Available online at www.sciencedirect.com
ScienceDirect
Applied Mathematics and Computation 200 (2008) 178–188
APPLIED MATHEMATICS AND COMPUTATION
www.elsevier.com/locate/amc

Parallel algorithm for the solutions of PDEs in linux clustered workstations

Feras A. Mahmoud, Mohammad H. Al-Towaiq *

Jordan University of Science and Technology, Department of Mathematics and Statistics, P.O. Box 3030, Irbid 22110, Jordan

Abstract

In this paper we propose parallel algorithm for the solution of partial differential equations over a rectangular domain using the Crank–Nicholson method by cooperation with the DuFort–Frankel method and apply it on a model problem, namely, the heat conduction equation. One of the well known parallel techniques in solving partial differential equations in cluster computing environment is the domain decomposition technique. Using this technique, the whole domain is decomposed into subdomains, each of them has its own boundaries that are called the interface points. Parallelization is realized by approximating interface values using the unconditionally stable DuFort–Frankel explicit scheme, and these values serve as Neumann boundary conditions for the Crank–Nicholson implicit scheme in the subdomains. The numerical results show that our algorithm is more accurate than the algorithm based on the forward explicit method to approximate the values of the interface points, especially, when we use a small number of time steps. Moreover, these numerical results show that increasing the number of processors which are used in the cluster, yields an increase in the algorithm speedup.

Keywords: Heat conduction equation; Parallel computing; Domain decomposition; Crank–Nicholson; DuFort–Frankel

1. Introduction

The mathematical formulation of most problems in science and engineering involving rates of change with respect to two or more independent variables, usually representing time, length or angle, leads either to a partial differential equation (PDE) or to a set of such equations. Numerical approximation methods for solving PDEs those employing finite differences are more frequently used and more universally applicable than any other. These numerical methods often require a large number of computations, which make us explore parallel methods for solving PDEs.

Finite difference solutions for PDEs can be found either explicitly or implicitly. The explicit method is easy to implement on parallel computers but it has severe conditions for stability; that is, in order to attain reasonable accuracy, the space step must be small which forces necessarily the time step to be small too. The implicit

* Corresponding author.
E-mail addresses: feras_mahmoud@hotmail.com (F.A. Mahmoud), towaiq@just.edu.jo (M.H. Al-Towaiq).

0096-3003/$ - see front matter © 2007 Published by Elsevier Inc.
doi:10.1016/j.amc.2007.11.013

method does not have these conditions for stability but instead a global linear system of equations needs to be solved at each time step and it is not easy for parallel implementation.

Domain decomposition is a method widely used for solving time dependent PDEs and powerful tool for devising parallel PDE methods. A conventional approach [1] of parallelizing the implicit scheme is to apply the domain decomposition based on preconditioning methods to the problem arising from the semidiscretization at each time step. In [2] is proved that the preconditioning methods is well conditioned when the time step is small; nevertheless, small step size is not always describe in situations where implicit schemes become necessary to use. If the original domain is decomposed into a set of non-overlapping subdomains, then the PDEs defined in different subdomains could be solved on different processors concurrently. This often requires numerical boundary conditions at the interface points between subdomains. Since these interface points are not a part from the original model of the problem, we have to generate them numerically. One way to generate these numerical boundary conditions is to use the solutions from the previous time step to calculate the solution at the next time step. This is often referred to as *time lagging* [3]. A modified approximation scheme of mixed type was proposed by Kuznetsov [4] where the standard second order implicit scheme is used inside each subdomain, while the explicit Euler scheme is applied to obtain the interface values on the new time level. Once the interface values are available, the global problem is fully decoupled and can thus be computed in parallel. In [5] Dawson proposed a similar hybrid scheme, where instead of using the same spacing as for the interior points where the implicit scheme is applied, a larger spacing is used at each interface point where the explicit scheme is applied. In [1] Du, Mu, and Wu proposed two new parallel finite difference methods for parabolic PDEs and he focused on a one-dimensional heat equation in a spatial interval $[0, 1]$ as an example. For computation on the subdomain interface, Du used in the first method a high-order scheme, while he used a multistep explicit scheme for the other one. He studied the stability and error analysis of the two new schemes, and addressed the parallel efficiency of these schemes. In [6] Zhang and Wan presented some new techniques in designing finite difference domain decomposition algorithm for the heat equation. The basic idea is to define the finite difference schemes at the interface grid points with smaller time steps by Saulyev's asymmetric schemes.

