An Efficient Parallel ADI Algorithm for Solving 3-D Convection Diffusion Equations with Neumann Boundary Conditions

Honghai Zeng
Engineering Research Center
Mississippi State University
Mississippi State, MS 39762, USA
hzeng@erc.msstate.edu

Jianping Zhu
Department of Theoretical and Applied Mathematics
University of Akron
Akron, OH 44325, USA
jzhu@math.uakron.edu

Abstract

Convection diffusion equations are widely used to model various important phenomena and processes in science and engineering. The calculation of numerical solutions for three-dimensional models is very computation intensive. The Alternate Direction Implicit (ADI) algorithm is very efficient for this kind of equations and suitable for parallel computing. However, when Neumann boundary conditions are involved in the equations, it is difficult to maintain the original order of accuracy. We will discuss a new method in this paper to deal with Neumann boundary conditions when the ADI algorithm is used. The new method maintains the second order accuracy and is very scalable on multiprocessor parallel computers.

1. Introduction

Many important phenomena and processes in science and engineering, such as transport and retention of chemical contaminants in groundwater flowing through soils [6, 7], are described by the following convection diffusion equation

$$\frac{\partial C}{\partial t} + d\frac{\partial C}{\partial x} + e\frac{\partial C}{\partial y} + f\frac{\partial C}{\partial z} = a\frac{\partial^2 C}{\partial x^2} + b\frac{\partial^2 C}{\partial y^2} + c\frac{\partial^2 C}{\partial z^2} + F(x,y,z,t), \quad (1)$$

where we have assumed, for simplicity, that a, b, c, d, e, f are constants, and the computational domain is normalized into a cube with $(x, y, z) \in [0,1] \times [0,1] \times [0,1]$, which is discretized using a uniform rectangular grid (x_i, y_j, z_k), $i = 1, 2, ..., NI+1$, $j = 1, 2, ..., NJ+1$, and $k = 1, 2, ..., NK+1$. The time dimension is also discretized using a uniform grid with a step size of Δt, and the discrete solution values are expressed as:

$$C^n_{i,j,k} = C((i-1)\Delta x,\ (j-1)\Delta y,\ (k-1)\Delta z,\ n\Delta t), \quad (2)$$

where $\Delta x = \frac{1}{NI}$, $\Delta y = \frac{1}{NJ}$, and $\Delta z = \frac{1}{NK}$.

Discretizing Eq. (1) using the Crank-Nicolson algorithm [1], we have:

$$\begin{aligned}
&\frac{C^{n+1}_{i,j,k} - C^n_{i,j,k}}{\Delta t} + d\delta_x(\frac{C^{n+1}_{i,j,k} + C^n_{i,j,k}}{2}) \\
&+e\delta_y(\frac{C^{n+1}_{i,j,k} + C^n_{i,j,k}}{2}) + f\delta_z(\frac{C^{n+1}_{i,j,k} + C^n_{i,j,k}}{2}) \\
&= a\delta^2_x(\frac{C^{n+1}_{i,j,k} + C^n_{i,j,k}}{2}) + b\delta^2_y(\frac{C^{n+1}_{i,j,k} + C^n_{i,j,k}}{2}) \\
&+c\delta^2_z(\frac{C^{n+1}_{i,j,k} + C^n_{i,j,k}}{2}) + \frac{F^{n+1}_{i,j,k} + F^n_{i,j,k}}{2} \qquad (3)
\end{aligned}$$

where the difference operators $\delta_x, \delta_y, \delta_z, \delta^2_x, \delta^2_y$, and δ^2_z are defined as

$$\begin{aligned}
\delta_x C^n_{i,j,k} &= \frac{C^n_{i+1,j,k} - C^n_{i-1,j,k}}{2\Delta x}, \\
\delta_y C^n_{i,j,k} &= \frac{C^n_{i,j+1,k} - C^n_{i,j-1,k}}{2\Delta y}, \\
\delta_z C^n_{i,j,k} &= \frac{C^n_{i,j,k+1} - C^n_{i,j,k-1}}{2\Delta z}, \\
\delta^2_x C^n_{i,j,k} &= \frac{C^n_{i+1,j,k} - 2C^n_{i,j,k} + C^n_{i-1,j,k}}{\Delta x^2}, \\
\delta^2_y C^n_{i,j,k} &= \frac{C^n_{i,j+1,k} - 2C^n_{i,j,k} + C^n_{i,j-1,k}}{\Delta y^2}, \\
\delta^2_z C^n_{i,j,k} &= \frac{C^n_{i,j,k+1} - 2C^n_{i,j,k} + C^n_{i,j,k-1}}{\Delta z^2}.
\end{aligned}$$

