Comparison of methods to solve differential equations with serial & parallel implementation

Akshay Raut[1], Rupesh Shingte[2], Mujahed Patil[3], Aniket Jadhav[4]
[1,2,3,4]Mechanical Engineering dept., IIT Bombay, Maharashtra, India

Abstract: In this report, we are going to solve the one-dimensional heat transfer problem (temperature evaluation with time). We would be doing temperature evaluation versus time, given linear initial temperature distribution with maximum temperature being at the centre and fixed temperature at boundaries. This report aims to solve the one-dimensional heat transfer problem with the implicit finite difference method, explicit finite difference method, and Jacobi iterative method and compare results with an analytical method using separation of the variable to find an error in each method. We solve the problem using python programming with both sequential and parallel programming. Here, we are doing parallel implementation of these above methods using OpenMP. After that time study is performed for each method and time is compared for each method.

Keywords: Explicit, implicit, Jacobi iterative, parallel programming, OpenMP

I. INTRODUCTION

A one-dimensional transient heat conduction equation without heat-generating sources can be represented as in equation (1).

$$\rho c_p \frac{\partial T}{\partial t} = \frac{\partial}{\partial x}\left(k \frac{\partial T}{\partial x}\right) \tag{1}$$

Where ρ is density, c_p heat capacity, k thermal conductivity, T temperature, x distance, and t is time. Here the temperature is a function of both length and time, hence T(x, t). If the thermal conductivity, density, and heat capacity are constant over the model domain, the equation can be simplified to,

$$\frac{\partial T}{\partial t} = \kappa \frac{\partial^2 T}{\partial x^2} \tag{2}$$

Where $\kappa = \frac{k}{\rho c_p}$ is thermal diffusivity

The type of boundary conditions considered to solve the equation (2) is as follows,

Dirichlet BCs: These boundary conditions specifies the values of the solution function to be constant as shown in equation (3) on the domain 0 to L,

$$\begin{aligned} T(x=0,t) &= T_{\text{left}} = 0°C \\ T(x=L,t) &= T_{\text{right}} = 0°C \end{aligned} \tag{3}$$

The material considered is Silver with $k = 429\ {}^{W}/_{m.k}$, $\rho = 10490\ {}^{Kg}/_{m^3}$, $c_p = 233\ {}^{kJ}/_{kg.K}$

Initial Condition is

$$\begin{aligned} T &= \frac{200x}{L} && 0 \le x \le {}^{L}/_{2} \\ T &= \frac{-200x}{L} + 200 && {}^{L}/_{2} \le x \le L \end{aligned} \tag{4}$$

$T_{midpoint} = 100\ ℃$

System Specifications:

System: Windows 10, 8 GB RAM

Processor Specifications:

1) Product collection: 8th generation intel core i5 processor
2) Processor Number: i5 8265U CPU @1.60 GHz
3) Number of Cores: 4
4) Number of threads: 8
5) Max turbo frequency: 3.9 GHz

II. ANALYTICAL METHOD

The 1D heat conduction problem is solved with given homogeneous BCs and initial condition function by using the separation of the variable method. The expression obtained is

$$T_n = B_n \sin\left(\frac{n\pi x}{L}\right) e^{-\lambda^2 t} \tag{5}$$

Where,

$$B_n = \frac{800}{(n\pi)^2} \sin\left(\frac{n\pi}{2}\right),\ \lambda = \frac{cn\pi}{L},\ c = \sqrt{\frac{k}{\rho c_p}}$$

and $T = \sum_{n=1}^{\infty} T_n(x,t)$

Heat equation (2) can be solved using the forward difference methods like explicit and implicit methods.

III. EXPLICIT FINITE DIFFERENCE METHOD

Here the temperature at a node at time n+1 depends only on the previously known temperatures at time n. The explicit finite difference discretization of the equation (2) can be represented as

$$\frac{T_i^{n+1} - T_i^n}{\Delta t} = \kappa \frac{T_{i+1}^n - 2T_i^n + T_{i-1}^n}{(\Delta x)^2}$$

Which can be rearranged as,

$$T_i^{n+1} = T_i^n + \kappa \Delta t \frac{T_{i+1}^n - 2T_i^n + T_{i-1}^n}{(\Delta x)^2}$$

Hence the temperature at i^{th} node at time n+1 (T_i^{n+1}) can be computed from already known values of $T_{i+1}^n, T_i^n, T_{i-1}^n$

This method of evaluation can be represented schematically as shown in fig. 1, where all the node positions are represented on the space axis and their temperature values at different time intervals are represented along the time axis.

