The Cyclic Odd-Even Reduction Method Applied in Mathematical Finance

Ion SMEUREANU
Department of Economy Informatics, Academy of Economic Studies, Bucharest, România
Dumitru FANACHE
Mathematics Department, Valahia University Târgovişte, România

In this paper is to give a possibility of reducing execution time involved in evaluating a financial option by means finite difference scheme, using a cyclic odd-even reduction technique on a coarse-grained parallel model.
***Keywords**: coarse grained multicomputer, parallel algorithm, tridiagonal linear system, band matrices, odd-even cyclic reduction algorithm.*

1 Introduction

The general strategy for solving Black Scholes equation is to use a change of variables to transform into one that's more tractable, the diffusion equation(**[7][3])**. The terminal boundary condition becomes the initial boundary condition. Once the problem has been transformed we use algebraic methods to solve it numerically. The Crank-Nicolson finite difference equation may be used to calculate a discretised solution to diffusion equation. Importantly, initial and *both* boundary conditions must be provided for a solution to be calculable.

We seek the function $V(S,t)$ satisfying the PDE

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

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's volatility(σ) are constant over the life of the option[**7**].

Analytic solutions to the PDE depend upon the observation(see detail in **[4]**) that it can be transformed into the diffusion equation in a function $u(x,\tau)$:

$$\frac{\partial u}{\partial \tau}=\frac{\partial^2 u}{\partial x^2} \qquad (3)$$

with the constraints

$$x\in(-\infty,+\infty);\quad \tau\in\left[0,\tau_f\right];\quad u(x,0)=f(x) \qquad (4)$$

where f, the initial boundary condition, is g from the original formulation of the problem suitably transformed. Using a change of variables that completely transforms (**1**) and (**2**) into (**3**) and (**4**):

$$x=\ln\left(\frac{S}{K}\right);\quad \tau=\frac{\sigma^2}{2}(T-t);\quad u(x,\tau)=\frac{V(S,t)}{K}e^{ax+b\tau} \qquad (5)$$

where: $a=\dfrac{r-\dfrac{\sigma^2}{2}}{\sigma^2};\quad b=\dfrac{2r}{\sigma^2}+a^2$ (**6**)

and K is an arbitrary positive constant, usually chosen to be the strike price of the option. Once the problem has been transformed into (**3**) and (**4**), we use algebraic methods to solve it numerically. Using Crank Nicolson method in (**3**), we obtain:

$$Au(\tau)=b(\tau) \qquad (7)$$

In (**7**), matrix A is tridiagonal, symmetric, and strictly diagonally dominated, meaning that our equation will have a *unique solution*(**[1][4]**).

2. The CGM model and Odd-Even Cyclic Reduction method

Many current application in parallel machines are restricted to trivially parallelizable problem with low communication requirements. In real machines communication time

is usually much greater than computation time.

The **C**oarse **G**rained **M**ulticomputer (CGM) model to be an adequate model of parallelism sufficiently close to existing parallel machines. It is a simple model and nevertheless intends to give a reasonable prediction of performance when parallel algorithms on this model are implemented.

In the CGM model the effort to reduce communication is centered on reducing the number of communication rounds. Under this model, we design a communication efficient parallel algorithm for the solution of tridiagonal linear systems with *n* equation and *n* unknowns. This algorithms requires only a constant number of communication rounds. The amount of data transmitted in each communication round is proportional to the number of processors and independent of *n*. In addition to showing the theoretical complexity, we have implemented the proposed algorithm on a real distributed memory parallel machine. The experimental results obtained, indicate the efficiency and scalability of the algorithm.

The CGM model uses only two parameters, *n* and *p*, where *n* is the size of the input and *p* the number of processors each with $O(n/p)$ local memory. Each processor is connected by a router that can deliver messages in a point to point fashion. A **CGM** algorithm consists of an alternating sequence of computation round and communication rounds separated by barrier synchronizations.

In the computation round, we usually the best possible sequential algorithm in each processor to process its data locally. A communication round consists of a single *h*-relation with $h \le n/p$, that is, each processor exchanges at most a total of $O(n/p)$ data with other processors in one communication round. The proposed algorithm requires the transmission of only $O(p)$ data in each communication round.

