta t)^2 \left(\frac{\partial^2 \check{C}^a}{\partial t^2}\right)_{i,\,j} - \frac{1}{6}(\Delta t)^3 \left(\frac{\partial^3 \check{C}^a}{\partial t^3}\right)_{i,\,j}$$

$$= P_{j,j-1}\left[\check{C}^a_{i,\,j} - \Delta y \left(\frac{\partial \check{C}^a}{\partial y}\right)_{i,\,j} + \frac{1}{2}(\Delta y)^2 \left(\frac{\partial^2 \check{C}^a}{\partial y^2}\right)_{i,\,j} - \frac{1}{6}(\Delta y)^3 \left(\frac{\partial^3 \check{C}^a}{\partial y^3}\right)_{i,\,j}\right] +$$

$$+P_{j,j}\check{C}^a_{i,\,j}+P_{j,\,j+1}\left[\check{C}^a_{i,\,j}+\triangle y\left(\frac{\partial\check{C}^a}{\partial y}\right)_{i,\,j}+\frac{1}{2}(\triangle y)^2\left(\frac{\partial^2\check{C}^a}{\partial y^2}\right)_{i,\,j}\right.$$

$$\left.+\frac{1}{6}(\triangle y)^3\left(\frac{\partial^3\check{C}^a}{\partial y^3}\right)_{i,\,j}\right]+...,$$

$$\check{C}^a_{i,\,j}\left[-1+P_{j,j-1}+P_{j,j}+P_{j,\,j+1}\right]+\triangle t\left(\frac{\partial\check{C}^a}{\partial t}\right)_{i,\,j}+$$

$$+\left(\frac{\partial\check{C}^a}{\partial y}\right)\left[-P_{j,j-1}\triangle y+P_{j,j+1}\triangle y\right]-\frac{1}{2}(\triangle t)^2\left(\frac{\partial^2\check{C}^a}{\partial t^2}\right)_{i,\,j}+$$

$$+\left(\frac{\partial^2\check{C}^a}{\partial y^2}\right)_{i,\,j}\left[P_{j,j-1}\frac{1}{2}(\triangle y)^2+\frac{1}{2}(\triangle y)^2P_{j,\,j+1}\right]+$$

$$+\left(\frac{\partial^3\check{C}^a}{\partial y^3}\right)_{i,\,j}\left[-\frac{1}{6}(\triangle y)^3P_{j,j-1}+\frac{1}{6}(\triangle y)^3P_{j,j+1}\right]+\frac{1}{6}(\triangle t)^3\left(\frac{\partial^3\check{C}^a}{\partial t^3}\right)_{i,\,j}+...=0,$$

$$\check{C}^a_{i,\,j}[0]+\triangle t\left(\frac{\partial\check{C}^a}{\partial t}\right)_{i,\,j}+\left(\frac{\partial\check{C}^a}{\partial y}\right)_{i,j}\left[(\frac{1}{T}-r\ln y_j)\,y_j+\frac{1}{2}\sigma^2y_j\,(\ln y_j)^2\right]\triangle\,t$$

$$-\frac{1}{2}(\triangle t)^2\left(\frac{\partial^2\check{C}^a}{\partial t^2}\right)_{i,\,j}+\left(\frac{\partial^2\check{C}^a}{\partial y^2}\right)_{i,\,j}\left[\frac{1}{2}\sigma^2(\ln y_j)^2(y_j)^2\triangle\,t+\frac{1}{2}\left[(\frac{1}{T}-r\ln y_j)\,y_j+\right.\right.$$

$$\left.\left.+\frac{1}{2}\sigma^2y_j\,(\ln y_j)^2\right]\triangle\,t\triangle\,y\right]+\left(\frac{\partial^3\check{C}^a}{\partial y^3}\right)_{i,\,j}\frac{1}{6}\left[(\frac{1}{T}-r\ln y_j)\,y_j+\frac{1}{2}\sigma^2y_j\,(\ln y_j)^2\right]+$$

$$+\triangle t\,(\triangle y)^2+\frac{1}{6}(\triangle t)^3\left(\frac{\partial^3\check{C}^a}{\partial t^3}\right)_{i,j}+\ldots=0,$$

If we eliminate $\triangle t$, we obtain

$$\left(\frac{\partial\check{C}^a}{\partial t}\right)_{i,j}+\frac{1}{2}\sigma^2(\ln y_j)^2(y_j)^2\left(\frac{\partial^2\check{C}^a}{\partial y^2}\right)_{i,j}+$$

$$+\left[\left(\frac{1}{T}-r\ln y_j\right)y_j+\frac{1}{2}\sigma^2 y_j\,(\ln y_j)^2\right]\left(\frac{\partial\check{C}^a}{\partial y}\right)_{i,j}-\frac{1}{2}\triangle t\left(\frac{\partial^2\check{C}^a}{\partial t^2}\right)_{i,j}+$$

$$+\frac{1}{2}\triangle y\left[\left(\frac{1}{T}-r\ln y_j\right)y_j+\frac{1}{2}\sigma^2 y_j\,(\ln y_j)^2\right]\left(\frac{\partial^2\check{C}^a}{\partial y^2}\right)_{i,j}+O(\triangle y)+O(\triangle t)=0$$

As $\triangle y\to 0$ and $\triangle t\to 0$ the equation simplifies to

$$\left(\frac{\partial\check{C}^a}{\partial t}\right)_{i,j}+\frac{1}{2}\sigma^2(\ln y_j)^2(y_j)^2\left(\frac{\partial^2\check{C}^a}{\partial y^2}\right)_{i,j}+$$

$$+\left[\left(\frac{1}{T}-r\ln y_j\right)y_j+\frac{1}{2}\sigma^2 y_j\,(\ln y_j)^2\right]\left(\frac{\partial\check{C}^a}{\partial y}\right)_{i,j}=0$$

which is the P.D.E. we want to solve. This means explicit finite-difference approximation is consistent. Furthermore, Alziary *et al.* (1997) states that if σ and r are such that $\sigma^2\leq r$, the solution of the difference equation (5.6) approaches to the solution of the P.D.E. (5.1) as $\triangle y\to 0$ and $\triangle t\to 0$ if

$$1-\frac{\sigma^2}{e^2}\frac{\triangle t}{(\triangle y)^2}-\frac{1}{T}\frac{\triangle t}{\triangle y}>0. \tag{5.8}$$

Equation (5.8) is necessary to insure the stability. [The proof is based on a theorem in Ames (1977), p. 75.] Then, for this type of parabolic equation, consistency

and stability imply convergence as it is shown in Lamberton and Lapeyre (1991).

Parallel Implementation of Explicit Method

We now give some information on what parallel computing is and what benefits we expect to gain by implementing it. Parallel computing can be defined as simultaneous usage of computers connected by a network in the solution of a problem. Parallel computing can speed up the process of computing by distributing the data to parallel processors and executing the same or different instructions on these computers. This is the most common type of parallel computing and it is known as a Multiple Instruction Multiple Data system. Second advantage of parallel computing is the decrease in the size of the problem by distribution of data to parallel processors so that it is possible to solve larger problems. This also has an impact on the reduction of wall clock-times in the solution of larger problems.

In parallel programming, we use Message Passing Interface subroutines to provide the communication between parallel processors. Parallel processors exchange data by sending and receiving messages. But, this exchange should be cooperative, such as a send message should have a matching receive massage. To provide the parallelism between processors is the responsibility of the programmer.

The code for explicit method should be implemented for very large number of time and y (or z) intervals to get an accurate value for Asian and European option prices. We use parallel programming to speed up the pricing process. In this section we introduce the approach we use in parallelizing the explicit pricing method. Because of the similarity of the structure of Asian and European options, the approach will be the same for both of the options.

We introduce the approach on an illustrative example. In the Figure 5.1, there are nine y (or z) intervals along each time column, and the number of time columns does not matter. Because of the boundary conditions, we know all the relative prices at time column i and relative prices at $j = 0$ and $j = 9$ at time column $i-1$. Moreover,

from Equations (5.6) and (5.7) it is obvious that the unknown price at $(i-1,\ j)$ will be dependent on prices at $(i,\ j-1)$, $(i,\ j)$ and $(i,\ j+1)$. We divide the y (or z) internal mesh points to three processors (P_0, P_1 and P_2) all over the time columns. This will automatically necessitate send-receive relationships between processors to price all the mesh points. This algorithm can be easily extended for number of processors greater than three.

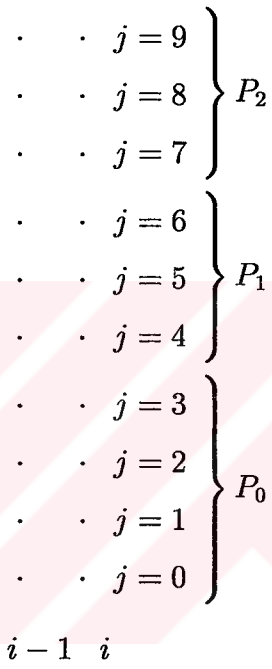

Figure 5.1. Partition of mesh points to processors along each time column

For an arbitrary time column i, we demonstrate the operation done by each processor. In the presentation, $(i, 3)$ stands for the relative price at time column i for $j = 3$.

The sequence of operations done by P_0 is

1- Send $(i, 3)$ to P_1

2- Receive $(i, 4)$ from P_1

3- Calculate unknown relative prices at the territory of P_0 at time column $i-1$.

The sequence of operations done by P_1 is

1- Send $(i,4)$ to P_0

2- Receive $(i,3)$ from P_0

3- Send $(i,6)$ to P_2

4- Receive $(i,7)$ from P_2

5- Calculate unknown relative prices at the territory of P_1 at time column $i-1$.

The sequence of operations done by P_2 is

1- Send $(i,7)$ to P_1

2- Receive $(i,6)$ from P_1

3- Calculate unknown relative prices at the territory of P_2 at time column $i-1$.

5.4. Crank-Nicolson Implicit Finite-Difference Method

Although the explicit method is computationally simple it has one series drawback. The time step $(\triangle t)$ and $(\triangle y)$ should be necessarily very small to attain reasonable accuracy. Crank and Nicolson (1947) proposed, and used, a method that reduces the total volume of calculation and is valid (i.e., convergent) for all values of $(\triangle t)$ and $(\triangle y)$. They replaced $\frac{\partial \check{C}^a}{\partial y}$ and $\frac{\partial^2 \check{C}^a}{\partial y^2}$ by the mean of its finite-difference representations on the $(i)'$th and $(i-1)'$th time columns.

$$\frac{\partial \check{C}^a}{\partial y} = \frac{1}{4 \triangle y}(\check{C}^a_{i,\,j+1} - \check{C}^a_{i,\,j-1} + \check{C}^a_{i-1,\,j+1} - \check{C}^a_{i-1,\,j-1}), \tag{5.9}$$

$$\frac{\partial^2 \check{C}^a}{\partial y^2} = \frac{1}{2 \triangle y^2}(\check{C}^a_{i,\,j+1} + \check{C}^a_{i,\,j-1} - 2\check{C}^a_{i,\,j} + \check{C}^a_{i-1,\,j+1} + \check{C}^a_{i-1,\,j-1} - 2\check{C}^a_{i-1,\,j}), \quad (5.10)$$

and the time derivative is approximated as

$$\frac{\partial \check{C}^a}{\partial t} = \frac{\check{C}^a_{i,\,j} - \check{C}^a_{i-1,\,j}}{\triangle t} \quad (5.11)$$

Substituting Equations (5.9)—(5.11) into Equation (5.1) gives

$$\frac{\check{C}^a_{i,\,j} - \check{C}^a_{i-1,\,j}}{\triangle t} + \frac{1}{2}\sigma^2 (\ln y_j)^2 (y_j)^2 \frac{1}{2 \triangle y^2}[\check{C}^a_{i,\,j+1} + \check{C}^a_{i,\,j-1} - 2\check{C}^a_{i,\,j} +$$

$$+\check{C}^a_{i-1,\,j+1} + \check{C}^a_{i-1,\,j-1} - 2\check{C}^a_{i-1,\,j}] + \left[(\frac{1}{T} - r \ln y_j)\, y_j + \frac{1}{2}\sigma^2 y_j\, (\ln y_j)^2\right] \times$$

$$\left[\frac{1}{4 \triangle y}(\check{C}^a_{i,\,j+1} - \check{C}^a_{i,\,j-1} + \check{C}^a_{i-1,\,j+1} - \check{C}^a_{i-1,\,j-1})\right] = 0$$

$$\left(A^a_j - B^a_j\right) \check{C}^a_{i-1,\,j-1} + \left(-\frac{1}{\triangle t} - 2A^a_j\right) \check{C}^a_{i-1,\,j} + \left(A^a_j + B^a_j\right) \check{C}^a_{i-1,\,j+1} = D^a_{i,j} \quad (5.12)$$

for j= 1,..,N-1 and where

$$A^a_j = \frac{1}{4\,(\triangle y)^2}\sigma^2 (\ln y_j)^2 (y_j)^2$$

$$B^a_j = \frac{1}{4 \triangle y}\left[(\frac{1}{T} - r \ln y_j)\, y_j + \frac{1}{2}\sigma^2 y_j\, (\ln y_j)^2\right]$$

$$D^a_{i,j} = \left(-A^a_j + B^a_j\right) \check{C}^a_{i,\,j-1} + \left(-\frac{1}{\triangle t} + 2A_j\right) \check{C}^a_{i,\,j} + \left(-A^a_j - B^a_j\right) \check{C}^a_{i,\,j+1}.$$

For European option, only one coefficient is a little bit different as given below

$$\left(A_j^e - B_j^e\right) \check{C}^e_{i-1,\,j-1} + \left(-\frac{1}{\triangle t} - 2A_j^e\right) \check{C}^e_{i-1,\,j} + \left(A_j^e + B_j^e\right) \check{C}^e_{i-1,\,j+1} = D^e_{i,j} \qquad (5.13)$$

for j= 1,..,N-1 and where

$$A_j^e = \frac{1}{4\left(\triangle z\right)^2}\sigma^2 \left(\ln z_j\right)^2 \left(z_j\right)^2$$

$$B_j^e = \frac{1}{4 \triangle z}\left[\left(-r \ln z_j\right) z_j + \frac{1}{2}\sigma^2 z_j \left(\ln z_j\right)^2\right]$$

$$D^e_{i,j} = \left(-A_j^e + B_j^e\right) \check{C}^e_{i,\,j-1} + \left(-\frac{1}{\triangle t} + 2A_j\right) \check{C}^e_{i,\,j} + \left(-A_j^e - B_j^e\right) \check{C}^e_{i,\,j+1}.$$

Equations (5.12) and (5.13) are systems of linear equations. If there are $N+1$ mesh points along each time column then there are $N-1$ variables to be solved at each time column. The boundary conditions at $j = 0$ and $j = N$ are used to calculate the relative prices there. In the solution of system of linear equations there are many algorithms. We can categorize them under three headings: Gauss Elimination, LU Decomposition and Iterative methods.