In this paper, we propose parallel finite difference scheme for solving PDEs. For simplicity, we consider as a model the heat conduction equation

$$\frac{\partial u}{\partial t} = \tau^2 \frac{\partial^2 u}{\partial x^2}. \tag{1}$$

The parallel difference scheme based on both, the Crank–Nicholson (CN) implicit and DuFort–Frankel (DF) explicit schemes. In this procedure, the values of interface points of each subdomain are calculated by using the DF explicit scheme, and then these values serve as Neumann boundary conditions for the CN implicit scheme in the subdomains. The rest of the paper is organized as follows. In Section 2, we present a detailed description of the proposed algorithm. The stability of our parallel algorithm is given in Section 3. A numerical results and performance analysis are presented in Section 4. Finally, we conclude this paper in Section 5.

2. DF–CN parallel algorithm

Consider the heat conduction equation

$$\frac{\partial u}{\partial t} = \tau^2 \frac{\partial^2 u}{\partial x^2} \quad \text{for } 0 < x < \ell \quad \text{and} \quad 0 < t < T, \tag{2}$$

with initial condition

$$u(x, 0) = f(x) \quad \text{for } 0 \leqslant x \leqslant \ell,$$

and boundary conditions

$$\begin{aligned} u(0, t) &= c_1 \\ u(\ell, t) &= c_2 \end{aligned} \quad \text{for } 0 \leqslant t \leqslant T.$$

We partition the space into n subintervals and the time into m subintervals, and we define the space step $h = \frac{\ell}{n}$ and the time step $k = \frac{T}{m}$. Therefore, the domain is discretized uniformly. For designing the parallel algorithm we begin by choosing primitive tasks, identifying data communication patterns among them, and looking for ways to agglomerate tasks.

By using the domain decomposition technique, the $n+1$ elements of the space are divided among p processors fairly; that is, p divides the number of the space subintervals n. Denote the grid points by $u(x_i, t_j) = u(ih, jk) = u_{i,j}$ where $i = 0, 1, \ldots, n$ and $j = 1, 2, \ldots, m$. So that the interface points correspond to $i = (\frac{n}{p}), 2(\frac{n}{p}), \ldots, (p-1)(\frac{n}{p})$ and the boundary points correspond to $i = 0$ and $i = n$. Each processor is responsible to compute $(\frac{n}{p}) + 1$ points where each two neighbor processors share only one interface point at each time step, and each processor will compute this interface point concurrently.

Let

$$r = \tau^2 \frac{k}{h^2},$$

then at any time step $j = 1, 2, \ldots, m$, and if we use the approximation $w_{i,j}$ for $u_{i,j}$ in (2), the DF explicit scheme

$$(1+2r)w_{i,j+1} = 2r(w_{i+1,j} + w_{i-1,j}) + (1-2r)w_{i,j-1} \tag{3}$$

is applied at the interface points whereas the CN implicit scheme

$$(2+2r)w_{i,j+1} - r[w_{i-1,j+1} + w_{i+1,j+1}] = (2-2r)w_{i,j} + r[w_{i-1,j} + w_{i+1,j}]. \tag{4}$$

will be used to compute the interior points of each subdomain.

Fig. 1 depicts the communications needed to compute the solution at time $j+1$ given the solution at time j and time $j-1$. Processor q is responsible for computing $w_{i,j+1}$ implicitly using the CN difference scheme (4). It can compute the values of the gray cells (interior points) without any communications. However, it cannot compute the values of these gray cells until it computes the values of the black cells (interface points). Processor q can compute the values of the black cells explicitly, using the DF scheme (3), only if it gets values from neighboring processors. In Fig. 1b we show how processor q exchanges values with the neighboring processors, $q-1$ and $q+1$. After these values are received, the black cells can be computed. The parallel program allocates two extra points for processor q at each time step (the dotted cells). These points will receive the values received from the neighboring processors that will be stored in memory locations. These memory locations are called the *ghost points*. During the iteration that computes row $j+1$, each processor sends each of its neighbors the appropriate border values from row j and receives the neighbor's row j border value in turn. After the values has been received into the ghost points, every processor can compute all of its row $j+1$ interior values using the CN scheme (4).