Grouping the unknown terms at time step $n+1$ to the left and the known terms at time step n to the right, we have:

$$\begin{aligned}
&(I + \frac{\Delta t d}{2}\delta_x - \frac{\Delta t a}{2}\delta^2_x + \frac{\Delta t e}{2}\delta_y - \frac{\Delta t b}{2}\delta^2_y \\
&+\frac{\Delta t f}{2}\delta_z - \frac{\Delta t c}{2}\delta^2_z)C^{n+1}_{i,j,k} = \\
&(I - \frac{\Delta t d}{2}\delta_x + \frac{\Delta t a}{2}\delta^2_x - \frac{\Delta t e}{2}\delta_y + \frac{\Delta t b}{2}\delta^2_y
\end{aligned}$$

Proceedings of the International Conference on Parallel Processing Workshops (ICPPW'02)
1530-2016/02 $17.00 © 2002 IEEE

$$-\frac{\Delta tf}{2}\delta_z + \frac{\Delta tc}{2}\delta_z^2)C_{i,j,k}^n + \frac{F_{i,j,k}^{n+1} + F_{i,j,k}^n}{2}\Delta t \quad (4)$$

This algorithm is unconditionally stable with an accuracy of order $O(\Delta t^2, \Delta x^2, \Delta y^2, \Delta z^2)$. The system of algebraic equations generated by this discretization, i.e.

$$\mathbf{A}\mathbf{c}^{n+1} = \mathbf{H}\mathbf{c}^n + \frac{\Delta t}{2}(\mathbf{f}^{n+1} + \mathbf{f}^n) \quad (5)$$

has the following coefficient matrix in a nested block form:

$$\mathbf{A} = \begin{bmatrix} \mathbf{B} & \mathbf{C} & & & \\ \mathbf{C} & \mathbf{B} & \mathbf{C} & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & \mathbf{C} \\ & & & \mathbf{C} & \mathbf{B} \end{bmatrix},$$

where

$$\mathbf{B} = \begin{bmatrix} \mathbf{D} & \mathbf{E} & & & \\ \mathbf{E} & \mathbf{D} & \mathbf{E} & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & \mathbf{E} \\ & & & \mathbf{E} & \mathbf{D} \end{bmatrix},$$

$$\mathbf{C} = \begin{bmatrix} \times & & & & \\ & \times & & & \\ & & \ddots & & \\ & & & \ddots & \\ & & & & \times \end{bmatrix},$$

$$\mathbf{D} = \begin{bmatrix} \times & \times & & & \\ \times & \times & \times & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & \times \\ & & & \times & \times \end{bmatrix},$$

and

$$\mathbf{E} = \begin{bmatrix} \times & & & & \\ & \times & & & \\ & & \ddots & & \\ & & & \ddots & \\ & & & & \times \end{bmatrix}.$$

Direct calculation of solutions from Eq. (5) is usually too time-consuming for large systems, even with the latest supercomputers. Assuming that $NI = NJ = NK = N$, it can be shown that the computational complexity for solving all equations in (5) is of order $O(N^7)$. One way to reduce the computational complexity is to use the fraction time step algorithm, in which only the spatial operators in one direction is treated implicitly at each fraction step:

$$\begin{aligned} &\frac{C_{i,j,k}^{n+\frac{1}{3}} - C_{i,j,k}^n}{\frac{1}{3}\Delta t} = \\ &a\frac{C_{i-1,j,k}^{n+\frac{1}{3}} - 2C_{i,j,k}^{n+\frac{1}{3}} + C_{i+1,j,k}^{n+\frac{1}{3}}}{\Delta x^2} - d\frac{C_{i+1,j,k}^{n+\frac{1}{3}} - C_{i-1,j,k}^{n+\frac{1}{3}}}{2\Delta x} \\ &+b\frac{C_{i,j-1,k}^n - 2C_{i,j,k}^n + C_{i,j+1,k}^n}{\Delta y^2} - e\frac{C_{i,j+1,k}^n - C_{i,j-1,k}^n}{2\Delta y} \\ &+c\frac{C_{i,j,k-1}^n - 2C_{i,j,k}^n + C_{i,j,k+1}^n}{\Delta z^2} - f\frac{C_{i,j,k+1}^n - C_{i,j,k-1}^n}{2\Delta z} \\ &+\frac{1}{2}(Q_{i,j,k}^{n+\frac{1}{3}} + Q_{i,j,k}^n) \end{aligned} \quad (6)$$

$$\begin{aligned} &\frac{C_{i,j,k}^{n+\frac{2}{3}} - C_{i,j,k}^{n+\frac{1}{3}}}{\frac{1}{3}\Delta t} = \\ &a\frac{C_{i-1,j,k}^{n+\frac{1}{3}} - 2C_{i,j,k}^{n+\frac{1}{3}} + C_{i+1,j,k}^{n+\frac{1}{3}}}{\Delta x^2} - d\frac{C_{i+1,j,k}^{n+\frac{1}{3}} - C_{i-1,j,k}^{n+\frac{1}{3}}}{2\Delta x} \\ &+b\frac{C_{i,j-1,k}^{n+\frac{2}{3}} - 2C_{i,j,k}^{n+\frac{2}{3}} + C_{i,j+1,k}^{n+\frac{2}{3}}}{\Delta y^2} - e\frac{C_{i,j+1,k}^{n+\frac{2}{3}} - C_{i,j-1,k}^{n+\frac{2}{3}}}{2\Delta y} \\ &+c\frac{C_{i,j,k-1}^{n+\frac{1}{3}} - 2C_{i,j,k}^{n+\frac{1}{3}} + C_{i,j,k+1}^{n+\frac{1}{3}}}{\Delta z^2} - f\frac{C_{i,j,k+1}^{n+\frac{1}{3}} - C_{i,j,k-1}^{n+\frac{1}{3}}}{2\Delta z} \\ &+\frac{1}{2}(Q_{i,j,k}^{n+\frac{2}{3}} + Q_{i,j,k}^{n+\frac{1}{3}}) \end{aligned} \quad (7)$$