5.4.1. Gauss Elimination Method

We implement the Gauss Elimination algorithm described in Smith (1978). System of linear equations in Equations (5.12) and (5.13) can be shown in matrix form as in Equation (5.14). This structure is known as tridiagonal matrix form in literature. Since we use common notations for both Asian and European options throughout the previous sections (only coefficients' values are different), the notation presented is directly applicable to both options. However, boundary conditions stated previously are

different for Asian and European options.

$$\begin{bmatrix} b_1 & -c_1 & & & & & \\ -a_2 & b_2 & -c_2 & & & & \\ & -a_3 & b_3 & -c_3 & & & \\ & & \cdot & \cdot & \cdot & & \\ & & & \cdot & \cdot & \cdot & \\ & & & & -a_{N-2} & b_{N-2} & -c_{N-2} \\ & & & & & -a_{N-1} & b_{N-1} \end{bmatrix} \begin{bmatrix} \check{C}_{i-1,1} \\ \check{C}_{i-1,2} \\ \check{C}_{i-1,3} \\ \cdot \\ \cdot \\ \check{C}_{i-1,N-2} \\ \check{C}_{i-1,N-1} \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \cdot \\ \cdot \\ d_{N-2} \\ d_{N-1} \end{bmatrix}, \tag{5.14}$$

where

$$\begin{aligned} a_j &= (B_j - A_j) && \text{for } j = 2, 3, ...N-1 \\ b_j &= \left(-\frac{1}{\Delta t} - 2A_j\right) && \text{for } j = 1, 2, ...N-1 \\ c_j &= (-A_j - B_j) && \text{for } j = 1, 2, ...N-2 \\ d_j &= D_{i,j} && \text{for } j = 2, 3, ...N-2 \end{aligned}$$

In addition to that we have

$$d_1 = (-A_1 + B_1)\,\check{C}_{i,0} + \left(-\frac{1}{\Delta t} + 2A_1\right)\check{C}_{i,1} + (-A_1 - B_1)\,\check{C}_{i,2} - (A_1 - B_1)\,\check{C}_{i-1,0}$$

$$d_{N-1} = (-A_{N-1} + B_{N-1})\,\check{C}_{i,N-2} + \left(-\frac{1}{\Delta t} + 2A_{N-1}\right)\check{C}_{i,N-1} +$$

$$+ (-A_{N-1} - B_{N-1})\,\check{C}_{i,N} + (-A_{N-1} - B_{N-1})\,\check{C}_{i-1,N}.$$

The first equation can be used to eliminate $\check{C}_{i-1,1}$ from the second equation, the new second equation used to eliminate $\check{C}_{i-1,2}$ from the third equation and so on, $\check{C}_{i-1,N-2}$ from the last equation, giving one equation with only one unknown,

$\check{C}_{i-1,N-1}$. The unknowns $\check{C}_{i-1,N-2}, \check{C}_{i-1,N-3}, ...\check{C}_{i-1,2}$, $\check{C}_{i-1,1}$ can be found in turn by back-substitution. An algorithm is given below to implement this logic.

$$\alpha_1 = b_1 \text{ for } j = 1, \text{then } \alpha_j = b_j - a_j \frac{c_{j-1}}{\alpha_{j-1}} \text{ for } j = 2, 3, ...N-1$$

$$S_1 = d_1 \text{ for } j = 1, \text{then } S_j = d_j + a_j \frac{S_{j-1}}{\alpha_{j-1}} \text{for } j = 2, 3, ...N-1$$

and finally

$$\check{C}_{i,N-1} = \frac{S_{N-1}}{\alpha_{N-1}} \text{ for } j = N-1, \text{then } \check{C}_{i,j} = \frac{1}{\alpha_j}(S_j + c_j \check{C}_{i,j+1}) \text{ for } j = N-2, N-1, ...1.$$

Parallel Implementation of Gauss Elimination Method

There are ingenious parallel algorithms designed to solve tridiagonal system of linear equations by using Gauss Elimination. Cyclic Elimination, Cyclic Reduction and Block Partition algorithms are some of these. We only discuss Block Partition algorithms.

We present the Block Partition algorithm described in Wang (1981). Wang (1981) starts by dividing the general system ($A\check{C} = d$) into p groups (i.e., number of processors) each consisting of $n = (N-1)/p$ consecutive rows. For convenience, we assume $(N-1)$ is multiple of p. Then the elimination can proceed simultaneously on all subsystems by elementary row transformations until finally A is diagonalized. We use the notation presented in Lin (2001) to make the algorithm more clear and codable. We illustrate the elimination pattern via an example accompanying formal presentation of the algorithm. We divide the whole process into stages to make it more clear. Consider a tridiagonal system of 12 equations and 3 processors (i.e. $N-1 = 12$ and $p = 3$) for the figures. However, in the formulas following the description of each stage we think the general system (i.e., $N-1 = pn$) to formally present the algorithm. Moreover, the

indices for the processors goes in the order of $P_0, P_1, ..., P_{p-1}$.

Step 1: Partition the matrix into, say 3 block tridiagonal form as shown in Figure 5.2. Although, obviously the variables are not the same for all processors, we show as if they are the same to run a unified code on all processors. This can be done because of the reality that each processor has the knowledge of only its data. The others are out of scope. First four rows belong to processor zero (P_0), second four rows belong to processor one (P_1) and last four rows belong to processor two (P_2). The variables that are not shown in Figure 5.2 are zero such as $a_1$in P_0. Moreover, d vector is not shown in figures because it does not change form. Only values of the variables change throughout the steps.

$$\begin{bmatrix}
b_1 & c_1 & & & & & & & & & & \\
a_2 & b_2 & c_2 & & & & & & & & & \\
 & a_3 & b_3 & c_3 & & & & & & & & \\
 & & a_4 & b_4 & c_4 & & & & & & & \\
- & - & - & - & - & - & - & - & - & - & - & - \\
 & & & a_1 & b_1 & c_1 & & & & & & \\
 & & & & a_2 & b_2 & c_2 & & & & & \\
 & & & & & a_3 & b_3 & c_3 & & & & \\
 & & & & & & a_4 & b_4 & c_4 & & & \\
- & - & - & - & - & - & - & - & - & - & - & - \\
 & & & & & & & a_1 & b_1 & c_1 & & \\
 & & & & & & & & a_2 & b_2 & c_2 & \\
 & & & & & & & & & a_3 & b_3 & c_3 \\
 & & & & & & & & & & a_4 & b_4
\end{bmatrix}$$

Figure 5.2. Partition of the general system into blocks

Step 2: Simultaneously apply the Gaussian Elimination scheme to each submatrix until you get Figure 5.3 on each processor. That is apply the formulas given below for all processors. It should be clear that even when there is a change in the numerical

values we do not change the variables . This strategy is used to avoid complexity. Moreover, in the formulas $a_j^{(m)}$ stands for the current value of a_j and $a_j^{(m-1)}$ stands for the previous value of a_j.

$$\left.\begin{aligned} a_j^{(m)} &= -a_{j-1}^{(m-1)} \frac{a_j^{(m-1)}}{b_{j-1}^{(m-1)}} && j = 2,3,...,n \\ b_j^{(m)} &= b_j^{(m-1)} - c_{j-1}^{(m-1)} \frac{a_j^{(m-1)}}{b_{j-1}^{(m-1)}} && j = 2,3,...,n \\ d_j^{(m)} &= d_j^{(m-1)} - d_{j-1}^{(m-1)} \frac{a_j^{(m-1)}}{b_{j-1}^{(m-1)}} && j = 2,3,...,n \end{aligned}\right\} \quad \text{for all processors}$$

$$\left[\begin{array}{cccccccccccc}
b_1 & c_1 & & & & & & & & & & \\
 & b_2 & c_2 & & & & & & & & & \\
 & & b_3 & c_3 & & & & & & & & \\
 & & & b_4 & c_4 & & & & & & & \\
- & - & - & - & - & - & - & - & - & - & - & - \\
 & & & a_1 & b_1 & c_1 & & & & & & \\
 & & & a_2 & & b_2 & c_2 & & & & & \\
 & & & a_3 & & & b_3 & c_3 & & & & \\
 & & & a_4 & & & & b_4 & c_4 & & & \\
- & - & - & - & - & - & - & - & - & - & - & - \\
 & & & & & & & a_1 & b_1 & c_1 & & \\
 & & & & & & & a_2 & & b_2 & c_2 & \\
 & & & & & & & a_3 & & & b_3 & c_3 \\
 & & & & & & & a_4 & & & & b_4
\end{array}\right]$$

Figure 5.3. The form of matrix A after the second step

Step 3: Simultaneously apply the Gaussian Elimination scheme to each submatrix until you get Figure 5.4 on each processor.

$$\left.\begin{aligned} a_j^{(m)} &= a_j^{(m-1)} - a_{j+1}^{(m-1)} \frac{c_j^{(m-1)}}{b_{j+1}^{(m-1)}} \quad j = n-2, n-1, ..., 1 \\ c_j^{(m)} &= -c_{j+1}^{(m-1)} \frac{c_j^{(m-1)}}{b_{j+1}^{(m-1)}} \quad j = n-2, n-1, ..., 1 \\ d_j^{(m)} &= d_j^{(m-1)} - d_{j+1}^{(m-1)} \frac{c_j^{(m-1)}}{b_{j+1}^{(m-1)}} \quad j = n-2, n-1, ..., 1 \end{aligned}\right\} \text{for all processors except } P_{p-1}$$

$$\left.\begin{aligned} a_j^{(m)} &= a_j^{(m-1)} - a_{j+1}^{(m-1)} \frac{c_j^{(m-1)}}{b_{j+1}^{(m-1)}} \quad j = n-1, n-2, ..., 1 \\ c_j^{(m)} &= -c_{j+1}^{(m-1)} \frac{c_j^{(m-1)}}{b_{j+1}^{(m-1)}} \quad j = n-1, n-2, ..., 1 \\ d_j^{(m)} &= d_j^{(m-1)} - d_{j+1}^{(m-1)} \frac{c_j^{(m-1)}}{b_{j+1}^{(m-1)}} \quad j = n-1, n-2, ..., 1 \end{aligned}\right\} \text{for } P_{p-1}$$

Step 4: Simultaneously apply the Gaussian Elimination scheme to each submatrix until you get Figure 5.5 on each processor. However, there have to be send-receive relationships between processors to apply the Gaussian Elimination scheme.

The operation done by P_0 is

1. Send $b_n^{(m)}, c_n^{(m)}$ and $d_n^{(m)}$ to P_1

The operation done by P_{p-1} is

1. Receive $b_0^{(m-1)}, c_0^{(m-1)}$ and $d_0^{(m-1)}$ from P_{p-2}

$$
\left[\begin{array}{cccccccccccc}
b_1 & & & c_1 & & & & & & & & \\
 & b_2 & & c_2 & & & & & & & & \\
 & & b_3 & c_3 & & & & & & & & \\
 & & & b_4 & c_4 & & & & & & & \\
- & - & - & - & - & - & - & - & - & - & - & - \\
 & & & a_1 & b_1 & & & c_1 & & & & \\
 & & & a_2 & & b_2 & & c_2 & & & & \\
 & & & a_3 & & & b_3 & c_3 & & & & \\
 & & & a_4 & & & & b_4 & c_4 & & & \\
- & - & - & - & - & - & - & - & - & - & - & - \\
 & & & & & & & a_1 & b_1 & & & \\
 & & & & & & & a_2 & & b_2 & & \\
 & & & & & & & a_3 & & & b_3 & \\
 & & & & & & & a_4 & & & & b_4
\end{array}\right]
$$

Figure 5.4. The form of matrix A after the third step

The sequence of operations done by other processors (P_i) is

1. Receive $b_0^{(m-1)}, c_0^{(m-1)}$ and $d_0^{(m-1)}$ from P_{i-1}

2. Send $b_n^{(m)}, c_n^{(m)}$ and $d_n^{(m)}$ to P_{i+1}

$$
\left.\begin{aligned}
b_0^{(m)} &= b_0^{(m-1)} - a_1^{(m-1)} \frac{c_0^{(m-1)}}{b_1^{(m-1)}} \\
c_0^{(m)} &= -c_1^{(m-1)} \frac{c_0^{(m-1)}}{b_1^{(m-1)}} \\
d_0^{(m)} &= d_0^{(m-1)} - d_1^{(m-1)} \frac{c_0^{(m-1)}}{b_1^{(m-1)}}
\end{aligned}\right\} \quad \text{for all processors except } P_0
$$

The operation done by P_0 is

1. Receive $b_n^{(m-1)}$ and $d_n^{(m-1)}$ from P_1

The operation done by P_{p-1} is

1. Send $b_0^{(m)}$ and $d_0^{(m)}$ to P_{p-2}

The sequence of operations done by other processors (P_i)is

1. Send $b_0^{(m)}$ and $d_0^{(m)}$ to P_{i-1}

2. Receive $b_n^{(m-1)}$ and $d_n^{(m-1)}$ from P_{i+1}

$$
\begin{bmatrix}
b_1 & & & c_1 & & & & & & & & \\
 & b_2 & & c_2 & & & & & & & & \\
 & & b_3 & c_3 & & & & & & & & \\
 & & & b_4 & & & & c_4 & & & & \\
- & - & - & - & - & - & - & - & - & - & - & - \\
 & & & a_1 & b_1 & & & c_1 & & & & \\
 & & & a_2 & & b_2 & & c_2 & & & & \\
 & & & a_3 & & & b_3 & c_3 & & & & \\
 & & & a_4 & & & & b_4 & & & & \\
- & - & - & - & - & - & - & - & - & - & - & - \\
 & & & & & & & a_1 & b_1 & & & \\
 & & & & & & & a_2 & & b_2 & & \\
 & & & & & & & a_3 & & & b_3 & \\
 & & & & & & & a_4 & & & & b_4
\end{bmatrix}
$$

Figure 5.5. The form of matrix A after the fourth step

Step 5: Apply the Gaussian Elimination scheme to each submatrix sequentially until you get Figure 5.6 on each processor. This stage is sequential since we have to wait until the values to be used in the next processor are determined. In addition, there has to be send-receive relationships between processors to apply the Gaussian Elimination scheme.

$$\text{do nothing } \} \qquad \text{for } P_0$$

$$\left.\begin{array}{c} a_n^{(m)} = 0 \\ b_n^{(m)} = b_n^{(m-1)} - c_0^{(m-1)} \dfrac{a_n^{(m-1)}}{b_0^{(m-1)}} \\ d_n^{(m)} = d_n^{(m-1)} - d_0^{(m-1)} \dfrac{a_n^{(m-1)}}{b_0^{(m-1)}} \\ \text{Send } b_n^{(m)}, d_n^{(m)} \text{ to } P_2 \end{array}\right\} \quad \text{for } P_1$$

$$\left.\begin{array}{c} \text{Receive } b_0^{(m-1)}, d_0^{(m-1)} \text{ from } P_{p-2} \\ a_n^{(m)} = 0 \\ b_n^{(m)} = b_n^{(m-1)} - c_0^{(m-1)} \dfrac{a_n^{(m-1)}}{b_0^{(m-1)}} \\ d_n^{(m)} = d_n^{(m-1)} - d_0^{(m-1)} \dfrac{a_n^{(m-1)}}{b_0^{(m-1)}} \end{array}\right\} \quad \text{for } P_{p-1}$$

$$\left.\begin{array}{c} \text{Receive } b_0^{(m-1)}, d_0^{(m-1)} \text{ from } P_{i-1} \\ a_n^{(m)} = 0 \\ b_n^{(m)} = b_n^{(m-1)} - c_0^{(m-1)} \dfrac{a_n^{(m-1)}}{b_0^{(m-1)}} \\ d_n^{(m)} = d_n^{(m-1)} - d_0^{(m-1)} \dfrac{a_n^{(m-1)}}{b_0^{(m-1)}} \\ \text{Send } b_n^{(m)}, d_n^{(m)} \text{ to } P_{i+1} \end{array}\right\} \quad \text{for other processors } (P_i)$$