3. Stability of the DF–CN algorithm

The DF–CN parallel algorithm is stable for all values of $r > 0$ if and only if both the DF explicit scheme and the CN implicit scheme are stable for all values of $r > 0$. When only one processor is used, the fully CN implicit scheme is applied and the algorithm is unconditionally stable [7]. Using two or more processors in the DF–CN algorithm leads us to approximate the values of the interface points by the DF explicit scheme, then

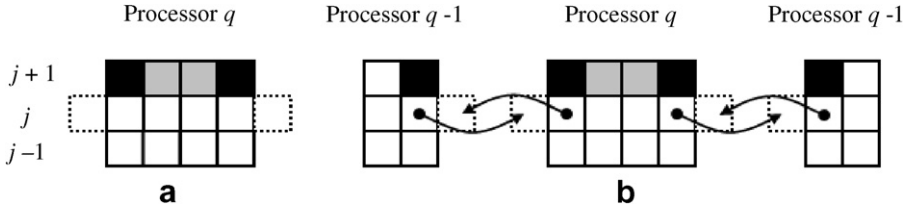

Fig. 1. Ghost points simplify parallel finite difference programs. (a) When computing row $j+1$, processor q has the data values it needs to fill in the gray cells, but it needs values from neighboring processors to fill in black cells. (b) Every processor sends its edge values to its neighbors. Every processor receives incoming values into ghost points.

these interface values serve as Neumann boundary conditions for the CN implicit scheme in the subdomains. So, we have to prove that the CN implicit scheme is unconditionally stable when boundary conditions of Neumann type are used for the heat conduction equation (2).

3.1. Neumann boundary conditions

Consider a thin rod of length ℓ that is thermally insulated along its length and which radiates heat from the end $x = 0$. Then the boundary condition at $x = 0$ is given by

$$-\frac{\partial u}{\partial x} = -\eta_1[u(0,t) - v_1].$$

A negative sign must be associated with $\frac{\partial u}{\partial x}$ because the outward normal to the rod at this end is in the *negative direction* of the x-axis. On the other hand, the boundary condition at $x = \ell$ is given by

$$\frac{\partial u}{\partial x} = -\eta_2[u(\ell,t) - v_2]$$

with a positive sign because the outward normal to the rod at this end is in the *same direction* of the x-axis. Note that η_1, η_2, v_1, v_2 are constants in which η_1 and η_2 are nonnegative. Hence, instead of the boundary conditions in (2), the boundary conditions have the form

$$\begin{matrix}\frac{\partial u(0,t)}{\partial x}\\ \frac{\partial u(\ell,t)}{\partial x}\end{matrix} \quad \text{for } 0 \leqslant t \leqslant T.$$

If we wish to represent $\frac{\partial u}{\partial x}$ more accurately at $x = 0$ and $x = \ell$ by the central difference formula

$$\frac{\partial u(x,t)}{\partial x} = \frac{u(x+h,t) - u(x-h,t)}{2h}, \tag{5}$$

it is necessary to introduce the *fictitious* temperature $w_{-1,j}$ and $w_{n+1,j}$ at the external mesh points $(-h, jk)$ and $(\ell + h, jk)$, respectively, by imagining the rod to be extended very slightly. Then, the boundary conditions can be represented by

$$w_{-1,j} = w_{1,j} - 2h\eta_1(w_{0,j} - v_1), \tag{6}$$

$$w_{n+1,j} = w_{n-1,j} - 2h\eta_2(w_{n,j} - v_2). \tag{7}$$

The temperatures $w_{-1,j}$ and $w_{n+1,j}$ are unknown and necessitates another two equations. Specifically, for the CN formula, at $i = 0$ and $i = n$, we have

$$(2 + 2r)w_{0,j+1} - r[w_{-1,j+1} + w_{1,j+1}] = (2 - 2r)w_{0,j} + r[w_{-1,j} + w_{1,j}], \tag{8}$$

and

$$(2 + 2r)w_{n,j+1} - r[w_{n-1,j+1} + w_{n+1,j+1}] = (2 - 2r)w_{n,j} + r[w_{n-1,j} + w_{n+1,j}]. \tag{9}$$

By substituting (6) and (7) into (8) and (9) respectively, the resulting formulae will be

$$[2 + 2r(1 + h\eta_1)]w_{0,j+1} - 2rw_{1,j+1} = [2 - 2r(1 + h\eta_1)]w_{0,j} + 2rw_{1,j} + 4rh\eta_1 v_1, \tag{10}$$

and

$$[2 + 2r(1 + h\eta_2)]w_{n,j+1} - 2rw_{n-1,j+1} = [2 - 2r(1 + h\eta_2)]w_{n,j} + 2rw_{n-1,j} + 4rh\eta_2 v_2. \tag{11}$$

