Professor Ion SMEUREANU, PhD
Department of Economic Informatics
The Bucharest Academy of Economic Studies
Dumitru FANACHE, PhD
Department of mathematics
"Valahia" University, Targoviste

THE PARALLEL METHODS FOR EVALUATING A STOCHASTIC EQUATION

Abstract. *This paper is organised as follows: first, it is given a brief introduction of a simple model for European options price and next, the parallel algorithms applied to this model. Considering the idea given by odd-even cyclic reduction, we start from an explicit scheme obtained by means of finite differences, and give an alternative of evaluating the approximate values using an odd-even cyclic reduction which generates a logarithmic time of execution. Also, it is solved numerically using a parallelised domain decomposition method. The spatial domain is splitted among several processors, with data communicated among the processors using MPI. Interface conditions between domains are calculated using finite differences. For smaller sized problems the communication time is much longer than the computation time.*

Key words: *tridiagonal systems, odd-even cyclic reduction, forward Euler method.*

JEL Classification: C 200.

1. Introduction

In option pricing the Black-Scholes model is based on the geometric Brownian motion model of asset prices: $dS/S = \mu dt + \sigma dX$ where μ and σ are fixed values during the lifetime of the option. The result of the derivation is the Black-Scholes equation:

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 .S^2 \frac{\partial^2 V}{\partial S^2} - r.S.\frac{\partial V}{\partial S} - r.V = 0 \qquad (1)$$

with the constraints:

$$S \in [0,+\infty); \quad t \in [0,T]; \quad V(S,T) = g(S) \qquad (2)$$

where g, the terminal boundary condition, is the payoff at maturity of the option whose value will be given by V. We will assume that it does not pay dividends, and that both the risk-free rate(r) and the underlying volatility(σ) are constant during the life of the option (see [**4**]). Starting from analythical formulas (1) and (2) we can use parallel Monte Carlo method to find an acuratte solution (see [**8**], [**9**]) or finite difference method and we can transform (1) and (2) in a tridiagonal linear system.

This system can be solved efficiently using sequential computers. It is difficult to solve it efficiently using parallel computers, because the communication takes a significant part of the total execution time. This, together with the fast progress that parallel computing has known in the last decades, has increased the interest and the efforts towards the development of fast and efficient algorithms of solving such systems (see [**3**], [**4**], [**11**]). Using multigrid methods (see [**10**]) or Fast Fourier Transform (see [**6**], [**7**]) we obtain also a solution for a linear system in an optimal parallel execution time.

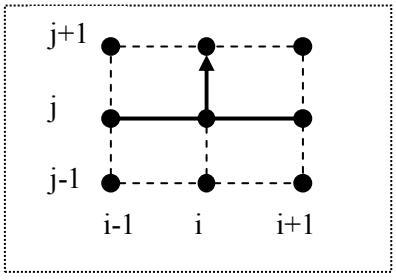
2. Parallel solution with odd-even cyclic method

The recursive doubling algorithm as developed by Stone (see [**5**]) can be used to solve a tridiagonal linear system of size N on a parallel computer with N processors using $O(\log N)$ parallel arithmetic steps.

Considering a mesh of equal S-steps of size δS and equal time-steps of size δt, with $(N+1)^2$ points and using central differences for S derivatives and backward differences for time derivatives, we get the explicit discretization of the Black-Scholes equation(see **[2])**:

$$\begin{aligned} b_0 V_0^j + c_0 V_1^j &= V_0^j \\ a_i V_{i-1}^j + b_i V_i^j + c_i V_{i+1}^j &= V_i^{j+1}, \quad i = 1,2,\cdots,N \end{aligned} \tag{3}$$

where *j* indicates the moment of time.



$$a_i = -\frac{1}{2}\left(\sigma^2 .i^2 - (r - S_0).i\right)\delta t;$$
$$b_i = 1 + \left(\sigma^2 .i^2 + r\right)\delta t;$$
$$c_i = -\frac{1}{2}\left(\sigma^2 .i^2 + (r - S_0).i\right)\delta t$$

Figure 1. Stencil of the explicit method for discretization Black Scholes equation