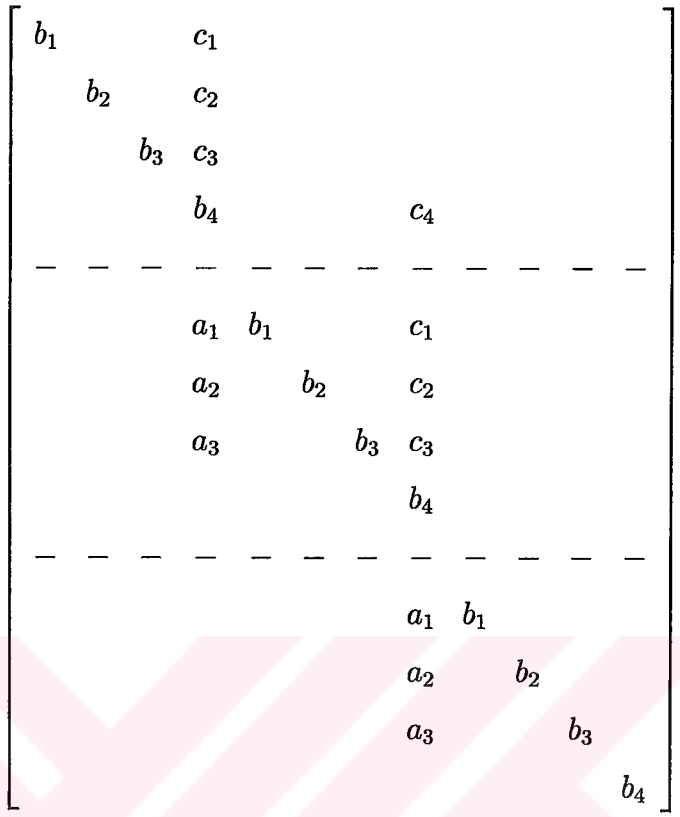

Figure 5.6. The form of matrix A after the fifth step

Step 6: Apply the Gaussian Elimination scheme to each submatrix sequentially until you get Figure 5.7 on each processor. This stage is also sequential since we have to wait until the values to be used in the previous processor are determined. In addition, there have to be send-receive relationships between processors to apply the Gaussian Elimination scheme.

$$\text{do nothing } \} \qquad \text{for } P_{p-1}$$

$$\left.\begin{array}{c} c_0^{(m)} = 0 \\ d_0^{(m)} = d_0^{(m-1)} - d_n^{(m-1)} \dfrac{c_0^{(m-1)}}{b_n^{(m-1)}} \\ \text{Send } d_0^{(m)} \text{ to } P_{p-3} \end{array}\right\} \qquad \text{for } P_{p-2}$$

Receive $d_n^{(m-1)}$ from P_1 $\}$ for P_0

$$\left.\begin{array}{c} \text{Receive } d_n^{(m-1)} \text{ from } P_{i+1} \\ c_0^{(m)} = 0 \\ d_0^{(m)} = d_0^{(m-1)} - d_n^{(m-1)} \dfrac{c_0^{(m-1)}}{b_n^{(m-1)}} \\ \text{Send } d_0^{(m)} \text{ to } P_{i-1} \end{array}\right\} \quad \text{for other processors } (P_i)$$

$$\begin{bmatrix}
b_1 & & c_1 & & & & & & & & & \\
& b_2 & c_2 & & & & & & & & & \\
& & b_3 \; c_3 & & & & & & & & & \\
& & b_4 & & & & & & & & & \\
- & - & - & - & - & - & - & - & - & - & - & - \\
& & a_1 & b_1 & & & c_1 & & & & & \\
& & a_2 & & b_2 & & c_2 & & & & & \\
& & a_3 & & & b_3 & c_3 & & & & & \\
& & & & & & b_4 & & & & & \\
- & - & - & - & - & - & - & - & - & - & - & - \\
& & & & & & a_1 & b_1 & & & & \\
& & & & & & a_2 & & b_2 & & & \\
& & & & & & a_3 & & & b_3 & & \\
& & & & & & & & & & b_4 &
\end{bmatrix}$$

Figure 5.7. The form of matrix A after the sixth step

Step 7: Simultaneously apply the Gaussian Elimination scheme to each submatrix until you get Figure 5.8 on each processor. That is, apply the formulas below.

$$\left.\begin{aligned} a_j^{(m)} &= 0 && j = 1, 2, ..., n-1 \\ d_j^{(m)} &= d_j^{(m-1)} - d_0^{(m-1)} \frac{a_j^{(m-1)}}{b_0^{(m-1)}} && j = 1, 2, ..., n-1 \end{aligned}\right\} \quad \text{for all processors except } P_0$$

$$\begin{bmatrix}
b_1 & & c_1 & & & & & & & & & \\
& b_2 & c_2 & & & & & & & & & \\
& & b_3 & c_3 & & & & & & & & \\
& & & b_4 & & & & & & & & \\
- & - & - & - & - & - & - & - & - & - & - & - \\
& & & & b_1 & & c_1 & & & & & \\
& & & & & b_2 & c_2 & & & & & \\
& & & & & & b_3 & c_3 & & & & \\
& & & & & & & b_4 & & & & \\
- & - & - & - & - & - & - & - & - & - & - & - \\
& & & & & & & & b_1 & & & \\
& & & & & & & & & b_2 & & \\
& & & & & & & & & & b_3 & \\
& & & & & & & & & & & b_4
\end{bmatrix}$$

Figure 5.8. The form of matrix A after the seventh step

Step 8: Simultaneously apply the Gaussian Elimination scheme to each submatrix until you get Figure 5.9 on each processor. That is, apply the formulas below.

$$\left.\begin{aligned} c_j^{(m)} &= 0 && j = 1, 2, ..., n-1 \\ d_j^{(m)} &= d_j^{(m-1)} - d_n^{(m-1)} \frac{c_j^{(m-1)}}{b_n^{(m-1)}} && j = 1, 2, ..., n-1 \end{aligned}\right\} \text{for all processors except } P_{p-1}$$

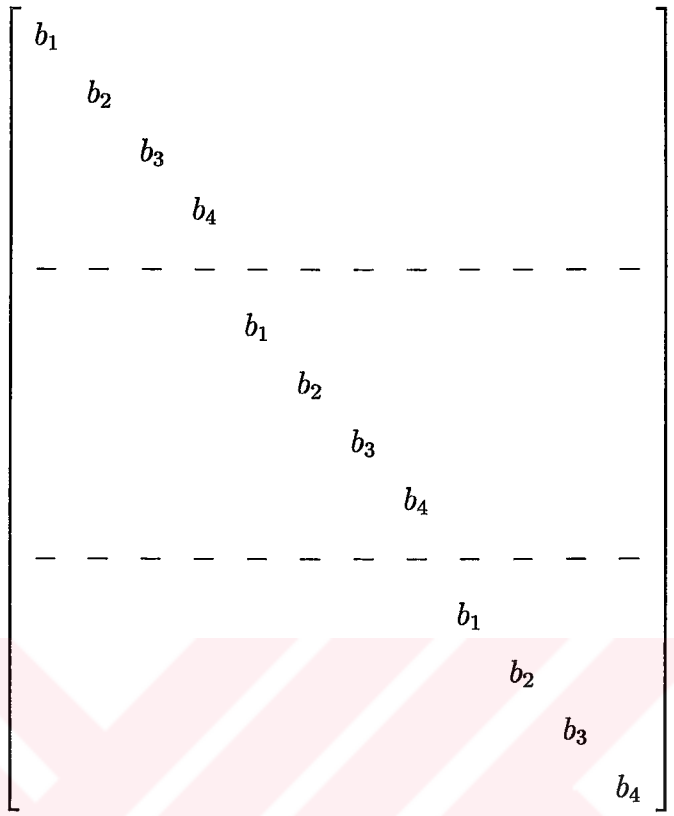

Figure 5.9. The form of matrix A after the eight step

Step 9: The solution can now be computed by

$$\left.\begin{array}{l} \check{C}_j = \dfrac{d_j^{(m-1)}}{b_j^{(m-1)}} \quad j = 1, 2, ..., n \end{array}\right\} \qquad \text{for all processors}$$

This completes the algorithm description.

5.4.2. LU Decomposition Method

LU decomposition method's principle is to divide the system ($A\check{C} = d$) given in Equation (5.14) for our problem into two separate systems. One of the matrices in the smaller systems is upper-triangular and the other one is lower-triangular matrix. The solution of two small systems is easier than one large system. This is the reason why

LU decomposition is effective in solving large system of linear equations. Let's first state how the smaller systems can be formulated from the large system.

$$A\check{C} = d$$
$$LU\check{C} = d$$
$$L^{-1}LU\check{C} = L^{-1}d$$
$$U\check{C} = L^{-1}d = z$$
$$Lz = d$$
$$U\check{C} = z$$

So that we have two smaller systems $Lz = d$ and $U\check{C} = z$ to solve. L is the lower-triangular and U is the upper-triangular matrix. Before starting to solve the two smaller systems we need to find the L and U out of A. There are many algorithms designed to accomplish the decomposition of A into L and U in the literature. We prefer to implement the Thomas (1949) algorithm for the decomposition of A. Thomas algorithm decomposes A and solves the systems in such a manner that

$$A = LU = \begin{bmatrix} b_1 & -c_1 & & & & & \\ -a_2 & b_2 & -c_2 & & & & \\ & -a_3 & b_3 & -c_3 & & & \\ & & \cdot & \cdot & \cdot & & \\ & & & \cdot & \cdot & \cdot & \\ & & & & -a_{N-2} & b_{N-2} & -c_{N-2} \\ & & & & & -a_{N-1} & b_{N-1} \end{bmatrix}.$$

where

$$L = \begin{bmatrix} 1 & & & & & \\ bb_2 & 1 & & & & \\ & bb_3 & 1 & & & \\ & & \cdot & \cdot & & \\ & & & \cdot & \cdot & \\ & & & & bb_{N-1} & 1 \end{bmatrix},$$

$$U = \begin{bmatrix} dd_1 & -c_1 & & & & \\ & dd_2 & -c_2 & & & \\ & & dd_3 & -c_3 & & \\ & & & \cdot & \cdot & \\ & & & & \cdot & \cdot \\ & & & & dd_{N-1} & -c_{N-1} \\ & & & & & dd_N \end{bmatrix}$$

and

$$\begin{aligned} bb_1 &= 0; & dd_1 &= b_1 & &\text{for } i = 1 \\ bb_i &= \frac{-a_i}{dd_{i-1}}; & dd_i &= b_i + bb_i c_{i-1} & &\text{for } i = 2, 3, ..., N-1 \end{aligned}$$

Finally, the solution of the smaller systems $Lz = d$ and $U\check{C} = z$ are

$$z_1 = d_1 \text{ for } i = 1, \text{ then } z_i = d_i - bb_i z_{i-1} \text{ for } i = 2, 3, ..., N-1$$

and

$$\check{C}_{N-1} = \frac{z_{N-1}}{dd_{N-1}} \text{ for } i = N-1, \text{ then } \check{C}_i = (z_i + c_i \check{C}_{i+1}) \,/\, dd_i \text{ for } i = N-2, N-3, ..., 1.$$

Parallel Implementation of LU Decomposition Method

Recursive Doubling and Parallel Factorization algorithms are parallel algorithms designed to solve tridiagonal system of linear equations by using LU decomposition. We discuss only Parallel Factorization algorithm.

Parallel Factorization algorithm was first proposed by Amodio and Brugnano (1992). However, we describe the algorithm presented in Mattor *et al.* (1995) as an more improved version of the algorithm. Parallel Factorization algorithm described in Mattor *et al.* (1995) starts by dividing the original tridiagonal matrix (A) into smaller tridiagonal matrices which can be allocated to different processors. Then it solves each system formed at all processors independently and obviously simultaneously. The next step is to form a reduced matrix coupling all of the systems. Finally it solves the reduced matrix to find the coefficients which are used to construct the real solutions.

Let's start the description of the algorithm. Divide the system ($A\check{C} = d$) into p groups (i.e., number of processors) each consisting of $n = (N-1)/p$ consecutive rows. For convenience, we assume $(N-1)$ is multiple of p. We illustrate the elimination pattern via an example accompanying formal presentation of the algorithm. We divide the whole process into stages to make it more clear. Consider a tridiagonal system of 9 equations and 3 processors (i.e., $N-1=9$ and $p=3$) for the figures. However, in the formulas following the description of each stage we think of the general system (i.e., $N-1=pn$) to formally present the algorithm. Moreover, the indices for the processors goes in the order of $P_0, P_1, ..., P_{p-1}$.

