Project Report
ID5130
Jan-May, 2024, IIT Madras, Chennai

PERFORMANCE EVALUATION OF UNSTEADY STATE HEAT CONDUCTION IN 2D AND 3D DOMAINS USING MPI, OPENMP AND HYBRID MPI-OPENMP

Peddi Harishteja
AM23S018
Department of Applied Mechanics
Indian Institute of Technology, Madras

Guide:
Prof. Kameswararao Anupindi
Department of Mechanical Engineering
Indian Institute of Technology, Madras

ABSTRACT

Unsteady heat conduction problems in 2D and 3D domains will be solved and parallelized using a hybrid parallel programming paradigm combining Message Passing Interface (MPI) and Open Multi-Processing (OpenMP) and then parallel programs for MPI and OpenMP solely will be coded. This hybrid program approach aims to use MPI for within and across the nodes of a cluster communication and OpenMP for within each node communication. Different levels of parallelism can be achieved by combining MPI and OpenMP parallelization. To evaluate the performance of the hybrid code, parameters like execution time, speed up will be evaluated and plotted to compare the Hybrid MPI-OpenMP approach against solely MPI and OpenMP implementations.

Keywords: OpenMP, MPI, Hybrid MPI-OpenMP, Unsteady heat conduction, Finite Difference Method, Jacobi method

INTRODUCTION

In applications of mechanical engineering, most commonly observed heat transfer phenomenon is Conduction. Conduction can be classified into two types based on time dependency: steady state and unsteady state heat conduction. The temperature distribution generally depends on both space and time. If it is dependent solely on space and not on time, it is termed steady state; however, if it relies on both space and time, it is termed unsteady state. The current focus is on unsteady state heat conduction in both 2D and 3D domains without internal heat generation. The generalized unsteady heat conduction equation is a partial differential equation. To numerically solve this equation, the Finite Difference Method (FDM) is being used. For the numerical solution of the discretized equation, an iterative numerical method, the Jacobi method, is utilized. As the domain size increases, the time complexity also increases. Considering the high potentiality for parallelizing the Jacobi iterative-based solvers, various parallelization techniques such as OpenMP and MPI are implemented. Furthermore, to utilize the advantages of both OpenMP and MPI, a hybrid parallelization technique is developed and implemented. All the algorithms mentioned are developed using FORTRAN language and were finely tested on the in-house cluster called PAVAN (CFTC Lab, Applied Mechanics Department). Different nodes and cores are varied in each parallelization method to observe the nature of time complexity.

COMPUTATIONAL MODEL DESCRIPTION

A solid 2D domain of size (256 x 256 units) in unsteady state and without any internal heat source is modeled. Fig.1 presents the detailed description of the domain including boundary conditions.

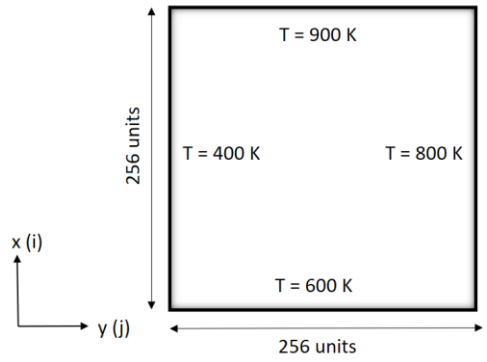


Fig.1. 2D Solid Domain

Similarly a solid 3D domain of size (100 x 100 x 100 units) in unsteady state and without any internal heat source is modeled. Detailed description of the domain is presented in Fig.2.

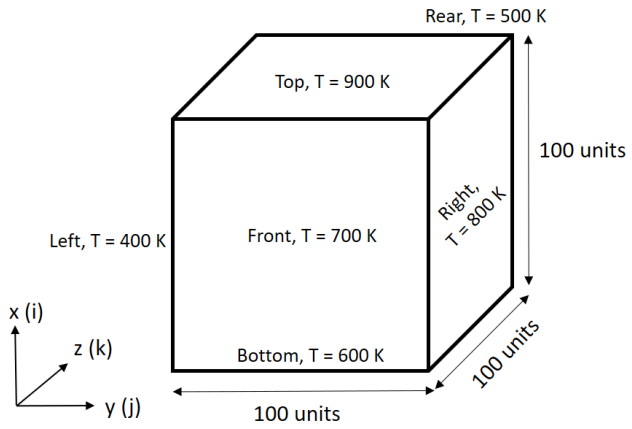


Fig.2. 3D Solid Domain

The objective is to obtain the temperature distribution due to conduction within the domains considered. Governing differential equation for generalized heat conduction in Cartesian co-ordinates is as follows:

$$\frac{\partial}{\partial x}\left\{k\frac{\partial T}{\partial x}\right\} + \frac{\partial}{\partial y}\left\{k\frac{\partial T}{\partial y}\right\} + \frac{\partial}{\partial z}\left\{k\frac{\partial T}{\partial z}\right\} + \dot{Q}_g = \rho C_p \frac{\partial T}{\partial t} \qquad (1)$$