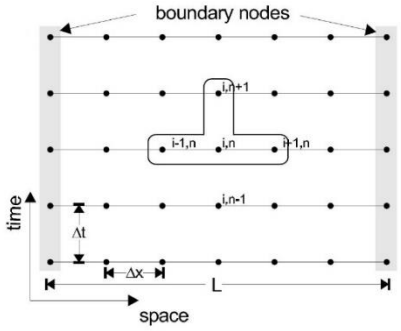

Fig. 1 Schematic of Explicit method

This method is coded in Python with Jupyter notebook IDE using the Numba library. The code is run for a different number of threads. The run-time vs No. of threads plot is as shown in fig. 2

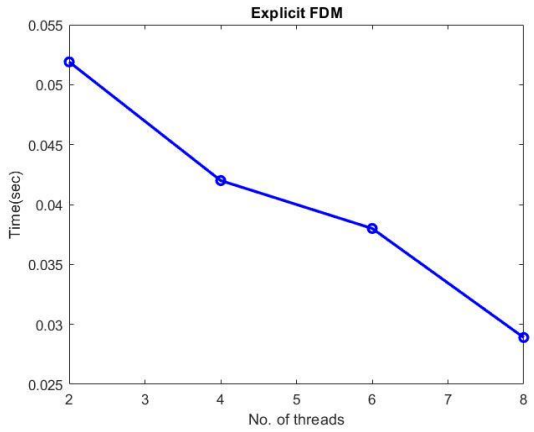

Fig. 2 Run-time vs No. of threads plot (Explicit method)

IV. IMPLICIT METHOD

Here the temperature at a node at time n+1 depends on the temperature of the same node at time n and the temperature of neighbouring nodes at time n+1. The implicit discretization for the equation (2) can be written as,

$$\frac{T_i^{n+1} - T_i^n}{\Delta t} = \kappa \frac{T_{i+1}^{n+1} - 2T_i^{n+1} + T_{i-1}^{n+1}}{(\Delta x)^2}$$

Which can be rearranged so that known terms are on the right and unknown terms are on the left,

$$-\frac{\kappa \Delta t}{(\Delta x)^2} T_{i+1}^{n+1} + (1 + 2\frac{\kappa \Delta t}{(\Delta x)^2}) T_i^{n+1} - \frac{\kappa \Delta t}{(\Delta x)^2} T_{i-1}^{n+1} = T_i^n \quad (6)$$

As can be seen from equation (6), an explicit relationship is absent, hence a linear system of equations has to be solved.

This method of evaluation can be represented schematically as shown in fig. 3,

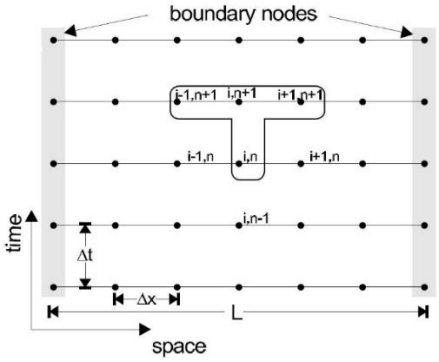

Fig. 3 Schematic of Implicit method

To solve the implicit finite difference scheme the boundary condition on the left and right boundary is achieved. Equation (6)can be written in matrix form as

$$Ax = b \quad (7)$$

For a six-node grid, the coefficient matrix A is,

$$\boldsymbol{A} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ -s & (1+2s) & -s & 0 & 0 & 0 \\ 0 & -s & (1+2s) & -s & 0 & 0 \\ 0 & 0 & -s & (1+2s) & -s & 0 \\ 0 & 0 & 0 & -s & (1+2s) & -s \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Where the unknown temperature vector x is,

$$x = \begin{pmatrix} T_1^{n+1} \\ T_2^{n+1} \\ T_3^{n+1} \\ T_4^{n+1} \\ T_5^{n+1} \\ T_6^{n+1} \end{pmatrix}$$