In the CGM model the communication cost of a parallel algorithm is modeled by the number of communication rounds. The objective is to design algorithms that require a small amount of communications rounds. Many algorithms for graph an geometric problems [**2**] require only a constant or $O(\log p)$. Contrary to PRAM algorithms that frequently are designed for $p = O(n)$ and each processor receives a small number of input data, here we consider the more realistic case of $n>>p$. The CGM model is particularly suitable in nowdays parallel machines where the overall computation speed is considerably larger than the overall communication speed.

A tridiagonal system is one in which all elements, except possibly those on the main diagonal, and the ones just above or below it, are 0's. Instead of the usual notation a_{ij} for the element in row *i*, column *j,* of *A*, we use d_i to represent the main diagonal element a_{ii}, u_i for upper diagonal element $a_{i,i+1}$ and l_i for the lower diagonal element $a_{i,i-1}$. For the sake of uniformity, we define $l_0 = u_{n-1} = 0$. With the notation defined above, a tridiagonal system of linear equation can be written as follows, where $x_{-1} = x_n = 0$ are dummy variables that are introduced for uniformity.

$$\begin{aligned} l_0 x_{-1} + d_0 x_0 + u_0 x_1 &= b_0 \\ l_1 x_0 + d_1 x_1 + u_1 x_2 &= b_1 \\ l_2 x_1 + d_2 x_2 + u_3 x_3 &= b_2 \\ &\vdots \\ l_{n-1} x_{n-2} + d_{n-1} x_{n-1} + u_{n-1} x_n &= b_{n-1} \end{aligned} \qquad (8)$$

Odd-even cyclic reduction ([**2**],**[1])** is a recursive method for solving tridiagonal systems of size $n = 2^m - 1$. This method is divided into two parts: *reduction* and *back substitution*. The first step of reduction is to remove each odd-indexed x_i and create a tridiagonal system of size $2^{m-1} - 1$. We then do the same to this new system and continue on in the same manner until we are left with a system of size 1. Observe that the *i*th equation can be rewritten as

$$x_i = (1/d_i)(b_i - l_i x_{i-1} - u_i x_{i+1}) \qquad (9)$$

Taking the above equation (**9**) for each odd *i* and substituing into even-numbered equations (the ones with even indices for *l*, *d*, *u* and *b*), we obtain for each even $i(0 \le i \le n)$ an equation of the form:

$$-\frac{l_{i-1}l_i}{d_{i-1}}x_{i-2}+\left(d_i-\frac{l_i u_{i-1}}{d_{i-1}}-\frac{u_i l_{i+1}}{d_{i+1}}\right)x_i-\frac{u_i u_{i+1}}{d_{i+1}}x_{i+2}=b_i-\frac{l_i b_{i-1}}{d_{i-1}}-\frac{u_i b_{i+1}}{d_{i+1}} \qquad (\mathbf{10})$$

In this way, the *n* equation are reduced to $\lceil n/2 \rceil$ tridiagonal linear equation in the even-indexed variables. Applying the same method recursively, leads to $n/4$ equations, then $n/8$ equations, and, eventually, a single equation in x_0. Solving this last equation to obtain the value of x_0, and substituting backwards, allows us to compute the value of each of the *n* variables. Figure 1 shows the structure of the odd-even reduction method.

Forming each new equation requires six multiplications, six divisions, and four additions, but these can all be done in parallel using $p = n/2$ processors. Assuming unit-time arithmetic operations, we obtain the recurrence $T(n) = T(n/2) + 8 \approx 8\log_2 n$ for the total number of computational steps. The six division operations can be replaced with one reciprocation per new equation, to find $1/d_j$ for each odd *j*, plus six multiplications. Obviously, the above odd-even reduction method is applicable only if none of the d_j values obtained in the course of the computation is 0.

In the above analysis, interprocessor communication time was not taken into account. The analysis is thus valid only for the PRAM or for an architecture whose topology matches the communication structure shown in Figure 1. A binary *X-tree* architecture whose communication structure closely matches the needs of the above computation.