$$\begin{aligned} &\frac{C_{i,j,k}^{n+1} - C_{i,j,k}^{n+\frac{2}{3}}}{\frac{1}{3}\Delta t} = \\ &a\frac{C_{i-1,j,k}^{n+\frac{2}{3}} - 2C_{i,j,k}^{n+\frac{2}{3}} + C_{i+1,j,k}^{n+\frac{2}{3}}}{\Delta x^2} - d\frac{C_{i+1,j,k}^{n+\frac{2}{3}} - C_{i-1,j,k}^{n+\frac{2}{3}}}{2\Delta x} \\ &+b\frac{C_{i,j-1,k}^{n+\frac{2}{3}} - 2C_{i,j,k}^{n+\frac{2}{3}} + C_{i,j+1,k}^{n+\frac{2}{3}}}{\Delta y^2} - e\frac{C_{i,j+1,k}^{n+\frac{2}{3}} - C_{i,j-1,k}^{n+\frac{2}{3}}}{2\Delta y} \\ &+c\frac{C_{i,j,k-1}^{n+1} - 2C_{i,j,k}^{n+1} + C_{i,j,k+1}^{n+1}}{\Delta z^2} - f\frac{C_{i,j,k+1}^{n+1} - C_{i,j,k-1}^{n+1}}{2\Delta z} \\ &+\frac{1}{2}(Q_{i,j,k}^{n+1} + Q_{i,j,k}^{n+\frac{2}{3}}) \end{aligned} \quad (8)$$

This method reduces a three-dimensional problem into a sequence of one-dimensional problems that only require solving systems of equations with tri-diagonal coefficient matrices, thereby greatly decreasing the computational complexity. However, it is only conditionally stable and first-order accurate in time [4].

Another method to improve the efficiency and accuracy of the calculations is the Alternating Direction Implicit (ADI) method, which was first introduced by Peaceman and

Rachford [5] for solving parabolic PDEs. It can also be derived from the Crank-Nicolson algorithm (4) [4, 5]. Note that Eq. (4) can be approximately factorized as:

$$(I + \frac{\Delta td}{2}\delta_x - \frac{\Delta ta}{2}\delta_x^2)(I + \frac{\Delta te}{2}\delta_y - \frac{\Delta tb}{2}\delta_y^2)$$
$$(I + \frac{\Delta tf}{2}\delta_z - \frac{\Delta tc}{2}\delta_z^2)C_{i,j,k}^{n+1} =$$
$$(I - \frac{\Delta td}{2}\delta_x + \frac{\Delta ta}{2}\delta_x^2)(I - \frac{\Delta te}{2}\delta_y + \frac{\Delta tb}{2}\delta_y^2)$$
$$(I - \frac{\Delta tf}{2}\delta_z + \frac{\Delta tc}{2}\delta_z^2)C_{i,j,k}^{n} + \frac{F_{i,j,k}^{n+1} + F_{i,j,k}^{n}}{2}\Delta t \quad (9)$$

The difference between Eq. (9) and Eq. (4), or the additional error due to the approximate factorization, is of order $O(\Delta t^3)$ if the necessary partial derivatives of C are bounded. This additional error is of the same order as the truncation error in the original algorithm (4). Therefore, the following three-step ADI method still maintains second-order accuracy in both time and space:

$$(I + \frac{\Delta td}{2}\delta_x - \frac{\Delta ta}{2}\delta_x^2)C_{i,j,k}^{*}$$
$$= (I - \tfrac{\Delta td}{2}\delta_x + \tfrac{\Delta ta}{2}\delta_x^2)(I - \tfrac{\Delta te}{2}\delta_y + \tfrac{\Delta tb}{2}\delta_y^2)$$
$$(I - \tfrac{\Delta tf}{2}\delta_z + \tfrac{\Delta tc}{2}\delta_z^2)C_{i,j,k}^{n} + \tfrac{F_{i,j,k}^{n+1}+F_{i,j,k}^{n}}{2}\Delta t, \quad (10)$$

$$(I + \frac{\Delta te}{2}\delta_y - \frac{\Delta tb}{2}\delta_y^2)C_{i,j,k}^{**} = C_{i,j,k}^{*}, \quad (11)$$

$$(I + \frac{\Delta tf}{2}\delta_z - \frac{\Delta tc}{2}\delta_z^2)C_{i,j,k}^{n+1} = C_{i,j,k}^{**}. \quad (12)$$