For each time step we have to solve the $n + 1$ tridiagonal system of linear equations, using Thomas algorithm [8], which represented in matrix form as follows:

$$A\mathbf{w}^{(j+1)} = B\mathbf{w}^{(j)} + \mathbf{c}, \quad \text{for each } j = 0, 1, 2, \ldots, \tag{12}$$

where

$$\mathbf{w}^{(j)} = (w_{0,j}, w_{1,j}, \ldots, w_{n,j})^{\mathrm{t}},$$

and the matrices A and B and the vector $\mathbf{c}$ are given by

$$A = \begin{bmatrix} 2+2(1+h\eta_1)r & -2r & 0 & \cdots & \cdots & 0 \\ -r & 2+2r & -r & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & 0 \\ \vdots & \cdots & 0 & -r & 2+2r & -r \\ 0 & \cdots & \cdots & 0 & -2r & 2+2(1+h\eta_2)r \end{bmatrix},$$

$$B = \begin{bmatrix} 2-2(1+h\eta_1)r & 2r & 0 & \cdots & \cdots & 0 \\ r & 2-2r & r & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & 0 \\ \vdots & \cdots & 0 & r & 2-2r & r \\ 0 & \cdots & \cdots & 0 & 2r & 2-2(1+h\eta_2)r \end{bmatrix},$$

and

$$\mathbf{c} = (4hv_1\eta_1 r, 0, \ldots, 0, 4hv_2\eta_2 r)^{\mathrm{t}}.$$

As η_1, η_2, h and r are all nonnegative, then the matrix A in (12) is strictly diagonally dominant [9]. Since a diagonally dominant matrix is nonsingular [9], then there is a unique solution to the tridiagonal linear system (12) given by

$$\mathbf{w}^{(j+1)} = A^{-1}B\mathbf{w}^{(j)} + A^{-1}\mathbf{c}. \tag{13}$$

To examine stability of the CN difference scheme (12), let us assume that an error

$$\mathbf{e}^{(0)} = (e_0^{(0)}, e_1^{(0)}, \ldots, e_n^{(0)})^{\mathrm{t}}$$

is made in representing the initial data

$$\mathbf{w}^{(0)} = (w_{0,0}, w_{1,0}, \ldots, w_{n,0})^{\mathrm{t}}.$$

So, the initial vector is actually $\mathbf{w}^{(0)} + \mathbf{e}^{(0)}$, and so we have

$$\begin{aligned} \mathbf{w}^{(1)} &= A^{-1}B(\mathbf{w}^{(0)} + \mathbf{e}^{(0)}) + A^{-1}\mathbf{c} = A^{-1}B\mathbf{w}^{(0)} + A^{-1}\mathbf{c} + A^{-1}B\mathbf{e}^{(0)} \\ \mathbf{w}^{(2)} &= A^{-1}B\mathbf{w}^{(1)} + A^{-1}\mathbf{c} \\ &= (A^{-1}B)^2\mathbf{w}^{(0)} + (A^{-1}B)(A^{-1}\mathbf{c}) + A^{-1}\mathbf{c} + (A^{-1}B)^2\mathbf{e}^{(0)} \\ &\vdots \\ \mathbf{w}^{(k)} &= A^{-1}B\mathbf{w}^{(k-1)} + A^{-1}\mathbf{c} \\ &= (A^{-1}B)^k\mathbf{w}^{(0)} + \sum_{i=0}^{k-1}(A^{-1}B)^i(A^{-1}\mathbf{c}) + (A^{-1}B)^k\mathbf{e}^{(0)}. \end{aligned}$$

Hence, at the kth time step, the error in $\mathbf{w}^{(k)}$ due to $\mathbf{e}^{(0)}$ is $(A^{-1}B)^k\mathbf{e}^{(0)}$. In order for this error not to be magnified in the successive steps, we want

$$\left\|(A^{-1}B)^k\mathbf{e}^{(0)}\right\| \leqslant \|\mathbf{e}^{(0)}\|$$

for all values of k. Therefore, we must have

$$\|(A^{-1}B)^k\| \leqslant 1,$$

which requires that

$$\rho[(A^{-1}B)^k] = [\rho(A^{-1}B)]^k \leqslant 1,$$

where $\rho(A)$ is the *spectral radius* of the matrix A [9]. The CN difference scheme (12) is therefore stable only when the modulus of every eigenvalue of the matrix $A^{-1}B$ does not exceed one. Since the matrix B can be written as $B = 4I - A$, then

$$A^{-1}B = 4A^{-1} - I.$$

Therefore, the method is stable only when

$$\left|\frac{4}{\mu} - 1\right| \leqslant 1,$$

where μ is an eigenvalue of A. This is equivalent to $\mu \geqslant 2$. Since η_1, η_2, h and r are all nonnegative, then an application of Gerschgorin's circle theorem [9] to the matrix A in (12) shows that all its eigenvalues are at least 2 for any value of $r \geqslant 0$. Hence, the CN difference scheme (12) is unconditionally stable. Note that the local truncation error of the CN implicit scheme is $O(k^2 + h^2)$, see [9].