Relations (3) generate a system of equations of the following form:

$$Ax = d \tag{4}$$

where A is a (nonsymmetric) *tridiagonal matrix* of order $N+1$, x and d are vectors of dimension $N+1$

$$x = \left(x_0, x_1, \cdots, x_{N-1}, x_N\right)^T = \left(V_0^j, V_1^j, \cdots, V_{N-1}^j, V_N^j\right)$$
$$d = \left(d_0, d_1, \cdots, d_{N-1}, d_N\right)^T = \left(V_0^{j+1}, V_1^{j+1}, \cdots, V_{N-1}^{j+1}, V_N^{j+1}\right)$$

Detailing (4) we get

$$\begin{bmatrix} b_0 & c_0 & 0 & \cdots & 0 \\ a_1 & b_1 & c_1 & \cdots & 0 \\ 0 & a_2 & \ddots & \ddots & 0 \\ \vdots & \vdots & \ddots & \ddots & c_{N-1} \\ 0 & 0 & \cdots & a_N & b_N \end{bmatrix} \cdot \begin{bmatrix} V_0^j \\ V_1^j \\ \vdots \\ V_{N-1}^j \\ V_N^j \end{bmatrix} = \begin{bmatrix} V_0^{j+1} \\ V_1^{j+1} \\ \vdots \\ V_{N-1}^{j+1} \\ V_N^{j+1} \end{bmatrix} \tag{5}$$

which requires a time of execution of $O\left(N^2\right)$. Then we get the following:

Theorem. Using an odd-even cyclic reduction techinque, equation (3) can be computed in a $O\left(N\left[\log_2 j\right]\right)$ time.

Proof. Rewriting (3) for one single value *i*, and replacing V_i^j using the same connection among values, we get:

$$a_i V_{i-1}^j + b_i\left(a_i V_{i-1}^{j-1} + b_i V_i^{j-1} + c_i V_{i+1}^{j-1}\right) + c_i V_{i+1}^j = V_i^{j+1}$$

or, making some computations:

$$a_i\left(V_{i-1}^j + b_i V_{i-1}^{j-1}\right) + b_i^2 V_i^{j-1} + c_i\left(b_i V_{i+1}^{j-1} + V_{i+1}^j\right) = V_i^{j+1}$$

So, for *i* given, value V_i^{j+1} can be computed by using the values of two previous moments of time. Repeating the same substitution, we finally get:

$$a_i\left(\omega_j V_{i-1}^j + \omega_{j-1} V_{i-1}^{j-1} + \cdots + \omega_0 V_{i-1}^0\right) + \left(b_i\right)^j V_i^0 +$$
$$c_i\left(\omega_j^1 V_{i-1}^j + \omega_{j-1}^1 V_{i-1}^{j-1} + \cdots + \omega_0^1 V_{i-1}^0\right) = V_i^{j+1}$$

where the final coefficients are denoted by ω_i and ω_i^1, $i = \overline{0, j}$.

Using the double recursive technique (see **[3]**), in $\lceil \log_2 j \rceil$ parallel steps, the values in paranthesis are computed. Finally, for $i = 1,2,\cdots,N$, the total execution time is $O(N\lceil \log_2 j \rceil)$.□

On a parallel computer with $p < N$ processors, the version of the recursive doubling algorithm for solving a tridiagonal linear system (**4**) requires $O\left(\frac{N}{p} + \log p\right)$ parallel arithmetic steps (see [**3**]).

3. Parallel solution with forward Euler method

3.1. Deriving sequential algorithm. Analytic solutions to the PDE (**1**) depend upon the observation that it can be transformed into the diffusion equation in a function $u(x,\tau)$:

$$\frac{\partial u}{\partial \tau} = \frac{\partial^2 u}{\partial x^2} \tag{6}$$

with the constraints

$$x \in (-\infty,+\infty); \quad \tau \in [0,\tau_f]; \quad u(x,0) = f(x) \tag{7}$$

where f, the initial boundary condition, is g from the original formulation of the problem suitably transformed. Using a change of variables (8) that completely transforms (1) and (2) into (6) and (7):

$$x = \ln\left(\frac{S}{K}\right); \quad \tau = \frac{\sigma^2}{2}(T-t); \quad u(x,\tau) = \frac{V(S,t)}{K}e^{ax+b\tau} \tag{8}$$

where: $a = \frac{r - \frac{\sigma^2}{2}}{\sigma^2}$; $b = \frac{2r}{\sigma^2} + a^2$ and K is an arbitrary positive constant, usually chosen to be the strike price of the option. Once the problem has been transformed into (6) and (7), we use algebraic methods to solve it numerically.