Step 1: Partition the matrix, so that 3 tridiagonal matrices are obtained to be allocated to different processors as shown in Figure 5.10. Call L_i the tridiagonal matrix of processor i shown below. Although, obviously the variables are not the same for all processors, we show as if they are the same to run a unified code on all processors. This can be done because of the fact that each processor has the knowledge of only its data. The others are out of scope.

$$\left[\begin{array}{cccc|cccc|ccc}
b_1 & -c_1 & & & & & & & & & \\
-a_2 & b_2 & -c_2 & & & & & & & & \\
 & -a_3 & b_3 & & -c_3 & & & & & & \\
- & - & - & & - & - & - & - & & & \\
 & & -a_1 & & b_1 & -c_1 & & & & & \\
 & & & & -a_2 & b_2 & -c_2 & & & & \\
 & & & & & -a_3 & b_3 & & -c_3 & & \\
 & & & & - & - & - & - & - & - & - \\
 & & & & & & -a_1 & & b_1 & -c_1 & \\
 & & & & & & & & -a_2 & b_2 & -c_2 \\
 & & & & & & & & & -a_3 & b_3
\end{array}\right]$$

Figure 5.10. Partition of the general system into parallel processors

For $i = 0, 1$ and 2, L_i has the following structure L_i for $i = 0, 1$ and 2.

$$L_i = \left[\begin{array}{ccc}
b_1 & -c_1 & \\
-a_2 & b_2 & -c_2 \\
 & -a_3 & b_3
\end{array}\right]$$

Step 2: Three vectors x_i^R, x_i^{UH} and x_i^{LH} are defined as the solutions to equations stated below. Apply these equations on all processors (i.e., i = 0,1,.., p-1) to solve for x_i^R, x_i^{UH} and x_i^{LH}. The superscripts on the variable x_i stand for "particular" solution, "upper homogenous" solution and "lower homogenous" solution in that order. As we inform the reader before, in the formulas below the general system (i.e., $N - 1 = pn$) is considered.

$$L_i x_i^R = d_i \;\; i = 0, 1, .., p - 1 \tag{5.15}$$

$$L_i x_i^{UH} = (a_1 0\ 0\ .\ .\ .0)^T \quad i = 0, 1, .., p-1 \tag{5.16}$$

$$L_i x_i^{LH} = (0\ 0\ 0\ .\ .\ .c_n)^T \quad i = 0, 1, .., p-1 \tag{5.17}$$

where

$$L_i = \begin{bmatrix} b_1 & -c_1 & & & & \\ -a_2 & b_2 & -c_2 & & & \\ & -a_3 & b_3 & -c_3 & & \\ & & . & . & . & \\ & & & . & . & . \\ & & & & -a_n & b_n \end{bmatrix}$$

and

$$d_i = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ . \\ . \\ d_n \end{bmatrix}$$

Step 3: The general solution (x_i) at all processors (i.e., i = 0,1,..,p-1) is the linear combination of x_i^R, x_i^{UH} and x_i^{LH}. And, it is given as

$$x_i = x_i^R + \xi_i^{UH} x_i^{UH} + \xi_i^{LH} x_i^{LH} \quad i = 0, 1, .., p-1 \tag{5.18}$$

where ξ_i^{UH} and ξ_i^{LH} are the coefficients to be determined by coupling all subsystems. To find ξ_i^{UH} and ξ_i^{LH} simply substitute Equation (5.18) into Equation (5.14). After simple algebra and definitions it is easy to show $\xi_1^{UH} = \xi_{p-1}^{LH} = 0$, and form a tridiagonal

system of linear equations given below to solve the remaining of $(2p-2)$ coefficients.

$$\begin{bmatrix} x_{1,n}^{LH} & -1 & & & & & & \\ -1 & x_{2,1}^{UH} & x_{2,1}^{LH} & & & & & \\ & x_{2,n}^{UH} & x_{2,n}^{LH} & -1 & & & & \\ & & -1 & x_{3,1}^{LH} & x_{3,1}^{LH} & & & \\ & & & x_{3,n}^{UH} & x_{3,n}^{LH} & -1 & & \\ & & & & \cdot & \cdot & \cdot & \\ & & & & & \cdot & \cdot & \cdot \\ & & & & & & \cdot & -1 \\ & & & & & & -1 & x_{p-1,1}^{UH} \end{bmatrix} \begin{bmatrix} \xi_1^{LH} \\ \xi_2^{UH} \\ \xi_2^{LH} \\ \xi_3^{UH} \\ \xi_3^{LH} \\ \cdot \\ \cdot \\ \xi_{p-2}^{LH} \\ \xi_{p-1}^{UH} \end{bmatrix} = - \begin{bmatrix} x_{1,n}^{R} \\ x_{2,1}^{R} \\ x_{2,n}^{R} \\ x_{3,1}^{R} \\ x_{3,n}^{R} \\ \cdot \\ \cdot \\ x_{p-2,n}^{R} \\ x_{p-1,1}^{R} \end{bmatrix} \tag{5.19}$$

Moreover, as we state in stage (2) x_i^R, x_i^{UH} and x_i^{LH} are found by using Equations (5.15) -(5.17). It is possible to use different solution methods for these tridiagonal system of equations such as LU decomposition and Gauss Elimination. Mattor *et al.* (1995) employs the LU decomposition algorithm because of its efficiency. In addition to that they present an algorithm to reduce the number of computations in the LU decomposition method by exploiting the overlapping calculations and elements with value 0 in Equations (5.15) -(5.17). Because of its great advantage in our problem we also implement this algorithm. The algorithm is given below

$$w_1 = \frac{-c_1}{b_1} \text{ for } j = 1, \text{ then } w_j = \frac{-c_j}{b_j + a_j w_{j-1}} \text{ for } j = 2, 3, ..., n$$

$$\gamma_1 = \frac{d_1}{b_1} \text{ for } j = 1, \text{ then } \gamma_j = \frac{d_j + a_j \gamma_{j-1}}{b_j + a_j w_{j-1}} \text{ for } j = 2, 3, ..., n$$

then by using w_j and γ_j for $j = 1, 2, ..., n$

$$x_n^R = \gamma_n \text{ for } j = n, \text{ then } x_j^R = \gamma_j - w_j x_{j+1}^R \text{ for } j = n-1, n-2, ..., 1$$

$$x_n^{LH} = -w_n \text{ for } j = n, \text{ then } x_j^{LH} = -w_j x_{j+1}^{LH} \text{ for } j = n-1, n-2, ..., 1$$

and finally

$$w_n^{UH} = \frac{-a_n}{b_n} \text{ for } j = n, \text{ then } w_j^{UH} = \frac{-a_j}{b_j + c_j w_{j+1}^{UH}} \text{ for } j = n-1, n-2, ..., 1$$

$$x_1^{UH} = -w_1^{UH} \text{ for } j = 1, \text{ then } x_j^{UH} = -w_j^{UH} x_{j-1}^{UH} \text{ for } j = 2, 3, ..., n.$$

As in our problem where tridiagonal matrix does not change at different time columns this algorithm is extra efficient. Since we can compute $w_j, w_j^{UH}, 1\,/\,(b_j + a_j w_j), x_j^{LH}$ and use for all time columns. Finally, note that we form the reduced system on all of the processors and each processor solves its own system. To accomplish these, we broadcast the values that are needed to form the reduced matrix from each processor to all. Then to solve the reduced matrix, we use LU Decomposition method in all parallel processors.

5.4.3. Iterative Methods

In an iterative method, one tries to reach the solution by a number of steps in which he/she uses the previous approximation to compute the current one. Whenever the absolute difference between previous and current approximations drops to a level we want to reach we stop the iteration. However, this is not always the case. Increasing the number of steps may not result in a converging iterations. In addition as it is expressed in Smith (1978), converging rapidly to exact values within an aimed accuracy is more important than trying to find the exact solutions in a finite number of steps. Therefore, it is impossible to obtain exact solutions in a finite number of steps by iterative methods.

There are several iterative methods in the literature. We only mention the ones that are presented in Smith (1978). Jacobi, Gauss-Seidel and Successive over-relaxation

methods are described in this section. We try to solve a small tridiagonal system of linear equations shown in Figure 5.11 to describe these methods.

$$\begin{bmatrix} b_1 & -c_1 & & \\ -a_2 & b_2 & -c_2 & \\ & -a_3 & b_3 & -c_3 \\ & & -a_4 & b_4 \end{bmatrix} \begin{bmatrix} \check{C}_1 \\ \check{C}_2 \\ \check{C}_3 \\ \check{C}_4 \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \end{bmatrix}$$

Figure 5.11. Matrix representation of a tridiagonal system of linear equations

The tridiagonal system given in Figure 5.11 can also be written as

$$\begin{aligned} \check{C}_1 &= \frac{1}{b_1}(d_1 + c_1\check{C}_2) \\ \check{C}_2 &= \frac{1}{b_2}(d_2 + a_2\check{C}_1 + c_2\check{C}_3) \\ \check{C}_3 &= \frac{1}{b_3}(d_3 + a_3\check{C}_2 + c_3\check{C}_4) \\ \check{C}_4 &= \frac{1}{b_4}(d_4 + a_4\check{C}_3) \end{aligned} \tag{5.20}$$

5.4.3.1. Jacobi Method. The jacobi iteration formulas for Equations (5.20) are given below. In the formulas $\check{C}_j^{(m)}$ stands for the current value of $\check{C}_j$ and $\check{C}_j^{(m-1)}$ stands for the previous value of $\check{C}_j$ for $j = 1, 2, ..., 4$. As it is obvious from the equations stated below we just use the previous approximations for variables even if at that time we have the current values for some variables. This makes the algorithm slower.

$$\begin{aligned} \check{C}_1^{(m)} &= \frac{1}{b_1}(d_1 + c_1\check{C}_2^{(m-1)}) \\ \check{C}_2^{(m)} &= \frac{1}{b_2}(d_2 + a_2\check{C}_1^{(m-1)} + c_2\check{C}_3^{(m-1)}) \\ \check{C}_3^{(m)} &= \frac{1}{b_3}(d_3 + a_3\check{C}_2^{(m-1)} + c_3\check{C}_4^{(m-1)}) \\ \check{C}_4^{(m)} &= \frac{1}{b_4}(d_4 + a_4\check{C}_3^{(m-1)}) \end{aligned} \tag{5.21}$$

5.4.3.2. Gauss-Seidel Method. In this iteration method the disadvantage mentioned in Jacobi method is overcomed. As soon as current values are computed they are used in the corresponding Equations (5.20).

$$\begin{aligned}
\check{C}_1^{(m)} &= \frac{1}{b_1}(d_1 + c_1\check{C}_2^{(m-1)}) \\
\check{C}_2^{(m)} &= \frac{1}{b_2}(d_2 + a_2\check{C}_1^{(m)} + c_2\check{C}_3^{(m-1)}) \\
\check{C}_3^{(m)} &= \frac{1}{b_3}(d_3 + a_3\check{C}_2^{(m)} + c_3\check{C}_4^{(m-1)}) \\
\check{C}_4^{(m)} &= \frac{1}{b_4}(d_4 + a_4\check{C}_3^{(m)})
\end{aligned} \tag{5.22}$$

5.4.3.3. Successive Over-Relaxation Method. If the Equations (5.22) are written in the form

$$\begin{aligned}
\check{C}_1^{(m)} &= \check{C}_1^{(m-1)} + \frac{1}{b_1}(d_1 - b_1\check{C}_1^{(m-1)} + c_1\check{C}_2^{(m-1)}) \\
\check{C}_2^{(m)} &= \check{C}_2^{(m-1)} + \frac{1}{b_2}(d_2 - b_2\check{C}_2^{(m-1)} + a_2\check{C}_1^{(m)} + c_2\check{C}_3^{(m-1)}) \\
\check{C}_3^{(m)} &= \check{C}_3^{(m-1)} + \frac{1}{b_3}(d_3 - b_3\check{C}_3^{(m-1)} + a_3\check{C}_2^{(m)} + c_3\check{C}_4^{(m-1)}) \\
\check{C}_4^{(m)} &= \check{C}_4^{(m-1)} + \frac{1}{b_4}(d_4 - b_4\check{C}_4^{(m-1)} + a_4\check{C}_3^{(m)})
\end{aligned}$$

then it is easy to see that we simply add a correction to the previous approximation to find the current one. Successive Over-Relaxation method tells us that an increase in the correction may speed up the convergence. That is, one can simply multiply the correction by an amount w (generally $1 < w < 2$). So the Equations (5.22) change into

$$\begin{aligned}
\check{C}_1^{(m)} &= \check{C}_1^{(m-1)} + \frac{w}{b_1}(d_1 - b_1\check{C}_1^{(m-1)} + c_1\check{C}_2^{(m-1)}) \\
\check{C}_2^{(m)} &= \check{C}_2^{(m-1)} + \frac{w}{b_2}(d_2 - b_2\check{C}_2^{(m-1)} + a_2\check{C}_1^{(m)} + c_2\check{C}_3^{(m-1)}) \\
\check{C}_3^{(m)} &= \check{C}_3^{(m-1)} + \frac{w}{b_3}(d_3 - b_3\check{C}_3^{(m-1)} + a_3\check{C}_2^{(m)} + c_3\check{C}_4^{(m-1)}) \\
\check{C}_4^{(m)} &= \check{C}_4^{(m-1)} + \frac{w}{b_4}(d_4 - b_4\check{C}_4^{(m-1)} + a_4\check{C}_3^{(m)})
\end{aligned} \tag{5.23}$$

Although for an arbitrary system of linear equations there is no formula to calculate the optimum value (w_b) for w, fortunately there exists a formula for tridiagonal

matrices proposed firstly by Young (1954). The formula is stated as

$$w_b = \frac{2}{1 + \sqrt{(1 - \rho^2(B))}}$$

where $\rho(B)$ is the spectral radius of the Jacobi iteration matrix. To calculate $\rho(B)$ we use the Lyusternik (1947)'s method. For sufficiently large m, by implementing the Jacobi method,

$$\rho(B) \simeq \frac{\left\| d^{(m)} \right\|}{\left\| d^{(m-1)} \right\|}$$

where the norm of $d^{(m)}$ is given by

$$\left\| d^{(m)} \right\| = \max_{1 \leq i \leq N-1} \left| \check{C}_i^{(m)} - \check{C}_i^{(m-1)} \right|$$

or

$$\left\| d^{(m)} \right\| = \left| \check{C}_1^{(m)} - \check{C}_1^{(m-1)} \right| + \left| \check{C}_2^{(m)} - \check{C}_2^{(m-1)} \right| + ... + \left| \check{C}_{N-1}^{(m)} - \check{C}_{N-1}^{(m-1)} \right|.$$

Since finding w_b at each time column using the Jacobi method and then using this value in the implementation of the the Successive Over-Relaxation method is very time consuming, we find w_b at the first time column and then use this value for the next time columns. It is a practical and logical solution since at each time column only the right hand side changes for the system of linear equations. The structure of the system of linear equations at each time column is very similar.

5.4.3.4. Parallel Implementation of Iterative Methods. We try to parallelize iterative methods in an efficient manner in this section. Since we have knowledge of efficiency of each iterative algorithm, we do not need to parallelize all of the algorithms. We only implement two of them; Gauss-Seidel and Successive Over-Relaxation method in parallel.

Parallel Implementation of Gauss-Seidel and Successive Over-Relaxation Methods. We introduce the approach on an illustrative example. In Figure 5.12, there are nine y (or z) intervals along each time column, and number of time columns does not differ. Because of the boundary conditions, we know all the relative prices at time column i and relative prices at $j = 0$ and $j = 9$ at time column $i - 1$. Moreover, from Equations (5.12) and (5.13) we can leave $\check{C}_{i-1,\,j}$ at one side in the equality shown in Equation (5.24). Since it is the variable for which we want to determine what the dependencies are. So that we can perform the send-receive relationships between processors correctly. It is easy to see from Equation (5.24) that $\check{C}_{i-1,\,j}$ is dependent on $\check{C}_{i-1,\,j-1}$, $\check{C}_{i-1,\,j+1}$, $\check{C}_{i,\,j-1}$, $\check{C}_{i,\,j}$ and $\check{C}_{i,\,j+1}$. We divide the y (or z) internal mesh points to three processors (P_0, P_1 and P_2) all over the time columns. This algorithm can be easily extended for number of processors greater than three.

$$\check{C}^{e}_{i-1,\,j} = \frac{D_{i,j} - a_j \check{C}_{i-1,\,j-1} - c_j \check{C}_{i-1,\,j+1}}{b_j} \tag{5.24}$$

where

$$D_{i,j} = (-A_j + B_j)\,\check{C}_{i,\,j-1} + \left(-\frac{1}{\triangle t} + 2A_j\right)\check{C}_{i,\,j} + (-A_j - B_j)\,\check{C}_{i,\,j+1}.$$

For an arbitrary time column i, we demonstrate the works done by each processor. In the presentation, $(i, 3)$ stands for the relative price at time column i and for $j = 3$;

The sequence of operations done by P_0 is

1- Send $(i, 3)$ to P_1

2- Receive $(i, 4)$ from P_1

3- Send $(i - 1, 3)$ to P_1

4- Receive $(i - 1, 4)$ from P_1

5- Calculate unknown relative prices at the territory of P_0 at time column $i-1$.