Assumptions:

1) No internal heat generation , $\dot{Q}_g = 0$
2) Material is homogeneous ($\rho = constant$)
3) Thermal conductivity k is independent of temperature and position. So thermal diffusivity $\alpha = \frac{k}{\rho C_p}$

So equation (1) becomes:

$$\alpha\left\{\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2}\right\} = \frac{\partial T}{\partial t}$$

Using Central differencing scheme for spatial discretization,

$$\frac{\partial^2 T}{\partial x^2} = \frac{T^n_{i-1,j,k} - 2T^n_{i,j,k} + T^n_{i+1,j,k}}{\Delta x^2}$$

$$\frac{\partial^2 T}{\partial y^2} = \frac{T^n_{i,j-1,k} - 2T^n_{i,j,k} + T^n_{i,j+1,k}}{\Delta y^2}$$

$$\frac{\partial^2 T}{\partial z^2} = \frac{T^n_{i,j,k-1} - 2T^n_{i,j,k} + T^n_{i,j,k+1}}{\Delta z^2}$$

Using Forward time marching for temporal discretization,

$$\frac{\partial T}{\partial t} = \frac{T^{n+1}_{i,j,k} - T^n_{i,j,k}}{\Delta t}$$

Therefore discretized equation becomes,

$$\alpha\left\{\frac{T^n_{i-1,j,k} - 2T^n_{i,j,k} + T^n_{i+1,j,k}}{\Delta x^2} + \frac{T^n_{i,j-1,k} - 2T^n_{i,j,k} + T^n_{i,j+1,k}}{\Delta y^2} + \frac{T^n_{i,j,k-1} - 2T^n_{i,j,k} + T^n_{i,j,k+1}}{\Delta z^2}\right\} = \frac{T^{n+1}_{i,j,k} - T^n_{i,j,k}}{\Delta t}$$

Let $\frac{\alpha\Delta t}{\Delta x^2} = k_1$, $\frac{\alpha\Delta t}{\Delta y^2} = k_2$, $\frac{\alpha\Delta t}{\Delta z^2} = k_3$

Therefore final discretized equation for 3D domain becomes,

$$T^{n+1}_{i,j,k} = T^n_{i,j,k} + k_1\left\{T^n_{i-1,j,k} - 2T^n_{i,j,k} + T^n_{i+1,j,k}\right\} + k_2\left\{T^n_{i,j-1,k} - 2T^n_{i,j,k} + T^n_{i,j+1,k}\right\} + k_3\left\{T^n_{i,j,k-1} - 2T^n_{i,j,k} + T^n_{i,j,k+1}\right\}$$

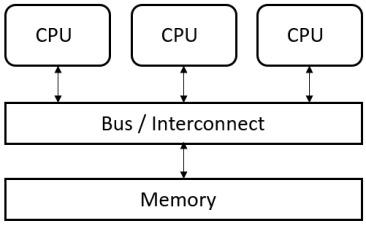
For 2D domain discretized equation is as follows,

$$T^{n+1}_{i,j} = T^n_{i,j} + k_1\left\{T^n_{i-1,j} - 2T^n_{i,j} + T^n_{i+1,j}\right\} + k_2\left\{T^n_{i,j-1} - 2T^n_{i,j} + T^n_{i,j+1}\right\}$$

PARALLELIZATION STRATEGY

i) OpenMP

OpenMP is an API for shared memory programming. MP in OpenMP stands for multi-processing. Each process will have independent access to available memory, and during programming, we will think of our system as a collection of cores / CPUs that can access main memory. For execution, it uses the forking and joining of threads method. It employs both specialized functions and preprocessor directives. One of OpenMP's most essential features is the ability to incrementally parallelize an existing serial program.