This involves splitting the finite time interval $[0,\tau_f]$ into M equal subintervals of length $\Delta\tau$, resulting in a discretized time domain with $M+1$ nodes. We also split the spatial dimension into equal subintervals: N intervals of length Δx, giving rise to a spatial discretization with $N+1$ nodes.

Since the spatial domain is actually infinite, we must choose endpoints: x_L (**L**eft) and x_R (**R**ight) at which $V(S,t))$ is known to an acceptable degree of accuracy in advance. These values become *a priori* side boundary

conditions. The result is a rectangular domain of $(N+1)(M+1)$ nodes, of which the nodes at $\tau = 0$ and those at $x = x_L$ and $x = x_R$ have known u-values. The algorithm will step through this domain in τ, solving for the unknown u-values at the interior nodes at each time step, and after M iterations will produce the u-values at τ_f, corresponding to the time at which we wish to price the option(detail see in [**2**]). The terminal boundary condition for a call option is: $V(S,T) = \max(S-K,0)$, under the change of variables (8), this becomes the initial boundary condition:

$$u(x,0) = \max(e^x - 1,0)e^{ax} \tag{9}$$

We make the assumption that the S-value corresponding to x_L is sufficiently far out of the money ([**4**]) that the probability that it expires in the money is 0. Thus, the left-side boundary condition is simply:

$$u(x_L,\tau) = 0 \tag{10}$$

We make the assumption that the S-value corresponding to x_R is sufficiently far in the money that the probability that it expires out of the money is 0 ([**4**]). It is easy to see that a put under this assumption struck at the same K has value 0, and thus from put-call parity ([**4**]) the value of the call is simply that of a forward contract struck at K:

$$u(x_R,\tau) = \left(e^{x_R} - e^{\frac{-2r\tau}{\sigma^2}}\right)\left(e^{ax_R + b\tau}\right) \tag{11}$$

Using a change of variables (8) and with above observations (10, 11, 12) the economic model (1) and (2) is (6), (9),(10) and (11).

We will a fully discrete method(see Figure 1). So we have to substitute all derivation with finite differences.

$$\frac{u_i^{j+1} - u_i^j}{\Delta\tau} = \frac{u_{i+1}^j - 2u_i^j + u_{i-1}^j}{(\Delta x)^2} \quad i = 1,2,\cdots,N \tag{12}$$

We can rearrange this equation so that we get an equation for u_i^{j+1}

$$u_i^{j+1} = (1-2\alpha)u_i^j + \alpha\left(u_{i-1}^j + u_{i+1}^j\right) \quad i = 1,2,\cdots,N \tag{13}$$

The steps for finding the solution are outlined above; pseudocode is provided below to illustrate the method thoroughly. This pseudocode imagines that u-values are stored in a matrix U with $N+1$ rows, indexed from $i = 0,\cdots,N$, corresponding to the spatial nodes, and $M+1$ columns, indexed from $j = 0,\cdots,M$, corresponding to the time nodes. Under this arrangement, we

use the boundary conditions above to populate the column vector at $i = 0$ (initial boundary condition) and the row vectors at $j = 0$ (boundary condition at x_L) and $j = N$ (boundary condition at x_R).

This seems to be naturally clear, because the value of an option at the ith point at the next step of time will be affected by the value of option of his neighbors and his own value of option. From the initial conditions (10) we get the values for $u_i^0 = \max\left(e^x - 1, 0\right)e^{ax}$ for $i = 1,2,\cdots,N$. This is the starting point from which we can march the approximate solution forward step by step in time. The boundary conditions providing the necessary values (from (12) and (13)) $u_0^k = c_1$ ($u(x_L, \tau) = 0$) and $u_{N+1} = c_2$ ($u(x_R, \tau) = \left(e^{x_R} - e^{\frac{-2r\tau}{\sigma^2}} \right)\left(e^{ax_R + b\tau} \right)$).