3.2. Stability analysis of the DF method

The stability of the DF scheme can be investigated by writing the DF formula (3) in the matrix form

$$\mathbf{w}^{(j+1)} = \frac{2r}{1+2r}A\mathbf{w}^{(j)} + \frac{1-2r}{1+2r}\mathbf{w}^{(j-1)} + \mathbf{c}, \tag{14}$$

where

$$\mathbf{w}^{(j)} = (w_{1,j}, w_{2,j}, \ldots, w_{n-1,j})^{\mathrm{t}}, \quad \mathbf{c} = (2rc_1, 0, \ldots, 0, 2rc_2)^{\mathrm{t}},$$

and

$$A = \begin{bmatrix} 0 & 1 & 0 & \cdots & \cdots & 0 \\ 1 & 0 & 1 & \ddots & & \vdots \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & \ddots & \ddots & 1 \\ 0 & \cdots & \cdots & 0 & 1 & 0 \end{bmatrix}.$$

Let

$$\mathbf{v}^{(j)} = \begin{bmatrix} \mathbf{w}^{(j)} \\ \mathbf{w}^{(j-1)} \end{bmatrix},$$

then Eq. (14) can be written as

$$\begin{bmatrix} \mathbf{w}^{(j+1)} \\ \mathbf{w}^{(j)} \end{bmatrix} = \begin{bmatrix} \frac{2r}{1+2r}A & \frac{1-2r}{1+2r}I \\ I & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{w}^{(j)} \\ \mathbf{w}^{(j-1)} \end{bmatrix} + \begin{bmatrix} \mathbf{c} \\ \mathbf{0} \end{bmatrix}$$

where I is the identity matrix of size $n-1$. Therefore,

$$\mathbf{v}^{(j+1)} = P\mathbf{v}^{(j)} + \mathbf{d}, \tag{15}$$

where

$$P = \begin{bmatrix} \frac{2r}{1+2r}A & \frac{1-2r}{1+2r}I \\ I & \mathbf{0} \end{bmatrix} \quad \text{and} \quad \mathbf{d} = \begin{bmatrix} \mathbf{c} \\ \mathbf{0} \end{bmatrix}.$$

This technique has reduced a three-level difference scheme to a two-level one. The DF scheme (15) will be unconditionally stable when each eigenvalue of p has a modulus less than or equal to 1. The following two theorems are useful for the analysis of the stability of three or more time level difference schemes and are easy to use. The proof of these theorems can be found in [7].

Theorem 1. *If the matrix A can be written as*

$$A = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & \cdots & A_{2n} \\ \vdots & \vdots & & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{nn} \end{bmatrix},$$

where each A_{ij} is an $m \times m$ matrix, and all the A_{ij} has a common set of n linearly independent eigenvectors, then the eigenvalues of A are given by the eigenvalues of the matrices

$$\begin{bmatrix} \lambda_{11}^{(k)} & \lambda_{12}^{(k)} & \cdots & \lambda_{1n}^{(k)} \\ \lambda_{21}^{(k)} & \lambda_{22}^{(k)} & \cdots & \lambda_{2n}^{(k)} \\ \vdots & \vdots & & \vdots \\ \lambda_{n1}^{(k)} & \lambda_{nn}^{(k)} & \cdots & \lambda_{nn}^{(k)} \end{bmatrix}; \quad k = 1, 2, \ldots, m,$$

where $\lambda_{ij}^{(k)}$ is the kth eigenvalue of A_{ij} corresponding to the kth eigenvector $\mathbf{g}_k$ common to all the A_{ij}'s.

Theorem 2. *The eigenvalues of the $n \times n$ tridiagonal matrix A, where*

$$A = \begin{bmatrix} a & b & 0 & \cdots & \cdots & 0 \\ c & a & b & \ddots & & \vdots \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & c & a & b \\ 0 & \cdots & \cdots & 0 & c & a \end{bmatrix},$$

are

$$\lambda_k = a + 2\sqrt{bc}\cos\left(\frac{k\pi}{n+1}\right); \quad k = 1, 2, \ldots, n.$$