Although steps 1 and 2 are executed only once, steps 3-5 are executed for every iteration.

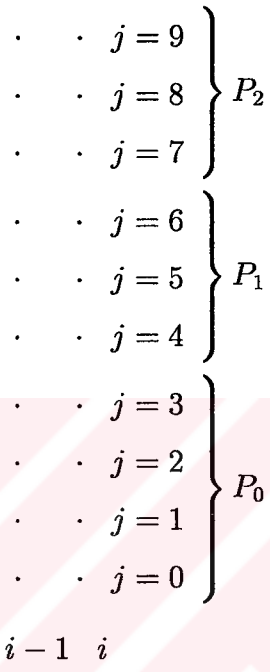

Figure 5.12. Partition of mesh points to processors along each time column

The sequence of operations done by P_1 is

1- Send $(i, 4)$ to P_0

2- Receive $(i, 3)$ from P_0

3- Send $(i, 6)$ to P_2

4- Receive $(i, 7)$ from P_2

5- Send $(i-1, 4)$ to P_0

6- Receive $(i-1,3)$ from P_0

7- Send $(i-1,6)$ to P_2

8- Receive $(i-1,7)$ from P_2

9- Calculate unknown relative prices at the territory of P_1 at time column $i-1$.

Although steps 1-4 are executed only once, steps 5-9 are executed for every iteration.

The sequence of operations done by P_2 is

1- Send $(i,7)$ to P_1

2- Receive $(i,6)$ from P_1

3- Send $(i-1,7)$ to P_1

4- Receive $(i-1,6)$ from P_1

5- Calculate unknown relative prices at the territory of P_2 at time column $i-1$.

Although steps 1 and 2 are executed only once, steps 3-5 are executed for every iteration.

We note that this parallel algorithm is not a direct parallel application of Gauss-Seidel method. Because as stated before according to Gauss-Seidel method we should use the most current approximations in the formulas. However, in the parallel algorithm we introduce as an example $\check{C}_{i-1,3}$ is not send to the next processor until the next iteration so that we use the previous approximation for $\check{C}_{i-1,3}$ in the calculation of $\check{C}_{i-1,4}$ for Figure 5.12. Of course this is only true at boundaries of processors even if

they affect the whole mesh points following $\check{C}_{i-1,3}$.

Parallel implementation of Successive Over-Relaxation method is not different from the Gauss-Seidel method in the send-receive relationships of processors. This is obvious from the Equations (5.22) and (5.23) that only $\check{C}_{i-1}^{(m-1)}$ is added as a difference to the Successive Over-Relaxation method equations. And, of course, we have this value in the required processor. So that only the difference in the implementation is including $\check{C}_{i-1}^{(m-1)}$ plus w in the equations.

6. COMPARISON OF NUMERICAL METHODS

In this section we compare the methods that are introduced throughout the previous sections by executing their codes on Advanced System for Multi-computer Applications' (ASMA) Linux PC cluster. ASMA is located in the Department of Computer Engineering in Boğaziçi University, Turkey. There are system parameters that may affect the outputs of the codes. These parameters and ASMA specifications for these parameters are given in Table A.1 (Gürdag, 2002).

Our criterions in this comparison are the wall-clock time of computers in executing the codes and accuracy of the price for European option. As stated earlier to get an information about how accurate we are in Asian option pricing, we also price European option which have a similar structure with Asian option by the same approach used for the Asian option. We can check whether or not the per cent error in European option pricing is an almost accurate indicator of the per cent error in pricing of Asian option by examining the Asian option prices based on Rogers and Shi (1995), Zvan *et al.* (1997) and Lim (2002) in Table B.1. PDE stands for the P.D.E. solution by Zvan *et al.* (1997). LB is the lower bound on Asian option price given by Rogers and Shi (1995). RNI stands for the price given by Lim (2002). The Asian and European option prices are evaluated at $t = 0$, where $x_0 = y_0 = K / S_0$.

First of all, we compare the explicit and Crank-Nicolson implicit numerical finite-difference methods on a single-processor. We prefer to use LU decomposition method in the implementation of the Crank-Nicolson implicit finite-difference method for comparison purposes since it is more efficient than Gauss Elimination method in our problem. The results are given in Table B.2 and Table B.3. Before the interpretations of the results, we explain the notation used and why we implement the comparison this way. M and N are used for the number of time and y intervals in the solution of the finite-difference method. Since maturity of the Asian option is taken as one year, $(1/M)$ gives the finite-spacing Δt. Moreover, $(1/N)$ gives the finite-spacing Δy because of the fact that $y \, \epsilon[0, 1]$.

To compare two things they should be weighed on the same parameters. However, this is not possible in the case of a comparison between explicit and implicit finite-difference method because of the stability criterions on the explicit method. So we keep the N the same for both of the methods and look for M that makes the explicit method stable for N. Even though it is possible to apply the same M for the implicit method, this brings no improvement to the accuracy of the implicit method at the expense of time. Thus, we keep the M and N the same for all implicit and iterative methods throughout the results. E stands for the per cent error in European option pricing, t_A is the time spent on pricing of Asian option in seconds. $\bar{t}_A$ is the average time spent on pricing of Asian option under different values of volatility and strike price for specified M and N values in seconds. And, finally, C^a is the price computed for the Asian option for the given interest rate, volatility, strike price, initial underlying security price and maturity.

As it can be seen from Tables B.2 and B.3 that explicit method requires very large values of M for a fixed N as the volatility increases to satisfy the stability criterion. So, in the explicit method, we only store the data that belongs to the last time column to avoid memory problems.

It is easy to see from Tables B.2 and B.3 that implicit method is better at converging to the accurate prices in the expense of time for small volatilities. However, as volatility increases explicit method looses its last competitive edge (quickness) because of the increase in M. If we check the compatibility of per cent error in European option pricing with per cent error in Asian option pricing by accepting the prices given in Table B.1 as true prices, we can conclude that almost perfect fit is on hand especially for the implicit method.

Although the explicit method is much worse than Crank-Nicolson implicit method, we want to see the performance of the parallel implementation of the explicit method over single-processor case. By interpreting Table B.4 and B.5, it is easy to conclude that parallel implementation of explicit method with 4 processors is up to three times faster than single-processor case for larger values of M and N. In the case of lower val-

ues of M and N, this statement is obviously bias because of time spend on send-receive relationships between processors.

Now, we are convinced to apply the implicit method in pricing of Asian option after evaluating the performances of the explicit and implicit methods over different values of interest rate, volatility and strike price. What is next is to compare the efficiencies of different parallel algorithms designed to solve linear systems of equations which are the outcome of Crank-Nicolson implicit method. We first compare Block Partition and Parallel Factorization algorithms by using four parallel processors in Tables B.6 and B.7 . Then, Gauss-Seidel and Successive-Over Relaxation iterative methods are compared in Tables B.8 and B.9 on single-processor. Parallel implementation of Gauss-Seidel and Successive-Over Relaxation methods are compared in Tables B.10 and B.11 by using four parallel processors. Since all tables are formed for the same values of parameters for a given interest rate, it is obviously right thing to make a comparison between the different algorithms given on different tables for the same values of interest rate.

After examining Tables B.6 and B.7, it is easy to conclude that Block Partition and Parallel Factorization algorithms give the same prices with the single-processor implicit method given in Tables B.2 and B.3. Although, there is an improvement in time usage in Block Partition algorithm for M and N values equal to 2000, Parallel Factorization algorithm is worse than single-processor implicit method for this level of M and N values using four parallel processors. In Table B.6, average time spent on Asian option pricing is equal to 10.2 second for Block Partition algorithm and 42.3 second for Parallel Factorization algorithm for M and N values equal to 2000. Moreover, in Table B.7, average time spent on Asian option pricing is equal to 8.9 second for Block Partition algorithm and 41.1 second for Parallel Factorization algorithm for M and N values equal to 2000.

We also give results for Block Partition and Parallel Factorization algorithms using three parallel processors in Tables B.12 and B.13. Since these are the best parallel algorithms for solving system of linear equations as it will be seen, and we want to

evaluate performances of them for different number of parallel processors. Although we do not give any results, as M and N increases Parallel Factorization algorithm provides great efficiencies in time usage so it is much more better than Block Partition algorithm for greater values of M and N. However, it would not be wise to implement the codes for higher values of M and N, because we already get accurate prices for this level of M and N.

Although parameter changes in Table B.6 and B.7 for the same values of M and N for both of methods should not affect the time usage, there are variations in time usage. This is the result of the fact that both of methods highly depend on send-receive relationships between processors. So, they are quite sensitive to traffic on ASMA.

In iterative methods, as stated before whenever the absolute difference between previous and current approximation drops to a level we want to reach we stop the iteration. We take this level of accuracy as 0.0000001 for Tables B.8, B.9, B.10 and B.11. It is easy to see from Tables B.7 and B.8 that for small values of volatilities as we increase M and N we converge to the accurate prices given in Table B.1 even sometimes per cent errors in European option prices show reverse for both Gauss-Seidel and Successive-Over Relaxation methods. However, for higher volatilities as we increase M and N computed prices are alienated from the accurate prices given in Table B.1. Because, the level of accuracy (0.0000001) used is not enough anymore to converge to the accurate prices. After evaluating both of methods together, we can compare them by interpreting the Tables B.8 and B.9. Successive-Over Relaxation method is better than Gauss-Seidel method at the expense of time for small volatilities. Moreover, Successive-Over Relaxation method is better in all aspects for higher values of volatilities.

Although we prefer to use Successive-Over Relaxation method instead of Gauss-Seidel method, we want to access efficiency of parallel computing in both of the methods. To get this information, we compare the prices and time usages given on Tables B.8 and B.10 for $r = 0.09$ and Tables B.9 and B.11 for $r = 0.15$ for both of the methods. We conclude that parallel implementations of both methods are up to twice faster

than single-processor implementations for M and N equal to 2000. As stated earlier parallel implementations of Gauss-Seidel and Successive-Over Relaxation methods are not a direct parallel application of these methods. Although most of the parallel implementation prices are the same with the single-processor implementations for both of the methods, it is interesting to note that the rest of prices computed by using parallel implementation of Gauss-Seidel and Successive-Over Relaxation methods are more accurate than single-processor implementations for both of the methods.

Now, we want to see how the results would be affected if we had used three parallel processors instead of four for Block Partition and Parallel Factorization algorithms. The results are given in Tables B.12 and B.13 for $r = 0.09$ and $r = 0.15$ in that order. We do this for only these two algorithms because of their promising performances using four parallel processors. In Table B.12, average time spent on Asian option pricing is equal to 7.2 second for Block Partition algorithm and 9.4 second for Parallel Factorization algorithm for M and N values equal to 2000. Moreover, in Table B.13, average time spent on Asian option pricing is equal to 8 second for Block Partition algorithm and 10.2 second for Parallel Factorization algorithm for M and N values equal to 2000. If we compare their average time usage ($\overline{t}_A$), Block Partition algorithm, which uses three parallel processors is a little bit faster than four parallel processor case of the same algorithm for M and N equal to 2000. Moreover, average time usage of Parallel Factorization algorithm drops drastically and close to the Block Partition algorithm's value. As a conclusion, time usages of Block Partition and Parallel Factorization algorithms executed on three parallel processors are approximately half of the single-processor implementation of Crank-Nicolson implicit method using LU decomposition method given in Tables B.2 and B.3.

7. SUMMARY AND CONCLUSIONS

In this thesis, we first discuss the methodology we followed to price European-style "backward-starting fixed-strike" Asian and standard European options using the P.D.E. approach. Then, a change of variables is implemented to reduce the unbounded domain to a bounded one. We apply both explicit and Crank-Nicolson implicit finite-difference methods to solve the one-state-variable partial differential equations at hand. Parallel implementation of the explicit finite-difference method is easy because of the simplicity of the method. On the other hand, we try to solve large sizes of system of linear equations in the Crank-Nicolson implicit finite-difference method. We implemented Gauss Elimination, LU decomposition and iterative methods (Gauss-Seidel and Successive Over-Relaxation) to solve these systems of linear equations. Furthermore, we implemented some of the parallel algorithms given in literature for solving system of linear equations in parallel.

Finally, we compare all of the algorithms and assess the efficiency of parallel computing. We conclude that Crank-Nicolson implicit finite difference method is better than the explicit-finite difference method in European-style "backward-starting fixed-strike" Asian option pricing. Moreover, LU decomposition method is the best in solving the system of linear equations which is the outcome of Crank-Nicolson implicit finite difference method on single-processor. Moreover, the best parallel algorithms among the algorithms we implemented are Block Partition and Parallel Factorization algorithms. Their three parallel-processor implementations' time usages are approximately half of the single-processor implementation of Crank-Nicolson implicit method to attain a reasonable accuracy. Furthermore, we presented practical results indicating per cent error in European option pricing is a very good estimator of error in Asian option pricing by using the same pricing methodology. However, this is not practical to apply unless one accepts to double the time spend on pricing of an Asian option. The time usages are approximately equal for Asian and European options by using all of the methods mentioned above.