This time stepping scheme is explicit because the approximate solution values at any given time step depend only on values that are available from the previous time step. The stencil is shown in **Figure 1**.

Note. The local truncation error of this scheme is $O(\Delta\tau) + O\left((\Delta x)^2\right)$, this means it is of first order in time and of second order in space. This scheme is simply Euler's method applied to semi-discrete system of ODEs derived for this problem using finite difference spatial discretization. The stability region of this method is definit by $\Delta\tau \le \frac{(\Delta x)^2}{2}$ or $\left(\alpha \le \frac{1}{2}\right)$ where $\alpha = \frac{\Delta\tau}{(\Delta x)^2}$ is Courant constant .

Finally, pseudocode for sequential implementation is:

Input data: $S_0, K, \sigma, r, T, M, \alpha, D, M$
Step 1 // *Computation domain and calculate relevant constant*
Step 2: *Boundary conditions*
// Declare matrix U's dimensions and populate its edges
// with boundary condition information
for *i*=0 **to** N **do** // initial boundary condition
$x_i = x_{left} + i.\Delta x$; $U(i,0) = max(\exp(x_i) - 1, 0).\exp(ax_i)$
end for
for *j*=1 **to** M **do** // side boundary condition
U(0,*j*)=0; $\tau_j = j\Delta\tau$
$U(N,j) = (\exp(x_{right}) - \exp(-2r\tau_j/\sigma^2))\exp(ax_{right} + b\tau_j)$
end for
Step 3: //*Forward Euler solution*
for j=1 **to** M **do**
for i=1 **to** N-1 **do**

$$u_i^{j+1} = (1-2\alpha)u_i^j + \alpha\left(u_{i-1}^j + u_{i+1}^j\right)$$

end for //i

end for //j

// Calculate V, the value of the option at time 0 and the current spot

Output data:V=K.U(N_{left},M)exp(-ax_0-$b\tau_f$)

So the time size depends on Δx, if Δx is small $\Delta\tau$ becomes very small. Therefore exists various implicit methods which would be unconditional stable. But we will use this explicit method, as we will also parallelize this.

3.2.Parallel Implementation. We implemented the method in C++ with MPI (see [**1**], [**4**]). The algorithm is relatively simple. We have only to calculate

$$u_i^{j+1} = (1-2\alpha)u_i^j + \alpha\left(u_{i-1}^j + u_{i+1}^j\right) \quad i = 1,2,\cdots,N \text{ for each node.}$$

At the beginning, the nodes are uniformly distributed to the available processors. So each processor is responsible for a disjunctive set of nodes. For each node the value for the next step of time is calculated using the values of its neighbors and there must be some communication between the different processors. We use Jacobi method to solve such types of systems of equations (13). A section of parallel program is given in **Figure 4** and the decomposition domain, calculation and changes of the new values among processes are shown in **Figure 2.** Each processor has two additional stores where the values of the neighbors will be saved. These values have to be updated after each step. At this equation is parabolic the solution converge to a stable state. So we have to include a convergence test. If the result doesn't change or it changes with only a small amount, the iteration will stop. In order to solve this we need some global communication.