The matrix A from (15) is a tridiagonal matrix. So, by Theorem (2), the matrix A has $n-1$ different eigenvalues which are

$$\lambda_k = 2\cos\left(\frac{k\pi}{n}\right); \quad k = 1, 2, \ldots, n-1,$$

and thus it has $n-1$ linearly independent eigenvectors $\mathbf{g}_i$; $i = 1, 2, \ldots, n-1$. Although the matrix I has $n-1$ eigenvalues each equal to 1, then it has $n-1$ linearly independent eigenvectors which may be taken as $\mathbf{g}_i$; $i = 1, 2, \ldots, n-1$. Hence, by Theorem (1), the eigenvalues μ of P are the eigenvalues of

$$\begin{bmatrix} \frac{2r}{1+2r}\lambda_k & \frac{1-2r}{1+2r} \\ 1 & 0 \end{bmatrix}$$

where λ_k is the kth eigenvalue of A. The values of μ can be computed by evaluating

$$\det\begin{bmatrix} \frac{2r}{1+2r}\lambda_k - \mu & \frac{1-2r}{1+2r} \\ 1 & -\mu \end{bmatrix} = 0,$$

which gives

$$\mu^2 - \frac{2r}{1+2r}\lambda_k\mu - \frac{1-2r}{1+2r} = 0.$$

Therefore,

$$\mu = \frac{2r\cos\left(\frac{k\pi}{n}\right) \pm \sqrt{1-4r^2\sin^2\left(\frac{k\pi}{n}\right)}}{1+2r},$$

and there are two cases: if $0 \leqslant 1-4r^2\sin^2(\frac{k\pi}{n}) < 1$, then

$$|\mu| < \frac{2r+1}{1+2r} = 1,$$

and if $1-4r^2\sin^2(\frac{k\pi}{n}) < 0$, then

$$\begin{aligned} |\mu|^2 &= \frac{\left(2r\cos\left(\frac{k\pi}{n}\right)\right)^2 + 4r^2\sin^2\left(\frac{k\pi}{n}\right) - 1}{(1+2r)^2} \\ &= \frac{4r^2-1}{4r^2+4r+1} < 1. \end{aligned}$$

Therefore the DF explicit difference scheme (3) is unconditionally stable for all values of $r > 0$.

The local truncation error of this scheme is $O(k^2 + h^2 + \frac{k^2}{h^2})$, see [9]. Successive refinement of the values of h and k may generate a finite difference solution that is stable, but that may converge to the solution of a different PDE. For example, in the DF explicit difference scheme, as both h and k tend to zero at the same rate, the ratio $\frac{k}{h}$ is constant, and thus we solve a modified PDE and not the original Eq. (2). However, the DF scheme is consistent if k tends to zero faster than h.

4. Numerical results and performance analysis

In this section, we consider a heat conduction equation. We use the DF–CN algorithm to approximate the solution of this equation. The example is implemented using the academic cluster built in the department of Computer Science at Jordan University of Science and Technology. This cluster contains 1 management node and 18 Linux (Kernel 2.4.20.8 RedHat 9) workstations connected as a star network, each of which has a single IBM Pentium IV with 2.4 GHz, 512 Cache, 512 MBs of memory and 40 GBs disk space. These hosts are connected together by fast Ethernet, 1 GB switch and 1 optical interconnection switch. We use the Message Passing Interface (MPI) with the `MPICH` version 1.5.2 as a message passing library throughout the implementations. The barrier synchronization and blocking point-to-point communication are used. The graphs reported in the figures represent the average speedup and efficiency over *many* runs of the DF–CN algorithm.

Example. Consider the heat conduction equation

$$\frac{\partial u}{\partial t} = \frac{4}{\pi^2}\frac{\partial^2 u}{\partial x^2} \quad \text{for } 0 < x < 4 \quad \text{and} \quad t > 0,$$

with boundary conditions

$$u(0,t) = u(4,t) = 0; \quad t > 0,$$

and initial condition

$$u(x,0) = \sin\left(\frac{\pi}{4}x\right)\left[1 + 2\cos\left(\frac{\pi}{4}x\right)\right]; \quad 0 \leqslant x \leqslant 4.$$

The *exact* solution to this problem is

$$u(x,t) = e^{-t}\sin\left(\frac{\pi}{2}x\right) + e^{\frac{-t}{4}}\sin\left(\frac{\pi}{4}x\right).$$

The solutions at $t = 0.01$ will be approximated using the proposed DF–CN algorithm with several values of h when $k = 1 \times 10^{-6}$. Figs. 2–4 show the execution time, speedup and efficiency of the DF–CN parallel difference scheme corresponding to several values of n when $1, 2, 4, 8, 12, 16$ and 18 processors are used.

It is clear, from Fig. 3, that the speedup of the DF–CN algorithm is not ideal, i.e, it is not linear with the number of processors. This because of the decreasing of the problem size when the number of processors increases, which makes the communication time to be the dominant in comparison with the computation one. Also, the height speed of the processors used in implementing our algorithm effects on the parallel execution time; that is, the small problem sizes will take little execution time to perform a certain calculations. However, in the DF–CN algorithm, as problem size n increases, so does the height of the speedup curve. Also, for a fixed number of processors, speedup is an increasing function of the problem size.