The solutions to the three sets of equations in (10), (11), and (12) can be computed by solving systems of algebraic equations with tri-diagonal coefficient matrices, since the left hand sides of the equations involve only three-point central difference operators δ_x^2, δ_x, δ_y^2, δ_y, δ_z^2, and δ_z. Although the right hand side of the equation (10) involves the product of these operators, it does not complicate the solution process since it is applied to the known solution values from the previous time step. Assuming that $NI = NJ = NK = N$, it can be shown that the computational complexity for solving all equations in (10) - (12) is of order $O(N^3)$, which is significantly lower than $O(N^7)$ of the direct solution method applied to Eq. (5).

2. Boundary conditions for the ADI method

Two types of boundary conditions are used in most modeling and simulation problems using convection diffusion equations: 1) the Dirichlet boundary condition that specifies function values of the solution to be calculated, and 2) the Neumann boundary condition that specifies the derivatives of the solution to be calculated. For example, when studying the transport and retention of contaminants in soil with groundwater flow, the Dirichlet boundary conditions are used for the boundary where a solution of known concentration is applied (for example corresponding to a leak in a tank holding the solution), while the Neumann boundary conditions are used for the boundary where the flow rate is known (for example corresponding to a no-flow or impermeable boundary). Sometimes the boundary conditions may also be given in terms of a combination of both function values and derivatives at the same spatial point.

For the fraction time step algorithm given by (6) - (8), since the solution values calculated at all three fraction time steps are clearly defined at $n + \frac{1}{3}$, $n + \frac{2}{3}$, and $n + 1$, it is straightforward to deal with boundary conditions of either type as long as they are given for all fraction time steps.

For the ADI algorithm based on approximate factorization given by (10) - (12), it is more complicated to deal with boundary conditions. Since the solutions $C_{i,j,k}^{*}$ and $C_{i,j,k}^{**}$, to be calculated in (10) and (11), respectively, are intermediate variables without clear connection to the physical time level, we can not use the given boundary conditions in a straightforward way. One way is to apply the known boundary conditions at the time levels $n + \frac{1}{3}$ and $n + \frac{2}{3}$ to $C_{i,j,k}^{*}$ in (10) and $C_{i,j,k}^{**}$ in (11), respectively. However, this will degrade the accuracy of the algorithm to only first order accurate in time, as will be demonstrated later in the section of Numerical experiment.

For the Dirichlet boundary conditions, there is another method that can maintain second order accuracy [4, 5]. When solving Eq. (10), we need boundary conditions $C_{1,j,k}^{*}$ and $C_{NI+1,j,k}^{*}$, $j = 2, ..., NJ$ and $k = 2, ..., NK$. Note that since the operators in (10) - (12) commute, we can combine Eqs. (11) and (12) as

$$C_{i,j,k}^{*} = (I + \frac{\Delta te}{2}\delta_y - \frac{\Delta tb}{2}\delta_y^2)(I + \frac{\Delta tf}{2}\delta_z - \frac{\Delta tc}{2}\delta_z^2)C_{i,j,k}^{n+1} \quad (13)$$

The boundary conditions for $C_{1,j,k}^{*}$ and $C_{NI+1,j,k}^{*}$ can then be calculated from the known values of $C_{i,j,k}^{n+1}$ on the boundary by setting $i = 1$ and $i = NI + 1$ in Eq. (13), respectively:

$$C_{1,j,k}^{*} = (I + \frac{\Delta te}{2}\delta_y - \frac{\Delta tb}{2}\delta_y^2)$$
$$(I + \frac{\Delta tf}{2}\delta_z - \frac{\Delta tc}{2}\delta_z^2)C_{1,j,k}^{n+1} \quad (14)$$

$$C_{NI+1,j,k}^{*} = (I + \frac{\Delta te}{2}\delta_y - \frac{\Delta tb}{2}\delta_y^2)$$
$$(I + \frac{\Delta tf}{2}\delta_z - \frac{\Delta tc}{2}\delta_z^2)C_{NI+1,j,k}^{n+1} \quad (15)$$

Similarly, when solving Eq. (11), we need boundary conditions for $C_{i,1,k}^{**}$ and $C_{i,NJ+1,k}^{**}$, $i = 2, ..., NI$ and $k = 2, ..., NK$. This can be calculated reversely from Eq. (12)

by setting $j = 1$ and $j = NJ + 1$, respectively:

$$C^{**}_{i,1,k} = (I + \frac{\Delta tf}{2}\delta_z - \frac{\Delta tc}{2}\delta_z^2)C^{n+1}_{i,1,k} \quad (16)$$

$$C^{**}_{i,NJ+1,k} = (I + \frac{\Delta tf}{2}\delta_z - \frac{\Delta tc}{2}\delta_z^2)C^{n+1}_{i,NJ+1,k} \quad (17)$$

For Eq. (12), the boundary conditions can be used in a straightforward way since the solutions to be calculated are defined at the time step $n + 1$.