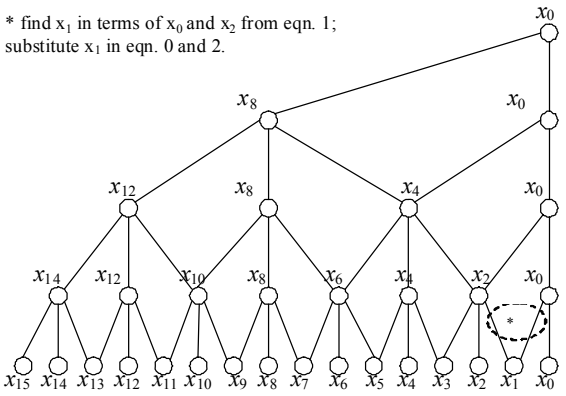

Fig.1. The structure of odd-even reduction for solving tridiagonal system of linear equation

To perform odd-even reduction on linear array of $p = n$ processors, we can assume that each processor initially holds one of the *n* equations. Direct one-step communication between neighboring processors leads to even-numbered processors obtaining the reduced set of $n/2$ equations as discussed above. The next reduction phase requires two-step communication, then four-step, and eventualy $(n/2)$step, leading to linear running time(of the same order as sequential time). On an *n*-processor 2*D* mesh, odd-even reduction can be easily organized to require $\Theta(\sqrt{p})$ time. It is worth noting that solving a tridiagonal system of linear equation can be converted to *a parallel prefix problem* as follows([**2**][**6**]).

We propose here a CGM algorithm(see Figure 4) that requires a constant number of communication rounds. By using the CGM paradigm, however, the algorithm proposed in this paper has been conceived indepen-

dently in a relatively natural way, following the CGM principles, namely, minimizing communication rounds and using as much local processing as possible. Furthermore, we have implemented this algorithm on a distributed memory parallel machine to verify its efficiency in practice.

Consider a distributed memory parallel computer of p processors $P_0, P_2, \cdots, P_{p-1}$ with $n >> p$. Assume that each processor has sufficient local memory to store $O(n/p)$ elements. (see Figure 2). We subdivide matrix A and the vector b into horizontal blocks or submatrices of n/p consecutive rows each. Each processor stores a submatrix of A and b.

The proposed algorithm makes use of a modified version of the odd-even reduction algorithm([**1**],[**2**],[**6**]). Each processor applies odd-even reduction to its n/p equations and eliminates all equation but the first and the last ones. Thus each processor will have only four unknowns. Each processor then send the two remaining equations to processor 0. Processor 0 applies odd-even reduction locally and solves for unknowns. Each processor receives the solved unknowns from processor 0 and solves for the remaining unknowns locally. The algorithm consists of the following *five phases* (see Figure 4) alternating between local processing and communication.

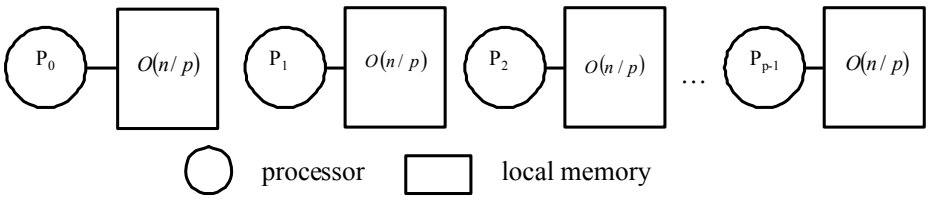

Fig.2. A parallel computer with distributed memory

Theorem 1. *A tridiagonal linear system with n equations and n unknowns can be solved on a CGM with p processors and* $O\left(\frac{n}{p}\right)$ *local memory per processor using* $O(1)$ *communication rounds with the transmission of* $O(p)$ *data per round.*

Proof: Step 1, 3 and 5 relate to local processing only. Step 2 and 4 require one communication round each. In the first communication round (step 2) each processor send a constant amount of data to processor 0, which in turn receives a total of $O(p)$ data. In the second communication round (step 4) processor 0 sends a total of $O(p)$ data and each of remaining processors receive a constant amount of data.

The sequential time was obtained with an optimized sequential algorithm run in a single processor (not the parallel algorithm run time one processor). For the experiment we use the following system: $l_i = u_i = -1$ for all $i = 1, \cdots, n-1$ and $d_i = 2$ for all $i = 0, \cdots, n-1$ (with the solution of all $x_i = 1$ for $i = 0,1, \cdots, n-1$).