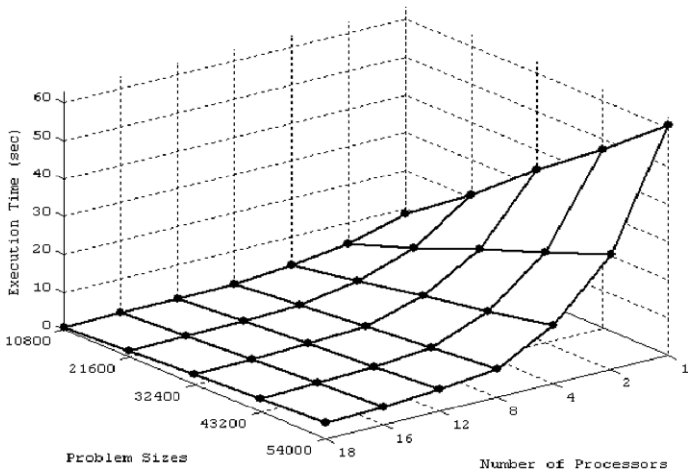

Fig. 2. The execution time of the DF–CN algorithm using several values of h with $k = 1 \times 10^{-6}$ at $t = 0.01$.

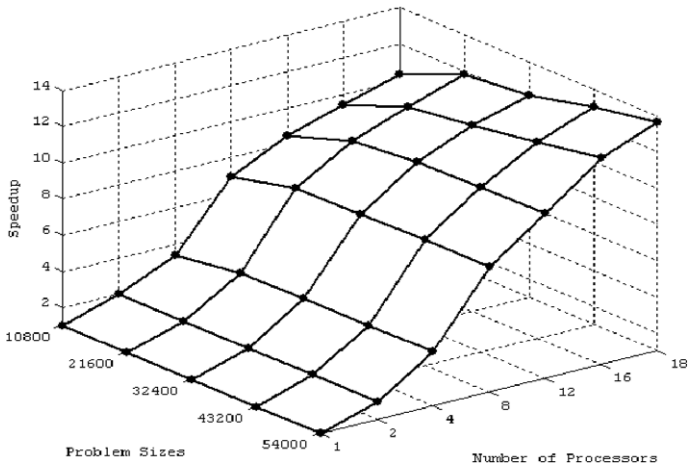

Fig. 3. Speedup of the DF–CN parallel algorithm using several values of h with $k = 1 \times 10^{-6}$ at $t = 0.01$.

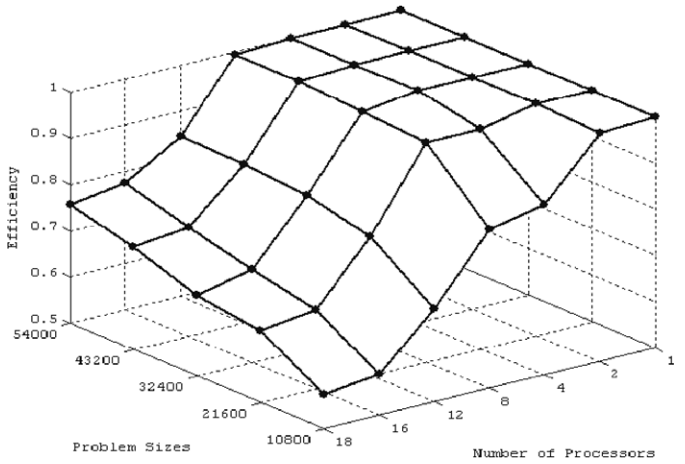

Fig. 4. Efficiency of the DF–CN parallel algorithm using several values of h with $k = 1 \times 10^{-6}$ at $t = 0.01$.

For a problem of fixed size, the efficiency of a parallel computation typically decreases as the number of processors increases, see Fig. 4. Since parallel communication increases when the number of processors increases, the way to maintain efficiency when increasing the number of processors is to increase the size of the problem being solved. The proposed algorithm assumes that the data structures we manipulate fit in primary memory. The maximum problem size we can solve is limited by the amount of primary memory that is available. Also, in the DF–CN parallel algorithm, as problem size n increases, the height of the efficiency curve increases.

The DF–CN parallel algorithm is accurate. For example, when $n = 54,000$, the value of r is approximately 73.863 and the maximum error is 5.035×10^{-9}, while using this choice of n if the *forward difference* scheme, see [7,9], is used to approximate the values of the interface points instead of the DF scheme makes the approximate solution diverges.