The above approach, however, does not apply to the Neumann-type boundary conditions. A new method is developed here to utilize the given derivative boundary conditions to solve Eq. (10) and (11). Assume the Neumann boundary conditions are given on the surfaces of the computational domain, i. e. $i = 1, i = NI+1, j = 1, j = NJ+1$, $k = 1$, and $k = NK + 1$. To construct second order approximation for the Neumann boundary conditions, we need to create an addition layer of "ghost-points" on each side of the boundary. Therefore, the computational domain now includes all points (i, j, k), with $i = 0, \cdots, NI + 2$, $i = 0, \cdots, NI + 2$, and $i = 0, \cdots, NI + 2$.

Applying a second order central difference operator δ_x to both sides of Eq. (13), we have:

$$\delta_x C^*_{i,j,k} = \delta_x(I+\frac{\Delta te}{2}\delta_y-\frac{\Delta tb}{2}\delta_y^2)(I+\frac{\Delta tf}{2}\delta_z-\frac{\Delta tc}{2}\delta_z^2)C^{n+1}_{i,j,k} \quad (18)$$

Since the operators in the right hand side of Eq. (18) commute, we have

$$\frac{C^*_{i+1,j,k} - C^*_{i-1,j,k}}{2\Delta x} = (I + \frac{\Delta te}{2}\delta_y - \frac{\Delta tb}{2}\delta_y^2) (I + \frac{\Delta tf}{2}\delta_z - \frac{\Delta tc}{2}\delta_z^2)(C_x)^{n+1}_{i,j,k} \quad (19)$$

where we have used the relation $\delta_x C \approx C_x$ in the right hand side of Eq. (19). Note that C_x is known as boundary conditions. Setting $i = 1$ and $i = NI + 1$, we have

$$\frac{C^*_{2,j,k} - C^*_{0,j,k}}{2\Delta x} = (I + \frac{\Delta te}{2}\delta_y - \frac{\Delta tb}{2}\delta_y^2) (I + \frac{\Delta tf}{2}\delta_z - \frac{\Delta tc}{2}\delta_z^2)(C_x)^{n+1}_{1,j,k} \quad (20)$$

$$\frac{C^*_{NI+2,j,k} - C^*_{NI,j,k}}{2\Delta x} = (I + \frac{\Delta te}{2}\delta_y - \frac{\Delta tb}{2}\delta_y^2) (I + \frac{\Delta tf}{2}\delta_z - \frac{\Delta tc}{2}\delta_z^2)(C_x)^{n+1}_{NI+1,j,k}$$

These two equations can be solved for $C^*_{0,j,k}$ and $C^*_{NI+2,j,k}$:

$$C^*_{0,j,k} = C^*_{2,j,k} - 2\Delta x(I + \frac{\Delta te}{2}\delta_y - \frac{\Delta tb}{2}\delta_y^2) (I + \frac{\Delta tf}{2}\delta_z - \frac{\Delta tc}{2}\delta_z^2)(C_x)^{n+1}_{1,j,k} \quad (21)$$

$$C^*_{NI+2,j,k} = C^*_{NI,j,k} + 2\Delta x(I + \frac{\Delta te}{2}\delta_y - \frac{\Delta tb}{2}\delta_y^2) (I + \frac{\Delta tf}{2}\delta_z - \frac{\Delta tc}{2}\delta_z^2)(C_x)^{n+1}_{NI+1,j,k} \quad (22)$$

which can be used as boundary conditions for solving Eq. (10).

Similarly, when solve Eq. (11), we apply the operator δ_y to both sides of Eq. (12) to get

$$\delta_y C^{**}_{i,j,k} = \delta_y(I + \frac{\Delta tf}{2}\delta_z - \frac{\Delta tc}{2}\delta_z^2)C^{n+1}_{i,j,k} \quad (23)$$

which leads to

$$\frac{C^{**}_{i,j+1,k} - C^{**}_{i,j-1,k}}{2\Delta y} = (I + \frac{\Delta tf}{2}\delta_z - \frac{\Delta tc}{2}\delta_z^2)(C_y)^{n+1}_{i,j,k} \quad (24)$$

Setting $j = 1$ and $j = NJ + 1$, respectively, in (24), we can solve for $C^{**}_{i,0,k}$ and $C^*_{i,NJ+2,k}$ as

$$C^{**}_{i,0,k} = C^{**}_{i,2,k} - 2\Delta y(I + \frac{\Delta tf}{2}\delta_z - \frac{\Delta tc}{2}\delta_z^2)(C_y)^{n+1}_{i,1,k} \quad (25)$$

$$C^{**}_{i,NJ+2,k} = C^{**}_{i,NJ\ ,k} + 2\Delta y(I + \frac{\Delta tf}{2}\delta_z - \frac{\Delta tc}{2}\delta_z^2)(C_y)^{n+1}_{i,NJ+1,k} \quad (26)$$

which can be used as boundary conditions for solving Eq. (11).