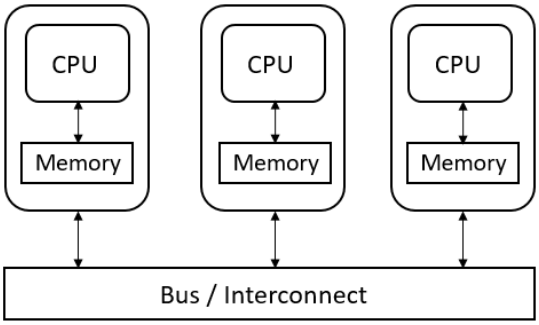
OpenMP Algorithm Implementation

```
1.  Initialization
2.  Grid generation
3.  Set boundary conditions
4.  for t = 1 : time steps
5.      !$omp parallel do // Open-MP Parallelization
6.      for i = 1 : grid points in x
7.          for j = 1 : grid points in y
8.              for k = 1 : grid points in z
9.                  Jacobi iteration for above mentioned discretization.
10.             end
11.         end
12.     end
13.     !$omp end parallel do
14. end
15. Calculate execution time
```

Fig.3. OpenMP Architecture and Algorithm Implementation

ii) MPI

The Message Passing Interface (MPI) is used to program distributed memory systems. In message passing applications, a program operating on a single core-memory pair is referred to as a process, and two processes can communicate by calling functions: one calls a send function, the other calls a receive function. MPI isn't a new programming language. It defines a collection of functions that can be accessed from FORTRAN, C, and C++ programs. Most of the heat conduction algorithms uses MPI because it works well for large-scale applications, but it is not recommended for smaller grids due to communication cost.



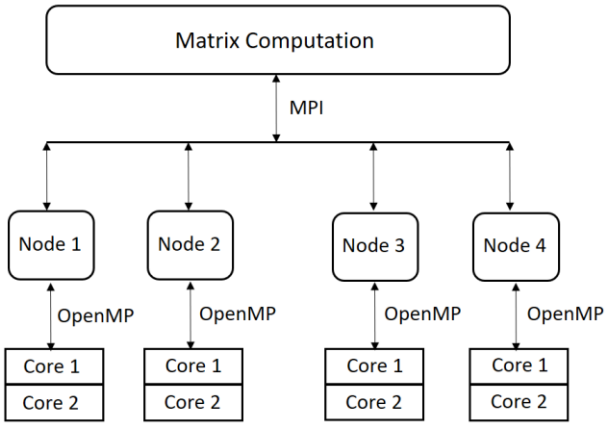
MPI Algorithm Implementation

1. Initialization
2. **MPI INITIALIZATION**
3. Grid generation
4. Domain decomposition
5. Set boundary conditions
6. for t = 1 : time steps
7. **MPI COMMUNICATION**
8. for i = 1 : grid points in x
9. for j = Decomposed domain for respective node
10. for k = 1 : grid points in z
11. **Jacobi iteration for above mentioned discretization.**
12. end
13. end
14. end
15. **MPI FINALIZE**
16. end
17. **Calculate execution time**

Fig.4. MPI Architecture and Algorithm Implementation

iIi) Hybrid OpenMP + MPI

Hybrid programming combines shared-memory (OpenMP) and distributed-memory (MPI) parallelisms to enhance performance. This hybrid program technique seeks to employ MPI for communication between the cluster nodes, as well as OpenMP for communication within each node. This means that MPI processes are generated to divide the workload over many computing nodes, and numerous threads are spawned within each process using OpenMP to further parallelize the calculation. Hybrid MPI + OpenMP employs decomposition as a two-level technique. On the MPI level, domain decomposition is performed. Parallelization on the OpenMP level is a decomposition, which can include loop-level parallelization.