APPENDIX A: SYSTEM PARAMETER VALUES FOR ASMA

Table A.1. System parameter values for ASMA

System Parameter	ASMA
OS	RedHat Linux 8.0
Linux kernel	2.4.18-14
C Compiler	gcc 3.2
Compiler options used	optimization level 2
MPI library used	MPICH 1.2.1 over TCP
Network Switch Environment	HP4000M Fast Ethernet Switch
Network Interface Card (NIC)	Intel Ether Express 100
Raw network bandwidth	100 Mb/s
MPI bandwidth	60 Mb/s
MPI latency	120 μ s
Per-node memory	512 Mb DDR
Per-node CPU	AMD Athlon XP 2100+ (1733 Mhz)
Node disk access type	local
Disk bandwidth (read-write)	(20 Mb/s) / (6 Mb/s)
Processor load	idle
Network load	flactuating
PL exponents (β_{in}/β_{out})	2.1/2.7
Convergence criteria (Δ)	0.0001
Dampening factor (D)	0.85

APPENDIX B: INITIAL PRICES FOR ASIAN OPTIONS

Table B.1. Accurate initial values for continuous fixed strike Asian Call options when $S_0 = 100$ and $T = 1$

σ	K	$r = 0.09$		$r = 0.15$		
		LB	RNI	PDE	LB	RNI
	95	8.809	8.809	11.094	11.094	11.094
0.05	100	4.308	4.308	6.793	6.794	6.794
	105	0.958	0.958	2.748	2.744	2.744
	90	13.385	13.385	15.399	15.399	15.398
0.10	100	4.915	4.915	7.030	7.028	7.027
	110	0.630	0.630	1.410	1.413	1.413
	90	13.831	13.831	15.643	15.641	15.641
0.2	100	6.777	6.776	8.409	8.408	8.408
	110	2.545	2.545	3.554	3.554	3.554
	90	14.983	14.983	16.514	16.512	16.512
0.30	100	8.827	8.829	10.210	10.208	10.210
	110	4.695	4.696	5.729	5.728	5.730

Table B.2. Initial values for continuous fixed strike Asian call options when $r = 0.09$, $S_0 = 100$ and $T = 1$ using Explicit and Implicit methods

		Explicit Method					Implicit Method				
σ	K	M	N	E	t_A	C^a	M	N	E	t_A	C^a
		600	500	0.1126	0	8.7292	500	500	0.0033	1	8.8104
	95	1400	1000	0.0580	3	8.7564	1000	1000	0.0007	3	8.8086
		3500	2000	0.0296	10	8.7787	2000	2000	0.0002	14	8.8088
		600	500	0.2171	0	4.6251	500	500	0.0256	1	4.2390
0.05	100	1400	1000	0.1040	2	4.4873	1000	1000	0.0060	2	4.2938
		3500	2000	0.0505	11	4.4082	2000	2000	0.0013	14	4.3048
		600	500	2.4314	0	1.7358	500	500	0.1157	1	0.9795
	105	1400	1000	1.2531	2	1.4626	1000	1000	0.0255	3	0.9613
		3500	2000	0.6369	10	1.2762	2000	2000	0.0050	14	0.9590
		900	500	0.0193	1	13.3108	500	500	0.0023	1	13.3830
	90	2400	1000	0.0101	4	13.3425	1000	1000	0.0005	3	13.3846
		7500	2000	0.0053	22	13.3623	2000	2000	0.0001	14	13.3851
		900	500	0.4897	0	5.3470	500	500	0.0151	1	4.8961
0.10	100	2400	1000	0.2487	4	5.1720	1000	1000	0.0029	3	4.9105
		7500	2000	0.1251	22	5.0642	2000	2000	0.0004	14	4.9140
		900	500	2.3856	0	1.1125	500	500	0.0004	1	0.6472
	110	2400	1000	1.2016	4	0.9133	1000	1000	0.0029	3	0.6343
		7500	2000	0.6067	22	0.7935	2000	2000	0.0006	14	0.6313
		2000	500	0.0868	1	13.9319	500	500	0.0015	1	13.8274
	90	7000	1000	0.0434	11	13.8861	1000	1000	0.0002	3	13.8305
		24000	2000	0.0216	70	13.8599	2000	2000	0.0000	14	13.8312
		2000	500	0.2793	2	7.1286	500	500	0.0047	1	6.7734
0.20	100	7000	1000	0.1404	10	6.9734	1000	1000	0.0008	3	6.7764

Table B.2. (continued)

		Explicit Method					Implicit Method				
σ	K	M	N	E	t_A	C^a	M	N	E	t_A	C^a
		24000	2000	0.0703	70	6.8812	2000	2000	0.0001	14	6.7771
		2000	500	0.6081	2	2.9394	500	500	0.0015	1	2.5484
	110	7000	1000	0.3038	10	2.7650	1000	1000	0.0010	3	2.5466
		24000	2000	0.1523	70	2.6622	2000	2000	0.0000	14	2.5463
		3600	500	0.0845	3	15.1223	500	500	0.0010	1	14.9818
	90	14000	1000	0.0421	21	15.0575	1000	1000	0.0001	3	14.9834
		52000	2000	0.0210	152	15.0217	2000	2000	0.0001	14	14.9838
		3600	500	0.1728	3	9.0864	500	500	0.0022	1	8.8270
0.30	100	14000	1000	0.0865	20	8.9665	1000	1000	0.0003	3	8.8284
		52000	2000	0.0433	153	8.8999	2000	2000	0.0000	14	8.8287
		3600	500	0.2959	3	4.9857	500	500	0.0010	1	4.6971
	110	14000	1000	0.1474	20	4.8511	1000	1000	0.0004	3	4.6967
		52000	2000	0.0738	153	4.7765	2000	2000	0.0000	14	4.6967

Table B.3. Initial values for continuous fixed strike Asian call options when $r = 0.15$, $S_0 = 100$ and $T = 1$ using Explicit and Implicit methods

		Explicit Method					Implicit Method				
σ	K	M	N	E	t_A	C^a	M	N	E	t_A	C^a
		600	500	0.1555	0	10.9733	500	500	0.0010	1	11.0938
	95	1400	1000	0.0782	2	11.0289	1000	1000	0.0002	3	11.0941
		3500	2000	0.0395	11	11.0603	2000	2000	0.0001	14	11.0941
		600	500	0.1861	0	6.8105	500	500	0.0018	1	6.7872
0.05	100	1400	1000	0.0977	2	6.7884	1000	1000	0.0005	3	6.7920
		3500	2000	0.0503	10	6.7856	2000	2000	0.0002	14	6.7937
		600	500	0.1274	1	3.2946	500	500	0.0280	1	2.6685
	105	1400	1000	0.0477	2	3.0890	1000	1000	0.0068	3	2.7253
		3500	2000	0.0190	10	2.9550	2000	2000	0.0016	14	2.7398
		900	500	0.0908	1	15.2939	500	500	0.0013	1	15.3983
	90	2400	1000	0.0464	3	15.3423	1000	1000	0.0003	3	15.3986
		7500	2000	0.0236	22	15.3693	2000	2000	0.0000	14	15.3987
		900	500	0.1442	1	7.2616	500	500	0.0095	1	7.0103
0.10	100	2400	1000	0.0712	4	7.1612	1000	1000	0.0022	3	7.0235
		7500	2000	0.0351	22	7.1033	2000	2000	0.0005	14	7.0267
		900	500	1.5373	0	2.004	500	500	0.0216	1	1.4218
	110	2400	1000	0.7800	4	1.7702	1000	1000	0.0064	3	1.4153
		7500	2000	0.3947	22	1.6235	2000	2000	0.0010	14	1.4141
		2000	500	0.0422	1	15.6784	500	500	0.0017	1	15.6385
	90	7000	1000	0.0208	11	15.6613	1000	1000	0.0003	3	15.6409
		24000	2000	0.0103	70	15.6517	2000	2000	0.0000	14	15.6416
		2000	500	0.2300	1	8.6952	500	500	0.0043	1	8.4039
0.20	100	7000	1000	0.1155	11	8.5682	1000	1000	0.0008	3	8.4076

Table B.3. (continued)

		Explicit Method					Implicit Method				
σ	K	M	N	E	t_A	C^a	M	N	E	t_A	C^a
		24000	2000	0.0578	70	8.4931	2000	2000	0.0001	14	8.4085
		2000	500	0.5806	2	3.9629	500	500	0.0035	1	3.5560
	110	7000	1000	0.2913	10	3.7834	1000	1000	0.0011	3	3.5556
		24000	2000	0.1462	70	3.6766	2000	2000	0.0000	14	3.5556
		3600	500	0.0697	3	16.6116	500	500	0.0010	1	16.5108
	90	14000	1000	0.0347	23	16.5652	1000	1000	0.0002	3	16.5124
		52000	2000	0.0173	153	16.5397	2000	2000	0.0000	14	16.5128
		3600	500	0.1652	3	10.4382	500	500	0.0022	1	10.2077
0.30	100	14000	1000	0.0826	23	10.3319	1000	1000	0.0004	3	10.2093
		52000	2000	0.0413	153	10.2728	2000	2000	0.0001	14	10.2097
		3600	500	0.3026	2	6.0183	500	500	0.0015	1	5.7299
	110	14000	1000	0.1512	21	5.8842	1000	1000	0.0004	3	5.7300
		52000	2000	0.0757	153	5.8098	2000	2000	0.0001	14	5.7301

Table B.4. Initial values for continuous fixed strike Asian call options when $r = 0.09$, $S_0 = 100$ and $T = 1$ using Explicit method with single-processor and parallel implementation of Explicit method with four parallel processors

				Explicit Method			Parallel Explicit Method		
σ	K	M	N	E	t_A	C^a	E	t_A	C^a
		600	500	0.1126	0	8.7292	0.1126	0	8.7292
	95	1400	1000	0.0580	3	8.7564	0.0580	1	8.7564
		3500	2000	0.0296	10	8.7787	0.0296	3	8.7787
		600	500	0.2171	0	4.6251	0.2171	1	4.6251
0.05	100	1400	1000	0.1040	2	4.4873	0.1040	1	4.4873
		3500	2000	0.0505	11	4.4082	0.0505	3	4.4082
		600	500	2.4314	0	1.7358	2.4314	0	1.7358
	105	1400	1000	1.2531	2	1.4626	1.2531	1	1.4626
		3500	2000	0.6369	10	1.2762	0.6369	4	1.2762
		900	500	0.0193	1	13.3108	0.0193	0	13.3108
	90	2400	1000	0.0101	4	13.3425	0.0101	2	13.3425
		7500	2000	0.0053	22	13.3623	0.0053	7	13.3623
		900	500	0.4897	0	5.3470	0.4897	1	5.3470
0.10	100	2400	1000	0.2487	4	5.1720	0.2487	1	5.1720
		7500	2000	0.1251	22	5.0642	0.1251	8	5.0642
		900	500	2.3856	0	1.1125	2.3856	0	1.1125
	110	2400	1000	1.2016	4	0.9133	1.2016	1	0.9133
		7500	2000	0.6067	22	0.7935	0.6067	7	0.7935
		2000	500	0.0868	1	13.9319	0.0868	1	13.9319
	90	7000	1000	0.0434	11	13.8861	0.0434	4	13.8861
		24000	2000	0.0216	70	13.8599	0.0216	24	13.8599
		2000	500	0.2793	2	7.1286	0.2793	1	7.1286
0.20	100	7000	1000	0.1404	10	6.9734	0.1404	5	6.9734
		24000	2000	0.0703	70	6.8812	0.0703	25	6.8812
		2000	500	0.6081	2	2.9394	0.6081	1	2.9394
	110	7000	1000	0.3038	10	2.7650	0.3038	4	2.7650
		24000	2000	0.1523	70	2.6622	0.1523	24	2.6622
		3600	500	0.0845	3	15.1223	0.0845	2	15.1223
	90	14000	1000	0.0421	21	15.0575	0.0421	9	15.0575
		52000	2000	0.0210	152	15.0217	0.0210	53	15.0217
		3600	500	0.1728	3	9.0864	0.1728	1	9.0864
0.30	100	14000	1000	0.0865	20	8.9665	0.0865	8	8.9665
		52000	2000	0.0433	153	8.8999	0.0433	54	8.8999
		3600	500	0.2959	3	4.9857	0.2959	2	4.9857
	110	14000	1000	0.1474	20	4.8511	0.1474	7	4.8511
		52000	2000	0.0738	153	4.7765	0.0738	53	4.7765

Table B.5. Initial values for continuous fixed strike Asian call options when $r = 0.15$, $S_0 = 100$ and $T = 1$ using Explicit method with single-processor and parallel implementation of explicit method with four parallel processors

				Explicit Method			Parallel Explicit Method		
σ	K	M	N	E	t_A	C^a	E	t_A	C^a
		600	500	0.1555	0	10.9733	0.1555	0	10.9733
	95	1400	1000	0.0782	2	11.0289	0.0782	1	11.0289
		3500	2000	0.0395	11	11.0603	0.0395	4	11.0603
		600	500	0.1861	0	6.8105	0.1861	0	6.8105
0.05	100	1400	1000	0.0977	2	6.7884	0.0977	1	6.7884
		3500	2000	0.0503	10	6.7856	0.0503	3	6.7856
		600	500	0.1274	1	3.2946	0.1274	0	3.2946
	105	1400	1000	0.0477	2	3.0890	0.0477	1	3.0890
		3500	2000	0.0190	10	2.9550	0.0190	4	2.9550
		900	500	0.0908	1	15.2939	0.0908	0	15.2939
	90	2400	1000	0.0464	3	15.3423	0.0464	2	15.3423
		7500	2000	0.0236	22	15.3693	0.0236	8	15.3693
		900	500	0.1442	1	7.2616	0.1442	0	7.2616
0.10	100	2400	1000	0.0712	4	7.1612	0.0712	2	7.1612
		7500	2000	0.0351	22	7.1033	0.0351	7	7.1033
		900	500	1.5373	0	2.004	1.5373	1	2.004
	110	2400	1000	0.7800	4	1.7702	0.7800	1	1.7702
		7500	2000	0.3947	22	1.6235	0.3947	8	1.6235
		2000	500	0.0422	1	15.6784	0.0422	0	15.6784
	90	7000	1000	0.0208	11	15.6613	0.0208	4	15.6613
		24000	2000	0.0103	70	15.6517	0.0103	25	15.6517
		2000	500	0.2300	1	8.6952	0.2300	1	8.6952
0.20	100	7000	1000	0.1155	11	8.5682	0.1155	4	8.5682
		24000	2000	0.0578	70	8.4931	0.0578	24	8.4931
		2000	500	0.5806	2	3.9629	0.5806	1	3.9629
	110	7000	1000	0.2913	10	3.7834	0.2913	4	3.7834
		24000	2000	0.1462	70	3.6766	0.1462	25	3.6766
		3600	500	0.0697	3	16.6116	0.0697	1	16.6116
	90	14000	1000	0.0347	23	16.5652	0.0347	10	16.5652
		52000	2000	0.0173	153	16.5397	0.0173	51	16.5397
		3600	500	0.1652	3	10.4382	0.1652	2	10.4382
0.30	100	14000	1000	0.0826	23	10.3319	0.0826	8	10.3319
		52000	2000	0.0413	153	10.2728	0.0413	53	10.2728
		3600	500	0.3026	2	6.0183	0.3026	2	6.0183
	110	14000	1000	0.1512	21	5.8842	0.1512	8	5.8842
		52000	2000	0.0757	153	5.8098	0.0757	53	5.8098

Table B.6. Initial values for continuous fixed strike Asian call options when $r = 0.09$, $S_0 = 100$ and $T = 1$ using Block Partition and Parallel Factorization algorithms using four parallel processors

				Block Partition			Parallel Factorization		
σ	K	M	N	E	t_A	C^a	E	t_A	C^a
		500	500	0.0033	2	8.8105	0.0033	10	8.8105
	95	1000	1000	0.0007	3	8.8086	0.0007	14	8.8086
		2000	2000	0.0002	9	8.8088	0.0002	34	8.8088
		500	500	0.0256	0	4.2390	0.0256	12	4.2390
0.05	100	1000	1000	0.0060	3	4.2938	0.0060	18	4.2938
		2000	2000	0.0013	7	4.3048	0.0013	44	4.3048
		500	500	0.1157	1	0.9795	0.1157	7	0.9795
	105	1000	1000	0.0255	3	0.9613	0.0255	18	0.9613
		2000	2000	0.0050	10	0.9590	0.0050	44	0.9590
		500	500	0.0023	2	13.3830	0.0023	11	13.3830
	90	1000	1000	0.0005	5	13.3846	0.0005	23	13.3846
		2000	2000	0.0001	6	13.3851	0.0001	40	13.3851
		500	500	0.0151	2	4.8961	0.0151	12	4.8961
0.10	100	1000	1000	0.0029	6	4.9105	0.0029	26	4.9105
		2000	2000	0.0004	10	4.9140	0.0004	35	4.9140
		500	500	0.0004	1	0.6472	0.0004	9	0.6472
	110	1000	1000	0.0029	6	0.6343	0.0029	17	0.6343
		2000	2000	0.0006	8	0.6313	0.0006	51	0.6313
		500	500	0.0015	2	13.8274	0.0015	7	13.8274
	90	1000	1000	0.0002	5	13.8305	0.0002	23	13.8305
		2000	2000	0.0000	8	13.8312	0.0000	43	13.8312
		500	500	0.0047	3	6.7734	0.0047	8	6.7734
0.20	100	1000	1000	0.0008	2	6.7764	0.0008	19	6.7764
		2000	2000	0.0001	6	6.7771	0.0001	43	6.7771
		500	500	0.0015	2	2.5484	0.0015	14	2.5484
	110	1000	1000	0.0010	3	2.5466	0.0010	30	2.5466
		2000	2000	0.0000	11	2.5463	0.0000	46	2.5463
		500	500	0.0010	1	14.9818	0.0010	6	14.9818
	90	1000	1000	0.0001	4	14.9834	0.0001	19	14.9834
		2000	2000	0.0001	5	14.9838	0.0001	36	14.9838
		500	500	0.0022	1	8.8270	0.0022	7	8.8270
0.30	100	1000	1000	0.0003	3	8.8284	0.0003	17	8.8284
		2000	2000	0.0000	10	8.8287	0.0000	41	8.8287
		500	500	0.0010	2	4.6971	0.0010	12	4.6971
	110	1000	1000	0.0004	2	4.6967	0.0004	16	4.6967
		2000	2000	0.0000	8	4.6967	0.0000	50	4.6967