When solving Eq. (12), we can generate boundary conditions for $C^{n+1}_{i,j,0}$ and $C^{n+1}_{i,j,NK+2}$ ($i = 1, 2, ..., NI + 1$ and $k = 1, 2, ..., NK + 1$) in a straightforward way:

$$\frac{C^{n+1}_{i,j,2} - C^{n+1}_{i,j,0}}{2\Delta z} = (C_z)^{n+1}_{i,j,1} \quad (27)$$

$$\frac{C^{n+1}_{i,j,NK+2} - C^{n+1}_{i,j,NK}}{2\Delta z} = (C_z)^{n+1}_{i,j,NK+1} \quad (28)$$

which lead to

$$C^{n+1}_{i,j,0} = C^{n+1}_{i,j,2} - 2\Delta z(C_z)^{n+1}_{i,j,1} \quad (29)$$

$$C^{n+1}_{i,j,NK+2} = C^{n+1}_{i,j,NK} + 2\Delta z(C_z)^{n+1}_{i,j,NK+1} \quad (30)$$

Note that when (21) - (22) are used to approximate the Neumann boundary conditions, the difference operators δ_y and δ_z are applied to $(C_x)_{i,j,k}$ in (21) and (22), As a result, additional derivative boundary conditions are needed along the following eight lines:

$$\begin{array}{lcl}
(1,1,0) & \rightarrow & (1, NJ+1, 0) \\
(NI+1,1,0) & \rightarrow & (NI+1, NJ+1, 0) \\
(1,1,NK+2) & \rightarrow & (1, NJ+1, NK+2) \\
(NI+1,1,NK+2) & \rightarrow & (NI+1, NJ+1, NK+2) \\
(1,0,1) & \rightarrow & (1, 0, NK+1) \\
(1,NJ+2,1) & \rightarrow & (1, NJ+2, NK+1) \\
(NI+1,0,1) & \rightarrow & (NI+1, 0, NK+1) \\
(NI+1,NJ+2,1) & \rightarrow & (NI+1, NJ+2, NK+1)
\end{array}$$

where the points on the left side of the arrows are the starting points of lines, and those on the right are the end points. For example, the first line starts at $(1, 1, 0)$ and finishes at $(1, NJ + 1, 0)$.

Similarly, the difference operator δ_z is applied to $(C_y)_{i,j,k}$ in (25) and (26). As a result, additional derivative boundary conditions are needed along the following four lines:

$$\begin{array}{lcl} (1,1,0) & \rightarrow & (NI+1,1,0) \\ (1,NJ+1,0) & \rightarrow & (NI+1,NJ+1,0) \\ (1,1,NK+2) & \rightarrow & (NI+1,1,NK+2) \\ (1,NJ+1,NK+2) & \rightarrow & (NI+1,NJ+1,NK+2) \end{array}$$

The values of derivative boundary conditions along these lines can be generated by second order extrapolations from the known derivative boundary conditions on the surface of the computational domain.

3. Parallel implementations

There are three substeps (10), (11), and (12) in the ADI algorithm for each time step. In each substep, the equations should be assembled along either x, y, or z directions depending on the nature of the spatial operators. For example, in the first step, the spatial operators δ_x and δ_x^2 act on the x direction, so all equations along that direction in (10), corresponding to different i-index values, $i = 1, \cdots, NI + 1$, should be assembled to form a system of equations with a tri-diagonal coefficient matrix. Similarly, equations corresponding to different j-index values, $j = 1, \cdots, NJ+1$, in (11) and different k-index values, $k = 1, \cdots, NK + 1$, in (12) should be assembled. This leads to three sets of algebraic equations:

$$\mathbf{A}\mathbf{c}^*_{j,k} = \mathbf{H}\mathbf{c}^n_{j,k} + \frac{\Delta t}{2}(\mathbf{f}^{n+1}_{j,k} + \mathbf{f}^n_{j,k}),$$
$$j = 1, \cdots, NJ + 1, \quad k = 1, \cdots, NK + 1, \quad (31)$$
$$\tilde{\mathbf{A}}\mathbf{c}^{**}_{i,k} = \mathbf{c}^*_{i,k},$$
$$i = 1, \cdots, NI + 1, \quad k = 1, \cdots, NK + 1, \quad (32)$$
$$\bar{\mathbf{A}}\mathbf{c}^{n+1}_{i,j} = \mathbf{c}^{**}_{j,k},$$
$$i = 1, \cdots, NI + 1, \quad j = 1, \cdots, NJ + 1. \quad (33)$$

The computations to solve these systems of equations within each substep are independent to each other. For example, there are $(NJ+1)\times(NK+1)$ systems of equations in (31) that can be solved concurrently by different processors. Similarly, the $(NI + 1) \times (NK + 1)$ equations in (32) can be solved concurrently once all $\mathbf{c}^*_{j,k}$ has been calculated. This makes parallelization of the algorithm fairly straightforward, particularly in a shared memory programming memory environment [8]. The following is the structure of the parallel code:

Table 1. Maximum errors between the calculated solution and the exact solution at $T = 1.0$. e_1 is the maximum error from the fraction time step method (1st order). e_2 is the maximum error from the approximate factorization ADI method (first order). e_3 is the maximum error from the approximate factorization ADI method (second order). $\Delta t = \Delta x = \Delta y = \Delta z = h$.

h	0.5	0.25	0.125	0.0625	0.03125
e_1	2.7e-04	2.7e-04	6.9e+00	5.3e+14	8.8e+48
e_2	6.5e-03	3.3e-03	1.6e-03	8.1e-04	4.0e-04
e_3	2.3e-03	3.8e-04	8.5e-05	2.1e-05	5.3e-06

0. Data input and initialization

1. For time step $n = 1, \cdots, N$

- **2.** Solve $(NJ+1) \times (NK+1)$ systems of equations in Eq. (31) concurrently
- **3.** Solve $(NI+1) \times (NK+1)$ systems of equations in Eq. (32) concurrently
- **4.** Solve $(NI+1) \times (NJ+1)$ systems of equations in Eq. (33) concurrently

5. End time stepping loop

6. Output

Since our code was implemented in shared memory programming environment, OpenMP [2, 3] was used for loop-based parallelization. The key in the code development is to identify the most computation intensive loops and to determine shared and private variables.

To evaluate the scalability of the parallel code, we calculate speedups by dividing the execution time for solving a given problem on one processor by the execution times on multiple processors.

4. Numerical experiments

The following three-dimensional convection diffusion equation is used in our numerical experiment:

$$C_t + dC_x + eC_y + fC_z = aC_{xx} + bC_{yy} + cC_{zz} + F(x, y, z, t), \quad (34)$$
$$0 < t < 1\,, \quad (x, y, z) \in [0, 1] \times [0, 1] \times [0, 1], \quad (35)$$

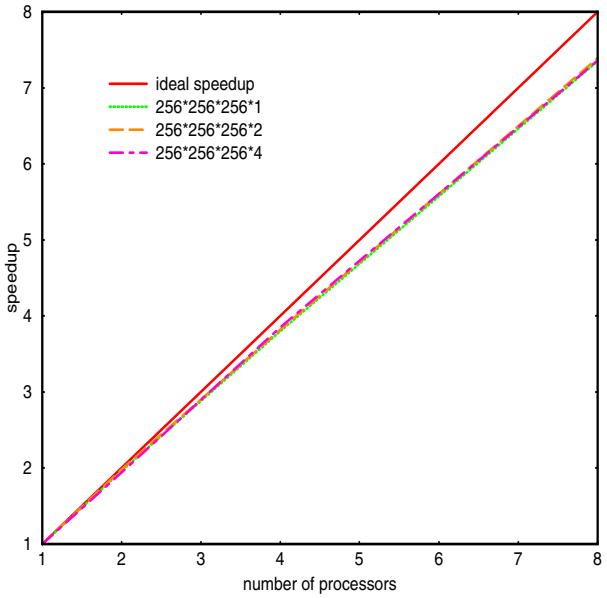

Figure 1. Speedup curves using a fixed grid of $256 \times 256 \times 256$ with different number of time steps 1, 2, and 4. It is clear that increasing the number of time steps does not affect the speedup

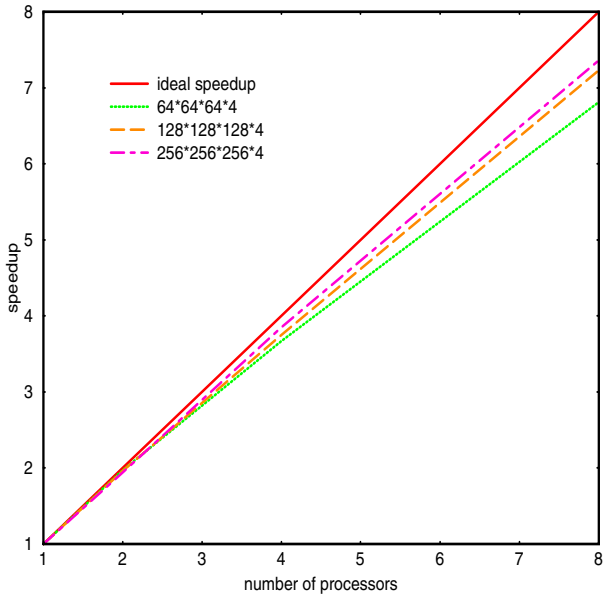

Figure 2. Speedup curves using different grids with the same number of time steps. It is clear that increasing the size of the grid also increase the speedup