Hybrid OpenMP + MPI Algorithm Implementation

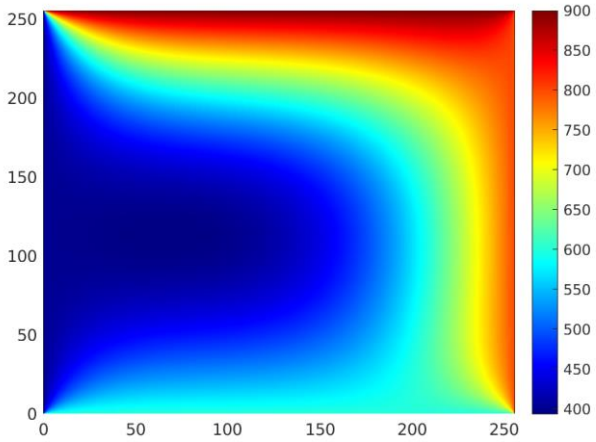
1. Initialization
2. **MPI INITIALIZATION**
3. Grid generation
4. Domain decomposition
5. Set boundary conditions
6. for t = 1 : time steps
7. **MPI COMMUNICATION**
8. **!Somp parallel do // Open-MP Parallelization**
9. for i = 1 : grid points in x
10. for j = Decomposed domain for respective node
11. for k = 1 : grid points in z
12. **Jacobi iteration for above mentioned discretization.**
13. end
14. end
15. end
16. **!Somp end parallel do**
17. **MPI FINALIZE**
18. end
19. **Calculate execution time**

Fig.5. Hybrid Architecture and Algorithm Implementation

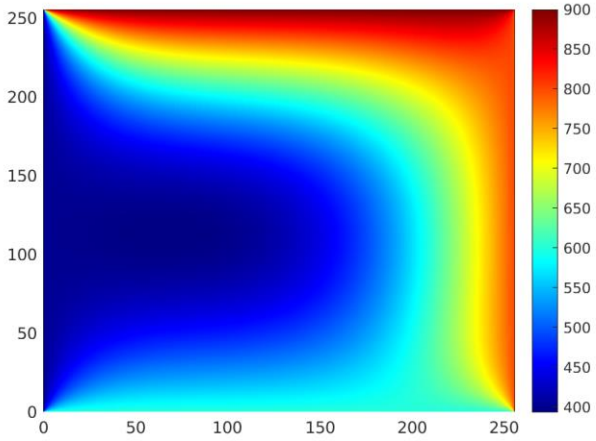
RESULTS

Unsteady state heat conduction is solved for the given 2D and 3D domains using FDM. Developed serial code for the

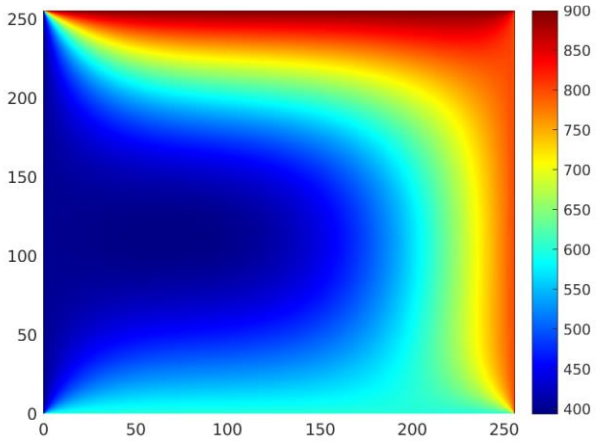
discretized equation is parallelized using OpenMP, MPI and Hybrid methods. For validation purpose temperature contours of parallel solvers are plotted and compared with that of serial solver. Fig.6. presents the temperature contours obtained for 2D domain at 2000^{th} time step.



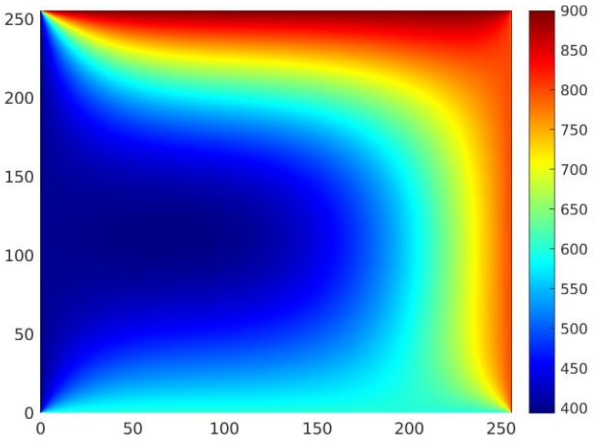
a)



b)



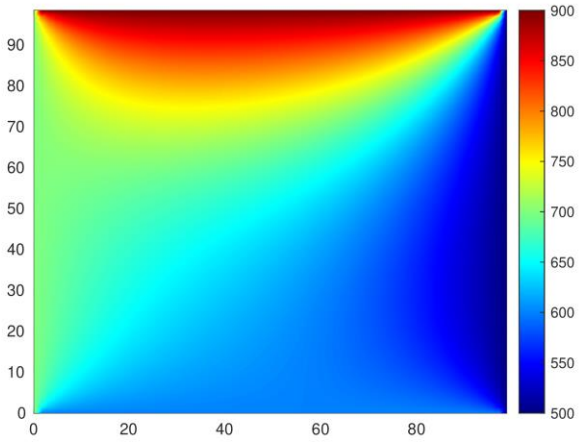
c)