Table B.7. Initial values for continuous fixed strike Asian call options when $r = 0.15$, $S_0 = 100$ and $T = 1$ using Block Partition and Parallel Factorization algorithms using four parallel processors

				Block Partition			Parallel Factorization		
σ	K	M	N	E	t_A	C^a	E	t_A	C^a
		500	500	0.0010	4	11.0938	0.0010	10	11.0938
	95	1000	1000	0.0002	5	11.0941	0.0002	23	11.0941
		2000	2000	0.0001	7	11.0941	0.0001	39	11.0941
		500	500	0.0018	1	6.7872	0.0018	13	6.7872
0.05	100	1000	1000	0.0005	3	6.7920	0.0005	27	6.7920
		2000	2000	0.0002	9	6.7937	0.0002	40	6.7937
		500	500	0.0280	3	2.6685	0.0280	9	2.6685
	105	1000	1000	0.0068	3	2.7253	0.0068	22	2.7253
		2000	2000	0.0016	10	2.7398	0.0016	50	2.7398
		500	500	0.0013	1	15.3983	0.0013	9	15.3983
	90	1000	1000	0.0003	3	15.3986	0.0003	22	15.3986
		2000	2000	0.0000	8	15.3987	0.0000	46	15.3987
		500	500	0.0095	3	7.0103	0.0095	11	7.0103
0.10	100	1000	1000	0.0022	4	7.0235	0.0022	28	7.0235
		2000	2000	0.0005	5	7.0267	0.0005	36	7.0267
		500	500	0.0216	2	1.4218	0.0216	10	1.4218
	110	1000	1000	0.0064	4	1.4153	0.0064	26	1.4153
		2000	2000	0.0010	8	1.4141	0.0010	41	1.4141
		500	500	0.0017	3	15.6385	0.0017	9	15.6385
	90	1000	1000	0.0003	5	15.6409	0.0003	22	15.6409
		2000	2000	0.0000	7	15.6416	0.0000	36	15.6416
		500	500	0.0043	2	8.4039	0.0043	15	8.4039
0.20	100	1000	1000	0.0008	3	8.4076	0.0008	23	8.4076
		2000	2000	0.0001	11	8.4085	0.0001	45	8.4085
		500	500	0.0035	1	3.5560	0.0035	11	3.5560
	110	1000	1000	0.0011	3	3.5556	0.0011	22	3.5556
		2000	2000	0.0000	12	3.5556	0.0000	37	3.5556
		500	500	0.0010	1	16.5108	0.0010	13	16.5108
	90	1000	1000	0.0002	4	16.5124	0.0002	20	16.5124
		2000	2000	0.0000	8	16.5128	0.0000	37	16.5128
		500	500	0.0022	3	10.2077	0.0022	10	10.2077
0.30	100	1000	1000	0.0004	2	10.2093	0.0004	21	10.2093
		2000	2000	0.0001	10	10.2097	0.0001	46	10.2097
		500	500	0.0015	2	5.7299	0.0015	9	5.7299
	110	1000	1000	0.0004	3	5.7300	0.0004	20	5.7300
		2000	2000	0.0001	8	5.7301	0.0001	40	5.7301

Table B.8. Initial values for continuous fixed strike Asian call options when $r = 0.09$, $S_0 = 100$ and $T = 1$ using Gauss-Seidel and Successive-Over Relaxation methods

				Gauss-Seidel			Successive-Over Relaxation		
σ	K	M	N	E	t_A	C^a	E	t_A	C^a
		500	500	0.0040	6	8.8104	0.0039	6	8.8104
	95	1000	1000	0.0040	23	8.8086	0.0031	25	8.8086
		2000	2000	0.0080	91	8.8087	0.0036	108	8.8088
		500	500	0.0264	6	4.2390	0.0263	6	4.2390
0.05	100	1000	1000	0.0096	23	4.2938	0.0086	25	4.2938
		2000	2000	0.0099	91	4.3048	0.0051	108	4.3048
		500	500	0.1167	6	0.9795	0.1166	6	0.9795
	105	1000	1000	0.0297	23	0.9613	0.0285	25	0.9613
		2000	2000	0.0143	91	0.9590	0.0092	107	0.9590
		500	500	0.0042	6	13.3830	0.0031	7	13.3830
	90	1000	1000	0.0076	23	13.3843	0.0058	25	13.3846
		2000	2000	0.0352	101	13.3806	0.0263	104	13.3850
		500	500	0.0175	6	4.8961	0.0162	7	4.8961
0.10	100	1000	1000	0.0121	22	4.9103	0.0099	25	4.9105
		2000	2000	0.0452	101	4.9106	0.0350	107	4.9139
		500	500	0.0035	6	0.6472	0.0018	6	0.6472
	110	1000	1000	0.0137	23	0.6342	0.0113	25	0.6343
		2000	2000	0.0498	100	0.6306	0.0392	108	0.6313
		500	500	0.0097	7	13.8263	0.0077	6	13.8271
	90	1000	1000	0.0395	32	13.8266	0.0210	28	13.8274
		2000	2000	0.1828	188	13.8156	0.0682	156	13.8184
		500	500	0.0155	7	6.7724	0.0133	6	6.7732
0.20	100	1000	1000	0.0520	32	6.7725	0.0296	28	6.7733
		2000	2000	0.2357	188	6.7612	0.0919	156	6.7642
		500	500	0.0144	7	2.5478	0.0120	6	2.5483
	110	1000	1000	0.0597	32	2.5442	0.0358	29	2.5447
		2000	2000	0.2660	188	2.5362	0.1066	156	2.5381
		500	500	0.0193	9	14.9798	0.0090	7	14.9802
	90	1000	1000	0.0788	51	14.9753	0.0310	43	14.9765
		2000	2000	0.3374	330	14.9503	0.1024	262	14.9556
		500	500	0.0265	9	8.8248	0.0134	8	8.8252
0.30	100	1000	1000	0.1018	51	8.8194	0.0423	43	8.8206
		2000	2000	0.4287	330	8.7921	0.1360	262	8.7977
		500	500	0.0297	9	4.6952	0.0148	8	4.6955
	110	1000	1000	0.1173	51	4.6891	0.0509	42	4.6901
		2000	2000	0.4855	330	4.6662	0.1591	257	4.6708

Table B.9. Initial values for continuous fixed strike Asian call options when $r = 0.15$, $S_0 = 100$ and $T = 1$ using Gauss-Seidel and Successive-Over Relaxation methods

				Gauss-Seidel			Successive-Over Relaxation		
σ	K	M	N	E	t_A	C^a	E	t_A	C^a
		500	500	0.0018	6	11.0938	0.0020	6	11.0938
	95	1000	1000	0.0024	23	11.0941	0.0034	25	11.0941
		2000	2000	0.0093	91	11.0941	0.0042	108	11.0941
		500	500	0.0027	6	6.7872	0.0029	6	6.7872
0.05	100	1000	1000	0.0030	23	6.7920	0.0041	25	6.7920
		2000	2000	0.0102	91	6.7937	0.0047	107	6.7937
		500	500	0.0291	6	2.6685	0.0293	6	2.6685
	105	1000	1000	0.0095	23	2.7253	0.0106	25	2.7253
		2000	2000	0.0124	91	2.7398	0.0065	108	2.7398
		500	500	0.0034	6	15.3983	0.0022	6	15.3983
	90	1000	1000	0.0077	23	15.3982	0.0059	25	15.3986
		2000	2000	0.0347	101	15.3944	0.0263	105	15.3986
		500	500	0.0122	6	7.0103	0.0107	6	7.0103
0.10	100	1000	1000	0.0118	23	7.0232	0.0096	25	7.0235
		2000	2000	0.0458	101	7.0226	0.0359	107	7.0266
		500	500	0.0250	6	1.4218	0.0232	7	1.4218
	110	1000	1000	0.0178	23	1.4152	0.0153	25	1.4153
		2000	2000	0.0526	101	1.4126	0.0422	107	1.4110
		500	500	0.0097	6	15.6375	0.0078	6	15.6382
	90	1000	1000	0.0358	33	15.6372	0.0162	28	15.6380
		2000	2000	0.1492	189	15.6267	0.0648	157	15.6295
		500	500	0.0151	6	8.4029	0.0129	6	8.4037
0.20	100	1000	1000	0.0483	33	8.4035	0.0232	29	8.4044
		2000	2000	0.1979	189	8.3919	0.0891	157	8.3950
		500	500	0.0165	7	3.5553	0.0141	6	3.5558
	110	1000	1000	0.0566	32	3.5526	0.0283	29	3.5532
		2000	2000	0.2284	190	3.5435	0.1045	157	3.5457
		500	500	0.0162	9	16.5089	0.0079	8	16.5093
	90	1000	1000	0.0688	51	16.5046	0.0255	43	16.5058
		2000	2000	0.2876	333	16.4809	0.0868	265	16.4860
		500	500	0.0225	9	10.2055	0.0119	8	10.2059
0.30	100	1000	1000	0.0903	51	10.2003	0.0355	42	10.2016
		2000	2000	0.3720	333	10.1728	0.1177	265	10.1785
		500	500	0.0257	9	5.7279	0.0135	8	5.7282
	110	1000	1000	0.1047	52	5.7218	0.0429	43	5.7230
		2000	2000	0.4267	333	5.6973	0.1397	265	5.7023

Table B.10. Initial values for continuous fixed strike Asian call options when $r = 0.09$, $S_0 = 100$ and $T = 1$ using parallel implementation of Gauss-Seidel and Successive-Over Relaxation methods using four parallel processors

				Gauss-Seidel			Successive-Over Relaxation		
σ	K	M	N	E	t_A	C^a	E	t_A	C^a
		500	500	0.0033	5	8.8104	0.0035	9	8.8104
	95	1000	1000	0.0010	16	8.8086	0.0009	20	8.8086
		2000	2000	0.0014	45	8.8087	0.0036	56	8.8088
		500	500	0.0256	6	4.2390	0.0259	9	4.2390
0.05	100	1000	1000	0.0063	16	4.2938	0.0062	19	4.2938
		2000	2000	0.0026	44	4.3048	0.0051	57	4.3048
		500	500	0.1158	8	0.9795	0.1160	8	0.9795
	105	1000	1000	0.0259	16	0.9613	0.0258	21	0.9613
		2000	2000	0.0064	43	0.9590	0.0092	54	0.9590
		500	500	0.0026	7	13.3830	0.0032	8	13.3830
	90	1000	1000	0.0024	15	13.3843	0.0017	22	13.3846
		2000	2000	0.0142	51	13.3817	0.0086	57	13.3850
		500	500	0.0155	7	4.8961	0.0162	9	4.8961
0.10	100	1000	1000	0.0053	17	4.9103	0.0045	19	4.9105
		2000	2000	0.0186	50	4.9112	0.0118	58	4.9139
		500	500	0.0009	6	0.6472	0.0018	9	0.6472
	110	1000	1000	0.0057	15	0.6342	0.0048	18	0.6343
		2000	2000	0.0200	49	0.6307	0.0127	58	0.6313
		500	500	0.0051	8	13.8266	0.0038	8	13.8271
	90	1000	1000	0.0236	24	13.8279	0.0072	22	13.8286
		2000	2000	0.1099	88	13.8211	0.0223	86	13.8236
		500	500	0.0093	8	6.7726	0.0077	9	6.7731
0.20	100	1000	1000	0.0308	25	6.7736	0.0101	24	6.7744
		2000	2000	0.1414	93	6.7661	0.0295	85	6.7687
		500	500	0.0069	9	2.5479	0.0051	8	2.5483
	110	1000	1000	0.0352	26	2.5448	0.0119	25	2.5453
		2000	2000	0.1600	92	2.5389	0.0340	86	2.5406
		500	500	0.0126	10	14.9804	0.0049	10	14.9807
	90	1000	1000	0.0559	36	14.9780	0.0149	34	14.9790
		2000	2000	0.2919	158	14.9599	0.0340	143	14.9650
		500	500	0.0170	11	8.8255	0.0073	10	8.8258
0.30	100	1000	1000	0.0712	36	8.8222	0.0193	37	8.8234
		2000	2000	0.3674	161	8.8022	0.0437	144	8.8077
		500	500	0.0182	13	4.6957	0.0070	11	4.6960
	110	1000	1000	0.0813	37	4.6814	0.0224	35	4.6923
		2000	2000	0.4118	156	4.6742	0.0503	147	4.6787

Table B.11. Initial values for continuous fixed strike Asian call options when $r = 0.15$, $S_0 = 100$ and $T = 1$ using parallel implementation of Gauss-Seidel and Successive-Over Relaxation method using four parallel processors