$$
\begin{aligned}
C(x,y,z,0) &= \sin\frac{x}{3}\sin\frac{y}{3}\sin\frac{z}{3}, \\
(C_x)(0,y,z,t) &= \frac{1}{3}e^{-\frac{t}{3}}\sin\frac{y}{3}\sin\frac{z}{3}, \\
(C_x)(1,y,z,t) &= \frac{1}{3}e^{-\frac{t}{3}}\cos\frac{1}{3}\sin\frac{y}{3}\sin\frac{z}{3}, \\
(C_y)(x,0,z,t) &= \frac{1}{3}e^{-\frac{t}{3}}\sin\frac{x}{3}\sin\frac{z}{3}, \\
(C_y)(x,1,z,t) &= \frac{1}{3}e^{-\frac{t}{3}}\sin\frac{x}{3}\cos\frac{1}{3}\sin\frac{z}{3}, \\
(C_z)(x,y,0,t) &= \frac{1}{3}e^{-\frac{t}{3}}\sin\frac{x}{3}\sin\frac{y}{3}, \\
(C_z)(x,y,1,t) &= \frac{1}{3}e^{-\frac{t}{3}}\sin\frac{x}{3}\sin\frac{y}{3}\cos\frac{1}{3}
\end{aligned}
$$

where the source term $F(x,y,z,t)$ is set as

$$
\begin{aligned}
F(x,y,z,t) = \frac{1}{3}e^{-\frac{t}{3}}(&\cos\frac{x}{3}\sin\frac{y}{3}\sin\frac{z}{3} \\
&+\sin\frac{x}{3}\cos\frac{y}{3}\sin\frac{z}{3}+\sin\frac{x}{3}\sin\frac{y}{3}\cos\frac{z}{3}),
\end{aligned}
$$

to accommodate the exact solution

$$
C(x,y,z,t) = e^{-\frac{t}{3}}\sin\frac{x}{3}\sin\frac{y}{3}\sin\frac{z}{3}.
$$

The data in Table 1 shows the maximum error between the numerical solutions obtained using different methods and the exact solution at $t = 1$. The computation grid is $\Delta t = \Delta x = \Delta y = \Delta z = h$. Three methods are considered here: The notation e_1 represents the error, using $||\cdot||_\infty$, between the exact solution and the numerical solution calculated using the fraction time step method (6) - (8); e_2 represents the error between the exact solution and the numerical solution calculated using the ADI method (10) - (12) defining the Neumann boundary conditions at the time levels $n+\frac{1}{3}$ and $n+\frac{2}{3}$; and e_3 represents the error between the exact solution and the numerical solution calculated using the ADI method (10) - (12) and the new algorithm (21), (22), (25), (26), (29), and (30) to handle Neumann boundary conditions. It is clear from the table that e_1 shows that the fraction time step method is only conditionally stable, e_2 demonstrates only first order accuracy when the grid is refined, and e_3 demonstrate second order accuracy in both time and space.

Fig. 1 shows the speedup curves obtained using an SGI Origin2000 parallel computer with 8 processors. The grid size is $256 \times 256 \times 256$. Three cases are reported using 1, 2, and 4 time steps respectively. It is clear from the figure that running more time steps does not change the speedup curve. This is not surprising since the parallelization was done within a time step. The measured speedup is very close to the ideal speedup, which indicates that the algorithm discussed here is very scalable.

Fig. 2 shows the speedup curves obtained on the same computer using different grid sizes, i.e. $64 \times 64 \times 64$, $128 \times 128 \times 128$, and $256 \times 256 \times 256$. Four time steps are used in all cases. It is clear from the figure that the speedup improves as the grid size increases. This is due to the increased computation/communication ratio with increased grid size.

5. Conclusions

A new method for handling Neumann boundary conditions with the ADI algorithm is discussed in this paper. Numerical results demonstrate that the new algorithm is efficient, accurate, and scalable on parallel computers. It can be used in complex computational simulations for many problems described by three-dimensional convection diffusion equations.

References

[1] W. Ames. *Numerical Methods for Partial Differential Equations*. Academic Press, NY, 2nd edition, 1977.

[2] O. A. R. Board. *OpenMP Fortran Application Program Interface, Version 1.1*. OpenMP Architecture Review Board, 1999. URL: http://www.openmp.org/specs/.

[3] R. Chandra, L. Dagum, D. Kohr, D. Maydan, J. McDonald, and R. Menon. *Parallel Programming in OpenMP*. Morgan Kaufmann Publishers, 2000.

[4] B. Gustafson, H. Kreiss, and J. Oliger. *Time Dependent Problems and Difference Methods*. John Wiley & Sons, NY, 1995.

[5] D. Peaceman and H. Rachford. The numerical solution of parabolic and elliptic differential equations. *SIAM J.*, 3:28–41, 1955.

[6] H. Selim and M. Amacher. *Reactivity and Transport of Heavy Metals in Soils*. CRC/Lewis Publishers, Boca Raton, FL, 1997.

[7] H. Selim, M. Amacher, and I. Iskandar. *Modeling the transport of heavy metals in soil*. Monograph 90-2, U.S. Army Cold Regions Research and Engineering Laboratory, Hanover, NH, 1990.

[8] J. Zhu. *Solving Partial Differential Equations On Parallel Computers*. World Scientific Publishing Co. Pte. Ltd., 1994.