And the unknown right-hand side vector b is,

$$\boldsymbol{b} = \begin{pmatrix} T_{\text{left}} \\ T_2^n \\ T_3^n \\ T_4^n \\ T_5^n \\ T_{\text{right}} \end{pmatrix}$$

The implicit method is coded and the code is run for the different number of threads for parallelizing. The run-time vs No. of threads plot is as shown in fig. 4

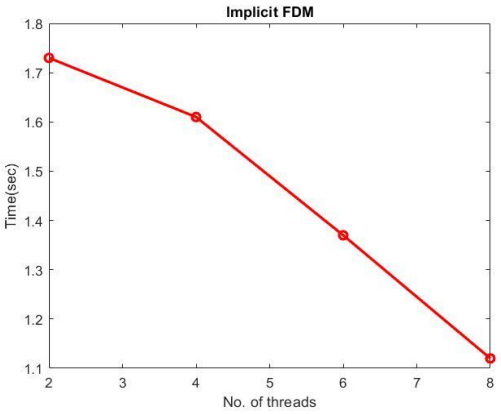

Fig.4 Run-time vs No. of threads plot (Implicit method)

Implicit methods are stable compared to explicit methods, but they can take a large time to compute. Hence the calculation can speed up using iterative methods. Iterative methods like the Jacobi method begin with an initial guess for the solution to the matrix equation. Each iteration updates a new estimate which converges on the exact solution.

V. JACOBI METHOD

The A matrix of the equation 7 is decomposed into a sum of lower-triangular (L), diagonal (D) and upper triangular (U) terms,

$$A = L + D + U$$

$$(D + L + U)\vec{x} = \vec{b} \qquad (8)$$

A matrix for backward-Euler method with Dirichlet boundary conditions is,

$$\text{A} = \begin{pmatrix} 1+2s & -s & & & & 0 \\ -s & & 1+2s & -s & & \\ & \ddots & & \ddots & & \ddots \\ & & -s & & 1+2s & -s \\ 0 & & & & -s & 1+2s \end{pmatrix}$$

Similarly,

$$\text{D} = \begin{pmatrix} 1+2s & & & 0 \\ & 1+2s & & \\ & & \ddots & \\ 0 & & & 1+2s \end{pmatrix}$$

$$\text{L} = \begin{pmatrix} 0 & & 0 \\ -s & \ddots & \\ & \ddots & \ddots \\ 0 & -s & 0 \end{pmatrix}$$

$$\text{U} = \begin{pmatrix} 0 & -s & 0 \\ \ddots & \ddots & \\ & \ddots & -s \\ 0 & & 0 \end{pmatrix}$$

The equation 8 can be rearranged as,

$$D\vec{x} = b - (L + U)\vec{x}$$

$$\vec{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

If $\vec{x}^{\,k}$ is the k^{th} estimate of the solution $A\vec{x} = \vec{b}$ then the $(k+1)^{\text{th}}$ estimate is,

$$D\vec{x}^{k+1} = b - (L + U)\vec{x}^{k}$$

Since D is diagonal $(D_{ij} = \delta_{ij}A_{ij})$, vector equation can be written for $\vec{x}^{k+1}$ for each component $(x_1^{k+1}, \ldots x_n^{k+1})$ as,

$$x_i^{k+1} = \underbrace{\frac{1}{A_{ij}}}_{D^{-1}} \left(b_i - \underbrace{\sum_{j \neq i} A_{ij} x_j^k}_{\text{L+U part}} \right), \qquad 1 \leq i \leq n \qquad (9)$$

This equation is used for coding the Jacobi method. The run-time vs No. of threads plot is as shown in fig. 5

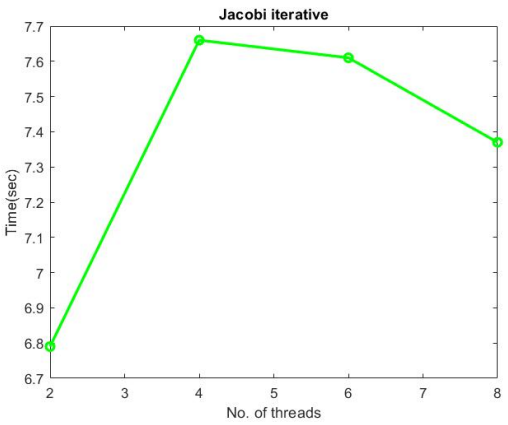

Fig. 5 Run-time vs No. of threads plot (Jacobi Iterative method)