				Gauss-Seidel			Successive-Over Relaxation		
σ	K	M	N	E	t_A	C^a	E	t_A	C^a
		500	500	0.0010	7	11.0938	0.0010	10	11.0938
	95	1000	1000	0.0006	17	11.0941	0.0005	19	11.0941
		2000	2000	0.0015	44	11.0941	0.0042	56	11.0941
		500	500	0.0019	6	6.7872	0.0019	8	6.7872
0.05	100	1000	1000	0.0010	17	6.7920	0.0009	21	6.7920
		2000	2000	0.0017	44	6.7937	0.0047	59	6.7937
		500	500	0.0281	7	2.6685	0.0281	7	2.6685
	105	1000	1000	0.0073	16	2.7253	0.0071	20	2.7253
		2000	2000	0.0033	43	2.7398	0.0065	58	2.7398
		500	500	0.0016	6	15.3983	0.0022	8	15.3983
	90	1000	1000	0.0023	16	15.3982	0.0016	19	15.3986
		2000	2000	0.0141	50	15.3956	0.0087	56	15.3986
		500	500	0.0099	8	7.0103	0.0107	8	7.0103
0.10	100	1000	1000	0.0047	16	7.0232	0.0039	18	7.0235
		2000	2000	0.0189	50	7.0234	0.0122	57	7.0266
		500	500	0.0221	6	1.4218	0.0232	8	1.4218
	110	1000	1000	0.0094	16	1.4152	0.0085	21	1.4153
		2000	2000	0.0222	49	1.4128	0.0148	54	1.4110
		500	500	0.0053	9	15.6378	0.0040	8	15.6382
	90	1000	1000	0.0197	22	15.6385	0.0073	22	15.6391
		2000	2000	0.0942	88	15.6320	0.0216	89	15.6344
		500	500	0.0089	8	8.4032	0.0074	7	8.4037
0.20	100	1000	1000	0.0263	23	8.4048	0.0101	24	8.4055
		2000	2000	0.1238	90	8.3971	0.0291	85	8.3998
		500	500	0.0090	9	3.5554	0.0072	10	3.5558
	110	1000	1000	0.0308	23	3.5534	0.0121	24	3.5539
		2000	2000	0.1417	87	3.5467	0.0338	86	3.5488
		500	500	0.0105	10	16.5095	0.0050	10	16.5098
	90	1000	1000	0.0506	36	16.5071	0.0120	37	16.5081
		2000	2000	0.2569	157	16.4896	0.0313	144	16.4946
		500	500	0.0145	11	10.2062	0.0073	12	10.2065
0.30	100	1000	1000	0.0654	39	10.2031	0.0159	34	10.2043
		2000	2000	0.3287	159	10.1827	0.0409	145	10.1884
		500	500	0.0159	10	5.7285	0.0075	9	5.7288
	110	1000	1000	0.0750	38	5.7243	0.0187	36	5.7253
		2000	2000	0.3731	158	5.7058	0.0471	148	5.7107

Table B.12. Initial values for continuous fixed strike Asian call options when $r = 0.09$, $S_0 = 100$ and $T = 1$ using Block Partition and Parallel Factorization algorithms using three parallel processors

				Block Partition			Parallel Factorization		
σ	K	M	N	E	t_A	C^a	E	t_A	C^a
		500	500	0.0033	2	8.8104	0.0033	3	8.8104
	95	1000	1000	0.0007	3	8.8086	0.0007	6	8.8086
		2000	2000	0.0002	9	8.8088	0.0002	12	8.8088
		500	500	0.0256	1	4.2390	0.0256	1	4.2390
0.05	100	1000	1000	0.0060	2	4.2938	0.0060	5	4.2938
		2000	2000	0.0013	7	4.3048	0.0013	8	4.3048
		500	500	0.1157	0	0.9795	0.1157	3	0.9795
	105	1000	1000	0.0255	2	0.9613	0.0255	5	0.9613
		2000	2000	0.0050	10	0.9590	0.0050	11	0.9590
		500	500	0.0023	1	13.3830	0.0023	3	13.3830
	90	1000	1000	0.0005	2	13.3846	0.0005	4	13.3846
		2000	2000	0.0001	9	13.3851	0.0001	8	13.3851
		500	500	0.0151	0	4.8961	0.0151	4	4.8961
0.10	100	1000	1000	0.0029	2	4.9105	0.0029	6	4.9105
		2000	2000	0.0004	7	4.9140	0.0004	13	4.9140
		500	500	0.0004	1	0.6472	0.0004	1	0.6472
	110	1000	1000	0.0029	3	0.6343	0.0029	5	0.6343
		2000	2000	0.0006	6	0.6313	0.0006	7	0.6313
		500	500	0.0015	2	13.8274	0.0015	3	13.8274
	90	1000	1000	0.0002	3	13.8305	0.0002	2	13.8305
		2000	2000	0.0000	7	13.8312	0.0000	7	13.8312
		500	500	0.0047	2	6.7734	0.0047	0	6.7734
0.20	100	1000	1000	0.0008	2	6.7764	0.0008	5	6.7764
		2000	2000	0.0001	7	6.7771	0.0001	14	6.7771
		500	500	0.0015	1	2.5484	0.0015	1	2.5484
	110	1000	1000	0.0010	3	2.5466	0.0010	4	2.5466
		2000	2000	0.0000	6	2.5463	0.0000	8	2.5463
		500	500	0.0010	0	14.9818	0.0010	2	14.9818
	90	1000	1000	0.0001	4	14.9834	0.0001	5	14.9834
		2000	2000	0.0001	6	14.9838	0.0001	6	14.9838
		500	500	0.0022	0	8.8270	0.0022	2	8.8270
0.30	100	1000	1000	0.0003	3	8.8284	0.0003	6	8.8284
		2000	2000	0.0000	6	8.8287	0.0000	12	8.8287
		500	500	0.0010	1	4.6971	0.0010	2	4.6971
	110	1000	1000	0.0004	2	4.6967	0.0004	5	4.6967
		2000	2000	0.0000	6	4.6967	0.0000	7	4.6967

Table B.13. Initial values for continuous fixed strike Asian call options when $r = 0.15$, $S_0 = 100$ and $T = 1$ using Block Partition and Parallel Factorization algorithms using three parallel processors

				Block Partition			Parallel Factorization		
σ	K	M	N	E	t_A	C^a	E	t_A	C^a
		500	500	0.0010	1	11.0938	0.0010	1	11.0938
	95	1000	1000	0.0002	3	11.0941	0.0002	6	11.0941
		2000	2000	0.0001	10	11.0941	0.0001	9	11.0941
		500	500	0.0018	1	6.7872	0.0018	1	6.7872
0.05	100	1000	1000	0.0005	4	6.7920	0.0005	4	6.7920
		2000	2000	0.0002	8	6.7937	0.0002	9	6.7937
		500	500	0.0280	1	2.6685	0.0280	3	2.6685
	105	1000	1000	0.0068	4	2.7253	0.0068	2	2.7253
		2000	2000	0.0016	6	2.7398	0.0016	11	2.7398
		500	500	0.0013	0	15.3983	0.0013	1	15.3983
	90	1000	1000	0.0003	3	15.3986	0.0003	4	15.3986
		2000	2000	0.0000	11	15.3987	0.0000	6	15.3987
		500	500	0.0095	2	7.0103	0.0095	1	7.0103
0.10	100	1000	1000	0.0022	2	7.0235	0.0022	3	7.0235
		2000	2000	0.0005	8	7.0267	0.0005	10	7.0267
		500	500	0.0216	1	1.4218	0.0216	1	1.4218
	110	1000	1000	0.0064	4	1.4153	0.0064	10	1.4153
		2000	2000	0.0010	7	1.4141	0.0010	14	1.4141
		500	500	0.0017	1	15.6385	0.0017	1	15.6385
	90	1000	1000	0.0003	2	15.6409	0.0003	3	15.6409
		2000	2000	0.0000	11	15.6416	0.0000	8	15.6416
		500	500	0.0043	0	8.4039	0.0043	1	8.4039
0.20	100	1000	1000	0.0008	4	8.4076	0.0008	5	8.4076
		2000	2000	0.0001	10	8.4085	0.0001	11	8.4085
		500	500	0.0035	0	3.5560	0.0035	3	3.5560
	110	1000	1000	0.0011	2	3.5556	0.0011	6	3.5556
		2000	2000	0.0000	6	3.5556	0.0000	10	3.5556
		500	500	0.0010	2	16.5108	0.0010	2	16.5108
	90	1000	1000	0.0002	2	16.5124	0.0002	4	16.5124
		2000	2000	0.0000	6	16.5128	0.0000	10	16.5128
		500	500	0.0022	1	10.2077	0.0022	3	10.2077
0.30	100	1000	1000	0.0004	3	10.2093	0.0004	5	10.2093
		2000	2000	0.0001	6	10.2097	0.0001	7	10.2097
		500	500	0.0015	3	5.7299	0.0015	1	5.7299
	110	1000	1000	0.0004	3	5.7300	0.0004	3	5.7300
		2000	2000	0.0001	7	5.7301	0.0001	17	5.7301

REFERENCES

Alziary B., J. P. Décamps and P. F. Koehl, 1997, "A P.D.E. Approach to Asian Options: Analytical and Numerical Evidence", *Journal of Banking and Finance*, Vol. 21, No. 5, pp. 613-640.

Ames, W. F., 1977, *Numerical Methods for Partial Differential Equations*, Academic Press, New York.

Amodio, P. and L. Brugnano, 1992, "Parallel Factorizations and Parallel Solvers for Tridiagonal Linear Systems", *Linear Algebra and Its Applications*, Vol. 172, pp. 347-364.

Baxter, M. W. and A. J. O. Rennie, 1996, *Financial Calculus: An Introduction to Derivative Pricing*, Cambridge University Press, Cambridge.

Bouaziz, L., E. Briys and M. Crouhy, 1994, "The Pricing of Forward Starting Asian Options", *Journal of Banking and Finance*, Vol. 18, No. 5, pp. 823-839.

Boyle, P., M. Broadie and P. Glasserman, 1997, "Monte Carlo Methods for Security Pricing", *Journal of Economic Dynamics and Control*, Vol. 21, No. 8, pp. 1267-1321.

Boyle, P. P. and D. Emanuel, 1980, "Options on the General Mean", *Working paper*, University of British Columbia.

Brennan, M. J. and E. S. Schwartz, 1978, "Finite Difference Method and Jump Processes Arising in the Pricing of Contingent Claims", *Journal of Financial and Quantitative Analysis*, Vol. 13, No. 3, pp. 461-474.

Bunnin, F. O., Y. Guo, Y. Ren and J. Darlington, 2000, "Design of High Performance Financial Modelling Environment", *Parallel Computing*, Vol. 26, pp. 601-622.

Courtadon, G., 1982, "A More Accurate Finite Difference Approximation for the Valuation of Options", *Journal of Financial and Quantitative Analysis*, Vol. 17, No. 5, pp. 301-330.

Crank, J. and P. Nicolson, 1947, "A Practical Method for Numerical Evaluation of Solutions of Partial Differential Equations of the Heat Conduction Type", *Proc. Camb Phil. Soc.*, Vol. 43, pp. 50-67.

Duffie, D., 1986, "Stochastic Equilibrium: Existence, Spanning Number and the No Expected Gains from Trade Hypothesis", *Econometrica*, Vol. 54, No. 5, pp. 1161-1184.

Dufresne, D., 2000, "Laguerre Series for Asian and Other Options", *Mathematical Finance*, Vol. 10, pp. 407-428.

Geman, H. and M. Yor, 1992, "Some Relations Between Bessel Processes, Asian Options and Confluent Hypergeometric Functions", *C.R. Acad. Sci., Paris*, Ser. I, Vol. 314, pp. 471-474.

Gürdağ, A. B., 2002, *A Parallel Implementation of a Link-based Ranking Algorithm for Web Search Engines*, M.S. Thesis, Boğaziçi University.

Harrison, J. M. and D. Kreps, 1979, "Martingales and Arbitrage in Multiperiod Securities Markets", J*ournal of Economic Theory*, Vol. 20, No. 3, pp. 380-408.

Harrison, J. M. and S. Pliska, 1981, "Martingales and Stochastic Integrals in the Theory of Continuous Trading", *Stochastic Processes and Their Applications*, Vol. 11, pp. 215-260.

Hull, J., 1992, *Options Futures and Other Derivative Securities*, 2nd ed., Prentice-Hall, New Jersey.

Hull, J. and A. White, 1990, "Valuing Derivative Securities Using the Explicit Finite Difference Method", *Journal of Financial and Quantitative Analysis*, Vol. 25, No. 1, pp. 87-100.

Ingersoll, J., 1987, *Theory of Financial Decision Making*, Rowman and Littlefield, New Jersey.

Kemna, A. G. Z. and A. C. F. Vorst, 1990, "A Pricing Method for Options Based on Average Asset Values", *Journal of Banking and Finance*, Vol. 14, No. 1, pp. 113-129.

Lamberton, D. and B. Lapeyre, 1991, "Introduction au Calcul Stochastique Appliqué à la Finance", *Collection Mathématiques et Applications* 9, *Ellipses.*

Lim, T. W., 2002, "Performance of Recursive Integration for Pricing European-style Asian options", *Revised for Journal of Computational Finance.*

Lin, H. X., 2001, "A Unifying Graph Model for Designing Parallel Algorithms for Tridiagonal Systems", *Parallel Computing*, Vol. 27, pp. 925-939.

Linetsky, V., 2001, "Exact Pricing of Asian Options: An Application of Spectral Theory", *Working paper*, Northwestern University.

Lyusternik, L. A., 1947, "A Note for Numerical Solution of Boundary Value Problems for the Laplace Equation and for the Calculation of Eigenvalues by the Method of Nets", *Trudy Inst. Math. Academy of Sciences of USSR*, Vol. 20, pp. 49-64 (Russian).

Mattor, N., T. J. Williams and D. W. Hewett, 1995, "Algorithm for Solving Tridiagonal Matrix Problems in Parallel", *Parallel Computing*, Vol. 21, pp. 1769-1782.

Milevsky, M. A. and S.E. Posner, 1998, "Asian Options, the Sum of Lognormals, and the Reciprocal Gamma Distribution", *Journal of Financial and Quantitative Analysis*, Vol. 33, No. 3, pp. 409-422.

Pacheco, P., 1997, *Parallel Programming with MPI*, Morgan Kaufmann Publishers Inc.

Rogers, L. C. G. and Z. Shi, 1995, "The Value of An Asian Option", *Journal of Applied Probability*, Vol. 32, pp. 1077-1088.

Schwartz, E. S., 1977, "The Valuation of Warrants: Implementing a New Approach", *Journal of Financial Economics*, Vol. 4, pp. 79-93.

Smith, G. D., 1978, *Numerical Solution of partial differential equations*, Oxford University Press, Oxford.

Stone, H. S., 1973, "An Efficient Parallel Algorithm for the Solution of Tridiagonal System of Linear Equations", *Journal of the Association for Computing Machinery*, Vol. 20, No. 1, 27-38.

Thomas, L. H., 1949, "Elliptic problems in Linear Difference Equations Over a Network", *Watson Sci. Comput. Lab. Rept.*, Columbia University, New York.

Turnbull, S. and L. Wakemann, 1992, "A Quick Algorithm for Pricing European Average Options", *Journal of Financial and Quantitative Analysis*, Vol. 26, No. 3, pp. 377-389.

Vecer, J., 2000, "A New PDE Approach for Pricing Arithmetic Asian Options", *Journal of Computational Finance*, forthcoming.

Vecer, J., 2001, "New Pricing of Asian Options", *Working paper*, Columbia University.

Wang, H. H., 1981, "A Parallel Method for Tridiagonal Equations", *ACM TOMS*, Vol. 7, No. 2, 170-183.

Yor, M., 1992, "On Some Exponential Functionals of Brownian Motion", *Advances in Applied Probability*, Vol. 24, pp. 509-531.

Young, D. M., 1954, "Iterative Methods for Solving Partial Differential Equations of

Eliptic Type", *Trans. Amer. Math. Soc.*, Vol. 76, pp. 92-111, 218, 242-272, 355, 394.

Zvan, R., P. Forsyth and K. Vetzal, 1997, "Robust Numerical Methods for PDE Models of Asian Options", J*ournal of Computational Finance*, Vol. 1, No. 2, 39-78.