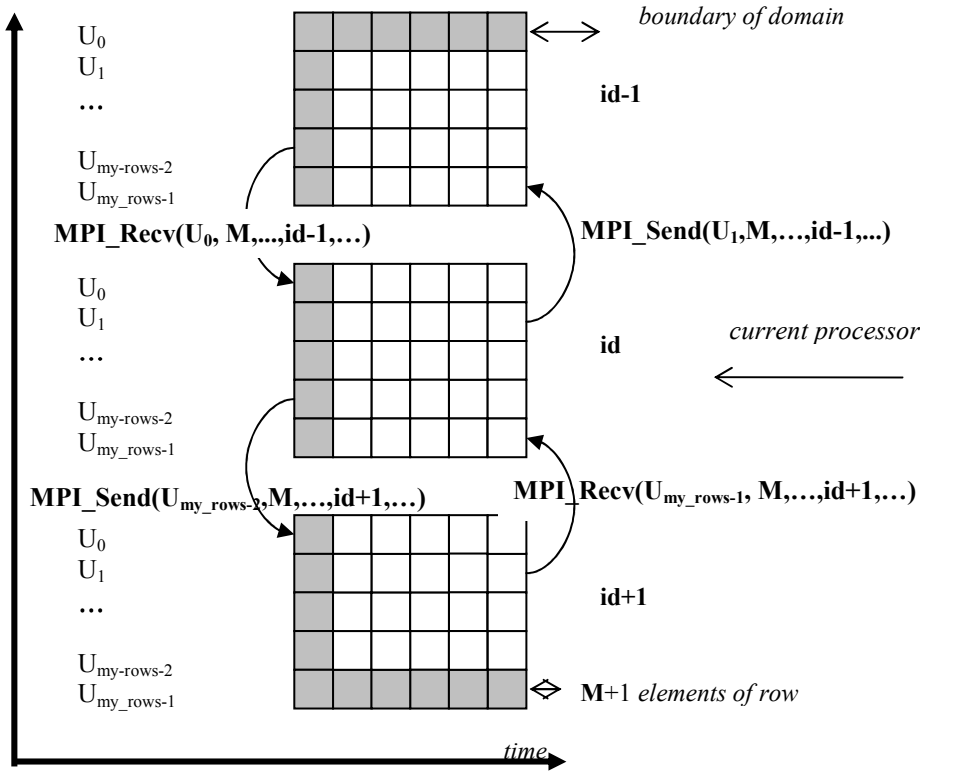


Figure 2. **Decomposition domain, calculating and changing the new values among processes**

Each processor send the norm of difference between the previous and the current solution to all others and sum all received norms. This will be done with `MPI_Allreduce` at each calculation step. At the end our calculation the results will gathered in to processor zero, and printed to `stdout`.

3.3. Results. The program was tested using a Windows cluster with 12× 1.73GHz processors. The execution time of a parallel program for different dimensions of discretization matrix is given in **Table 1**. First the program is run on one processor then on two, four, eight and finally on 12 processors. **Figure 3** shows that the speedup is nearly linear if the problem size is big. For smaller sized problems, the communication time is dominant in comparison with the computation time. That means that if there are too many processors involved, they spend more time to transfer the data to their neighbors than to calculate the next step. We also see that for a problem size of 5.10^4 working with eight processors would be optimal and 12 would be too much.

Processors Matrix size	**1**	**2**	**4**	**8**	**12**
initialization	0.0385	0.4270	1.0514	2.2763	4.25521
5.10^4	18.646	9.852	4.742	2.920	31.237
10^5	37.272	19.207	12.195	5.170	14.137
2.10^5	73.365	37.557	21.360	12.484	11.567
5.10^5	183.181	92.201	50.176	26.687	20.690
10^6	368.410	186.145	99.790	50.327	30.868

Table 1. **Execution time to a parallel program for different dimensions of discretization matrix**