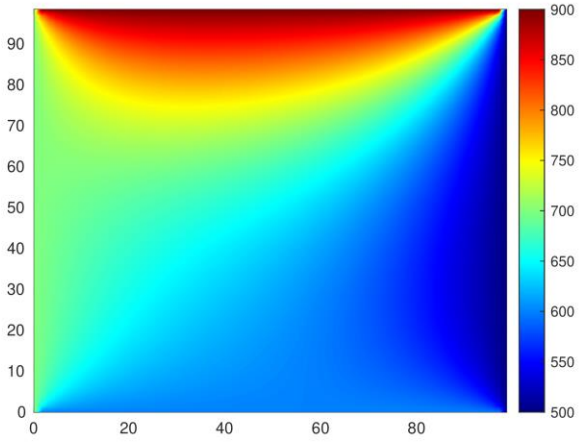
d)

Fig.6. Temperature contour for 2D domain a) Serial Solver b) OpenMP Solver c) MPI Solver d) Hybrid Solver

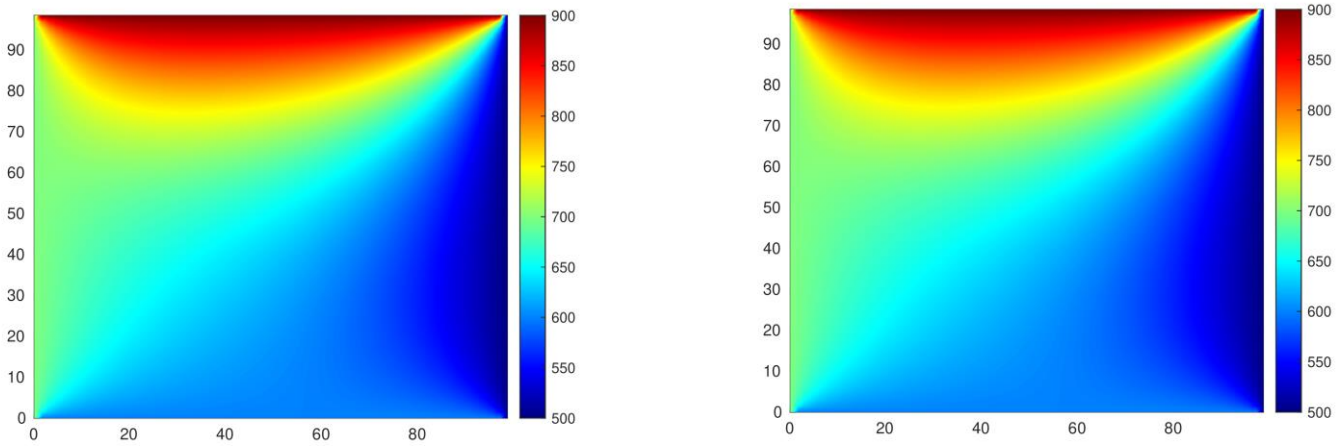
Fig.7. presents the temperature contours of a 2D plane (xz plane) at y=50 units for 2000^{th} time step in the given 3D computational domain.



a)



b)



c) d)

Fig.7. Temperature contour for 3D domain 2D plane (xz plane) at y=50 units
a) Serial Solver b) OpenMP Solver c) MPI Solver d) Hybrid Solver

After the validation of solvers, execution time of each solver is calculated and is tabulated below.

OpenMP	**No: of processors**	Serial	2	4	8
	Time (Seconds)	612.28	480.56	258.85	147.06

Pure MPI	**No: of processors**	Serial	2	4	8
	Time (Seconds)	612.28	251.35	120.43	58.026

Hybrid (OpenMP + MPI)	**Node count**	2			4			8		
	No: of cores	2	4	8	2	4	8	2	4	8
	Time (Seconds)	50.24	28.90	20.62	25.74	16.68	36.11	14.27	33.86	44.53

Table.1. Execution time for different solvers for the given 2D computational domain

OpenMP	**No: of processors**	Serial	2	4	8
	Time (Seconds)	907.27	881.46	513.98	385.08

Pure MPI	**No: of processors**	Serial	2	4	8
	Time (Seconds)	907.27	348.02	177.58	85.16

Hybrid (OpenMP + MPI)	**Node count**	2			4			8		
	No: of cores	2	4	8	2	4	8	2	4	8
	Time (Seconds)	132.58	51.17	38.84	48.61	28.85	59.21	25.22	53.98	74.05

Table.2. Execution time for different solvers for the given 3D computational domain

Fig.8. shows plots of execution time vs number of processes obtained for each solver for 2D computational domains.