To analyze the performance, let χ represent the time needed to compute an interior point using the Thomas algorithm. Using a single processor to update the $n + 1$ points requires time $(n + 1)\chi$. Because the algorithm has m time steps, the total expected execution time of the sequential algorithm is

$$t^s = m(n + 1)\chi. \tag{16}$$

To compute the parallel execution time using p processors, suppose that each of them is responsible for an equal-sized portion contains $(\frac{n}{p}) + 1$ points, two boundaries and $(\frac{n}{p}) - 1$ interiors, in general. The boundary points will be computed by the DF explicit scheme, while the interior points will be computed using the Thomas algorithm. Suppose ω represent the time needed to compute a boundary point, the parallel computation time for each iteration is

$$t^p_{\text{comp}} = \left[\left(\frac{n}{p}\right) - 1\right]\chi + 2\omega.$$

However, the parallel algorithm involves communication that the sequential algorithm does not. In general, each processor must send values to its two neighboring processors and receive two values from them. If ζ represents the time needed for a processor to send (receive) a value to (from) another processor, the necessary communications increase the parallel execution time for each iteration 2ζ. Therefore,

$$t^p_{\text{comm}} = 2\zeta.$$

Combining computation time with communication time, the overall parallel execution time for all m iterations of the algorithm is

$$t^p = m\{t^p_{\text{comp}} + t^p_{\text{comm}}\} = m\left\{\left[\left(\frac{n}{p}\right) - 1\right]\chi + 2\omega + 2\zeta\right\}. \tag{17}$$

The speedup relative to the sequential algorithm is

$$S = \frac{t^s}{t^p} = \frac{(n+1)\chi}{\left[\left(\frac{n}{p}\right) - 1\right]\chi + 2\omega + 2\zeta}, \tag{18}$$

and the parallel efficiency is given by

$$E = \frac{S}{p} = \frac{(n+1)\chi}{[n-p]\chi + 2p\omega + 2p\zeta}. \tag{19}$$

5. Conclusion

In this paper, the DF–CN parallel algorithm has been discussed in depth. This algorithm uses the DF explicit scheme to approximate the solution at the interior boundaries between subdomains. For the remaining points in each subdomain, the algorithm uses the CN implicit scheme. This scheme has no stability constraint and prevents the algorithm from moving to worse approximations.

A numerical example is given for the proposed DF–CN parallel algorithm. From the numerical results we conclude that the DF–CN algorithm is recommended when small values of time steps are used. Small number of time steps will decrease the inter-processors communications; which decreases the communication time of this parallel algorithm. This algorithm gave more accurate results than the parallel algorithm that uses the forward difference scheme to approximate the values of the interface points especially when a small number of time steps are used.

Furthermore, in the DF–CN parallel algorithm, as problem size n increases, so does the height of the speedup curve. Also, for a fixed number of processors, speedup is an increasing function of the problem size. Moreover, the efficiency of the DF–CN algorithm computation typically decreases as the number of processors increases. The way to maintain efficiency when increasing the number of processors is to increase the size of the problem being solved. Unfortunately, the maximum problem size we can solve is limited by the amount of primary memory that is available.

References

[1] Q. Du, M. Mu, Z.N. Wu, Efficient parallel algorithms for parabolic problems, SIAM J. Numer. Anal. 30 (2001) 1469–1487.
[2] X. Cai, Additive Schwarz algorithms for parabolic convection–diffusion equations, Numer. Math. 50 (1991) 41–52.
[3] Rivera-Gallego Wilson, Stability analysis of numerical boundary conditions in domain decomposition algorithms, Appl. Math. Comput. 137 (2003) 375–385.
[4] Y.A. Kuznetsov, New algorithms for approximate realization of implicit difference scheme, Soviet J. Numer. Anal. Math. Model. 3 (1988) 99–114.
[5] C.N. Dawson, Q. Du, T.F. Dupnot, A finite difference domain decomposition algorithm for numerical solution of the heat equation, Math. Comput. 57 (1991) 63–71.
[6] Zhang Bao-Lin, Wan Zheng-Su, New techniques in designing finite-difference domain decomposition algorithm for the heat equation, Comput. Math. Appl. 45 (2003) 1695–1705.
[7] G.D. Smith, Numerical Solution of Partial Differential Equations: Finite Difference Methods, Oxford University Press, London, 1978.
[8] Gearge Em Karniadakis, Robert M. Kirby II, Parallel Scientific Computing in C++ and MPI, Cambridge University Press, Cambridge, 2003.
[9] Richard L. Burden, J. Douglas Faires, Numerical Analysis, seventh ed., Brooks/Cole, United States of America, 2001.