VI. RESULTS

Temperature along the rod at t=0sec is as shown in fig.6

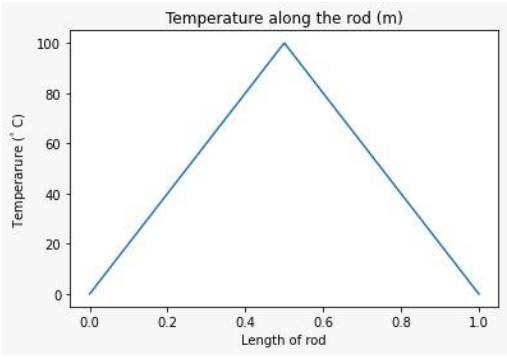

Fig. 6 Initial temperature distribution

The temperature along the rod after 5 sec i.e., t=5sec is as shown in fig.7

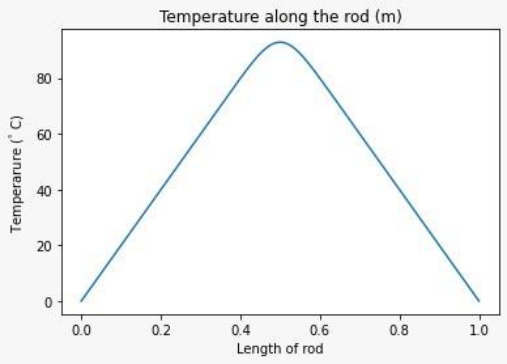

Fig. 7 Temperature distribution after 5 sec

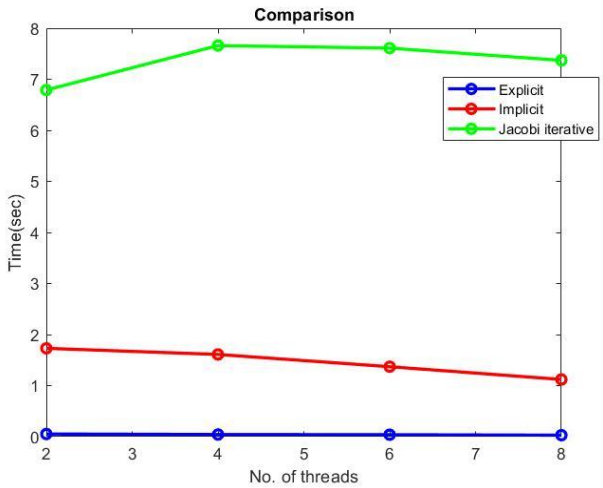

Fig. 8 Time vs No. of threads

The RMS error calculated and time for serial code is as shown in table 1. The considered methods are parallelized and the time taken for different number of threads is shown in table 2.

Table.1 RMS error & time for serial code wrt method

Method	Explicit FDM	Implicit FDM	Jacobi iterative
RMS error	0.007%	0.018%	0.016%
Serial code time(sec)	18.59	21.06	26.098

Table.2 Parallelization time for methods wrt No of threads

Parallelization	Explicit FDM(sec)	Implicit FDM(sec)	Jacobi iterative(sec)
2 threads	0.059	1.73	6.79
4 threads	0.042	1.61	7.66
6 threads	0.038	1.37	7.61
8 threads	0.0289	1.12	7.37

VII. CONCLUSION

The three methods are compared with analytical solution for their results and time taken for obtaining the results. It is found that Jacobi method gives more accurate result than FDM. Explicit and implicit method results are quite similar (0.1 % difference). Time study is performed for the methods so as to get faster and accurate method amongst all. OpenMP parallel coding methods with different number of threads is used to reduce the time taken for python script to run.

VIII. REFERENCES

[1]. Boris J.P. Kaus, Thorsten W. Becker, "Numerical Modeling of Earth Systems: An introduction to computational methods with focus on solid Earth applications of continuum mechanics"

[2]. Dr. Louise Olsen-Kettle, "Numerical solution of partial differential equations"

[3] C. T. Kelly, "Iterative methods for linear and non-linear equations"

[4] T. Bui, "Explicit and implicit methods in solving differential equations"