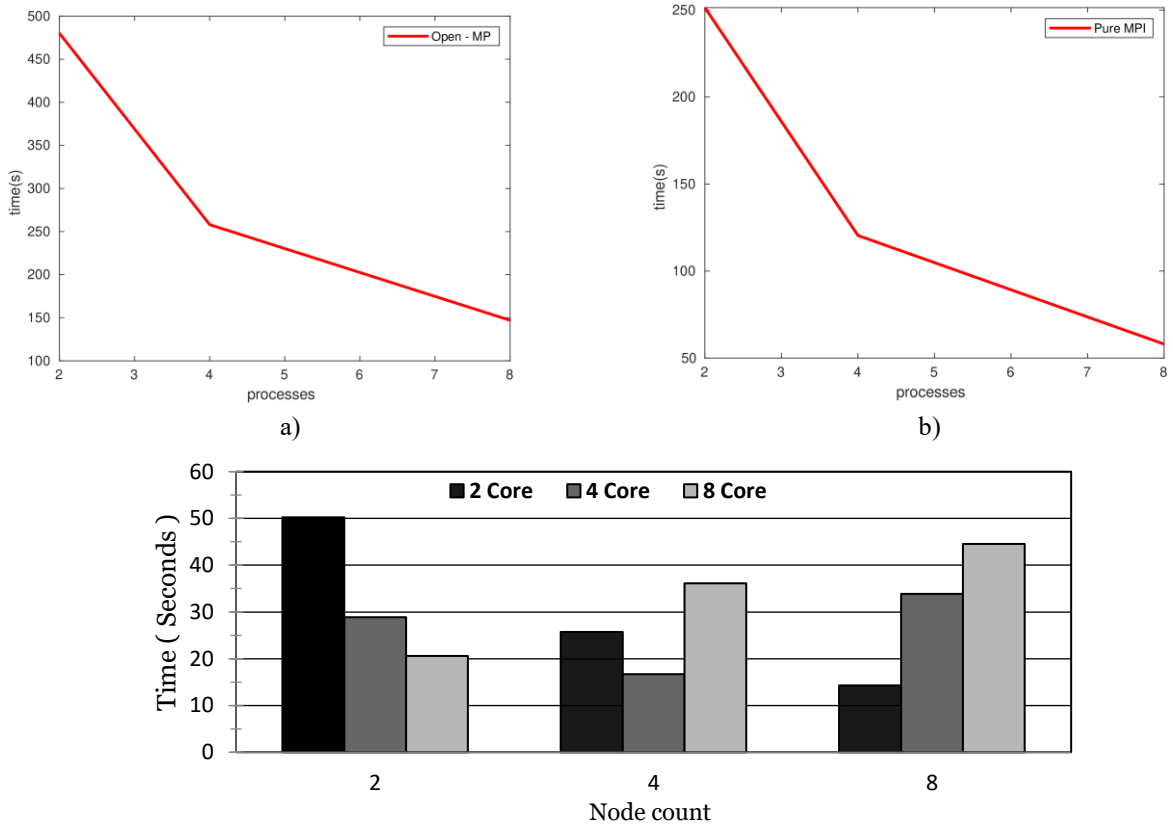
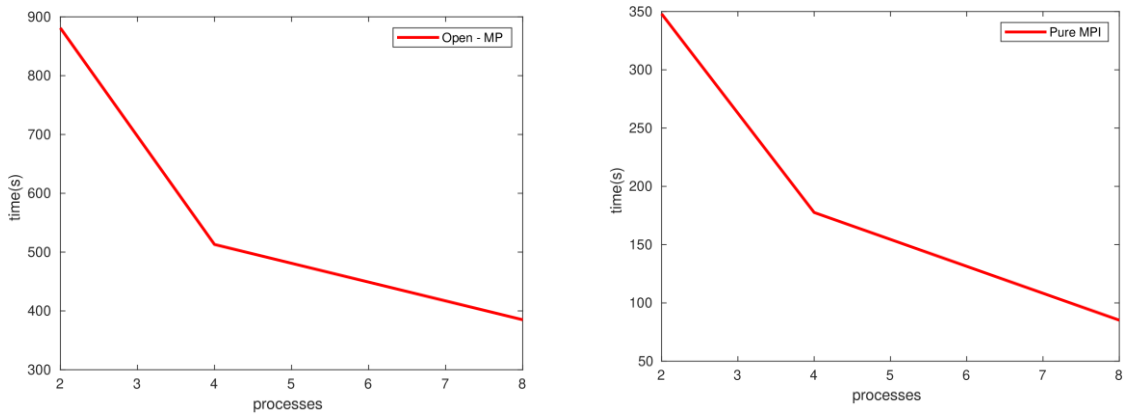
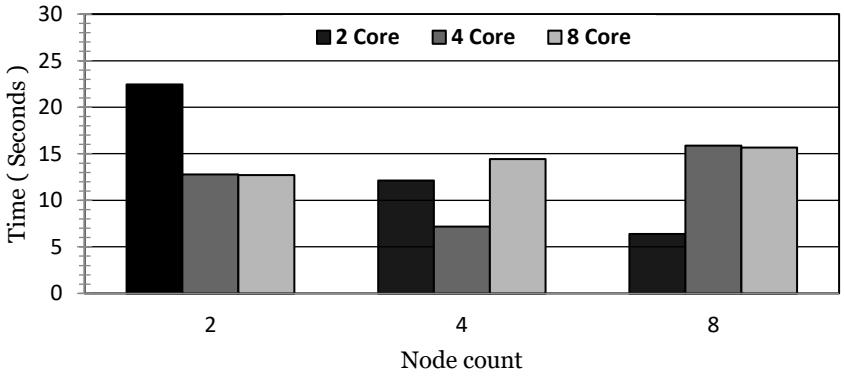