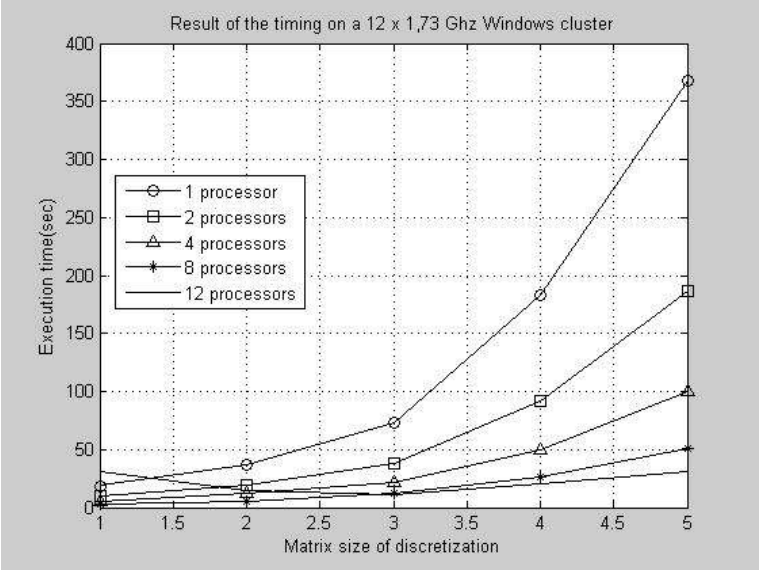


Figure 3. **Parallel execution time decrease once with increasing the number of processors**

```
int Jacobi_method(int p, int id, int my_rows,
             double **u, double **w,double alfa)
{double diff, global_diff;int iter=0,i,j;
 MPI_Status status;for(;;){
  if(id>0)
   MPI_Send(U[1],N,MPI_DOUBLE,id-
1,0,MPI_COMM_WORLD);
   if(id<p-1){ MPI_Send(U[my_rows-
2],M+1,MPI_DOUBLE,
                          id+1,0,MPI_COMM_WORLD);
               MPI_Recv(U[my_rows-
1],M+1,MPI_DOUBLE,

id+1,0,MPI_COMM_WORLD,&status);}
   if(id>0) MPI_Recv(U[0],M+1,MPI_DOUBLE,id-1,0,

MPI_COMM_WORLD,&status);
   diff=0.0; beta=1-2*alfa;
   for(i=1;i<my_rows-1;i++) for(j=1;j<=M;j++)
    {U_new[i][j]=alfa*U[i-1][j-1]+beta*U[i][j-1]
              +alfa*U[i+1][j-1];
     if(fabs(U_new[i][j]-U[i][j])>diff)
       diff=fabs(U_new[i][j]-U[i][j]);}
  if(diff<=EPSILON)break;
  for(i=1;i<=my_rows-1;i++)
   for(j=1;j<=M;j++) U[i][j]=U_new[i][j];
  MPI_Allreduce(&diff,&global_diff,1,MPI_DOUBLE,
                MPI_MAX, MPI_COMM_WORLD);
  if(global_diff<=EPSILON)break;
  iter++;}
return iter;}
```

Figure 4. **A section of parallel program what implemented forward Euler method for solving Black Scholes equation**

REFERENCES

[1] **Michael J. Quinn (2004),** ***Parallel Programming in C with MPI and OpenMP*****, McGraw-Hill;**

[2] **Ion Smeureanu, Dumitru Fanache (2008),** ***The Cyclic Odd-Even Reduction Method Applied in Mathematical Finance****,* Journal of Economics Informatics 3(47), ISSN 1842-8088 pp. 115-119;

[3]**Omer Egecioglu, Cetin K. Koc, Alan J. Laub (1988),** ***A Recursive Doubling Algorithm for Solution of Tridiagonal Systems on Hypercube Multiprocessors***, The Third Conference on Hypercube Concurrent Computers and Aplications, California Institute of Technology, Pasadena, California, January 19-20;

[4] **Koc M.B. Boztsun, I. Boztsun D**. **(2003),** ***On Numerical Solution of Black-Scholes Equation*****,** International Workshop on Mesh Free Methods;
[5] **Harold Stone (1975),** ***Parallel Tridiagonal Equation Solvers***, ACM Transactions on Mathematical Software,V.1 N.4 pg.289-307, Dec.;
[6] **Sajib Barua** (2004), ***Fast Fourier Transform for Option Pricing, Improved Mathematical Modeling and Design of an Efficient Parallel Algorithm***, Winnipeg, Manitoba, Canada;
[7] **A. Cerny** (2004), ***Introduction to Fast Fourier Transform in Finance***, Imperial College London, Tanaka Business School,)(a.cerny@imperial.ac.uk);
[8] **Bouchard**, **Bruno (2006),** ***Méthodes de Monte Carlo en Finance***, Notes de cours, Université Paris VI LPMA, et CREST, May;
[9] **Glasserman**, **P** (2003), ***Monte Carlo Methods in Financial Engineering*****,** Springer, Berlin;
[10] **Darae Jeong, Junseok Kim (2007),** ***An Accurate and Efficient Numerical Method for the Black*** **Scholes** ***Equation*****,** Department of Mathematics, Korea University;
[11] **Gerit van Wyk** (2007), *Numerical Methods for Valuing Financial Options*, Durham University, April.