We obtain an almost linear speedup for large *n*, regardless of the communication protocol utilized, as shown by the numerical results(see Figure 3 for the time curves).The times are given in units of clock ticks of the machine(1 clock tick =10^{-6} seconds).

3. Conclusions

Under the CGM model we have designed a communication-efficient parallel algorithm for the solution of tridiagonal linear systems. This algorithm requires only a constant number of communication round with of $O(p)$ data per round. In addition to showing its theoretical complexity, we have implemented this algorithm on a real distributed memory parallel machine. The experimental results show an almost linear speedup for large *n*. This is a very significant result since the particular machine we used presents a consider-

able *communication latency* and *low communication bandwidth*. It indicate the efficiency and scalability of the proposed algorithm

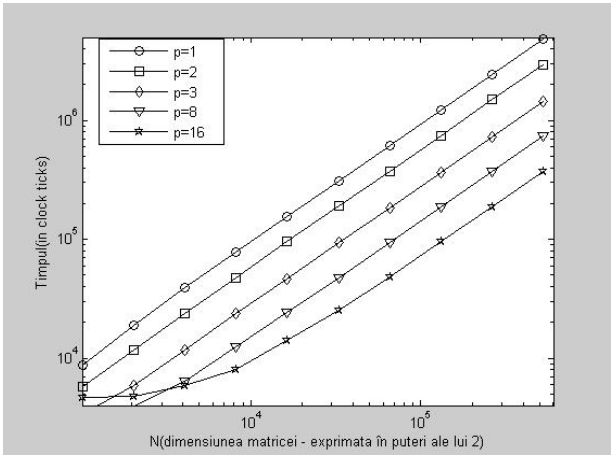

Fig.3. Total time of the tridiagonal algorithm in MPI

S1. Each processor applies the odd-even algorithm locally to eliminate all rows except the first and the last rows in its submatrix. With this, each processor P_i eliminates $n/p-2$ equations and $n/p-2$ unknowns, namely $x_{\frac{ni}{p}+2}, x_{\frac{ni}{p}+3}, \cdots, x_{\frac{n(i+1)}{p}-1}$.

S2. Each processor P_i sends its two remaining equation (with 4 unknowns $x_{\frac{ni}{p}}, x_{\frac{ni}{p}+1}, x_{\frac{n(i+1)}{p}}, x_{\frac{n(i+1)}{p}+1}$) to a same processor, say P_0. This processor thus obtains a system with $2p$ equations and $2p$ unknowns. Notice that this resulting system also consists of a tridiagonal matrix.

S3. Processor P_0 solves the system locally by od-even reduction or any other sequential method. It thus obtains the solution for the $2p$ unknowns.

S4. Processor P_0 sends to each processor P_i the computed value for 4 unknowns in the respective equations received in step 2.

S5. Each processor performs the inverse process of odd-even reduction used in step 1, by using the solution received for its two equations to solve the remaining equations.

Fig.4. Parallel algorithm odd-even reduction

References

[**1**]. **Eunice E. Santos**, *Optimal and Eficient Paralel Tridiagonal Solvers Using Direct Methods , The Journal of Supercomputing, 30, 97-115,2004 Kluwer Academic Publishers*, The Netherlands

[**2**]. **Behrooz Parhani**,*Introducing to Parallel Processing*(*Algorihms and Architectures* (electronic format), University of California at Santa Barbara, 2006

[**3**]. **Ion Smeureanu**, **Dumitru Fanache,** *A Linear Algorithm for Black Scholes Economic Model*, Revista de Informatică Economică , nr 1(45) / 2008, 150-156, ISSN 1842-8088

[**4**] **Dumitru Fanache,** *Parallelised Numeric Solutions for Implicit-Explicit Domain Decomposition,* 6th International Conference on Applied Mathematics, North University of Baia Mare ,2008, September 18-21

[5]. **C.M. Da Fonseca**, *On the Eigenvalues of Some Tridiagonal Matrices*, Departamento de Matemática, Universidade de Coimbra, 3001-454 Coimbra, Portugal

[**6**]. **R.Usmani**, *Inversion of Tridiagonal Jacobi matrix*, Linear Algebra Appl.212/213(1994) 413-414

[**7**] http://www.quantnotes.com/fundamentals/option/solvingbs.html