Fig.8. Execution time vs number of processes for a) OpenMP b) MPI c) Hybrid

Fig.9. shows plots of execution time vs number of processes obtained for each solver for 3D computational domains



a) b)



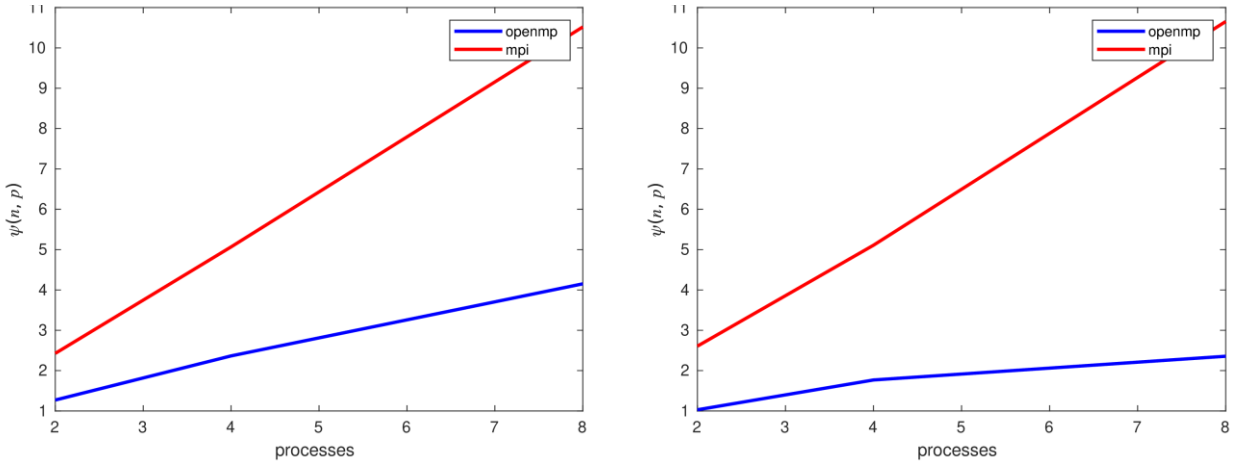
c)

Fig.9. Execution time vs number of processes for a) OpenMP b) MPI c) Hybrid

To know the performance of different solvers, speedup vs number of processes is plotted.

$$Speedup = \frac{Sequential\ execution\ time}{Parallel\ execution\ time} = \frac{T_s}{T_p}$$

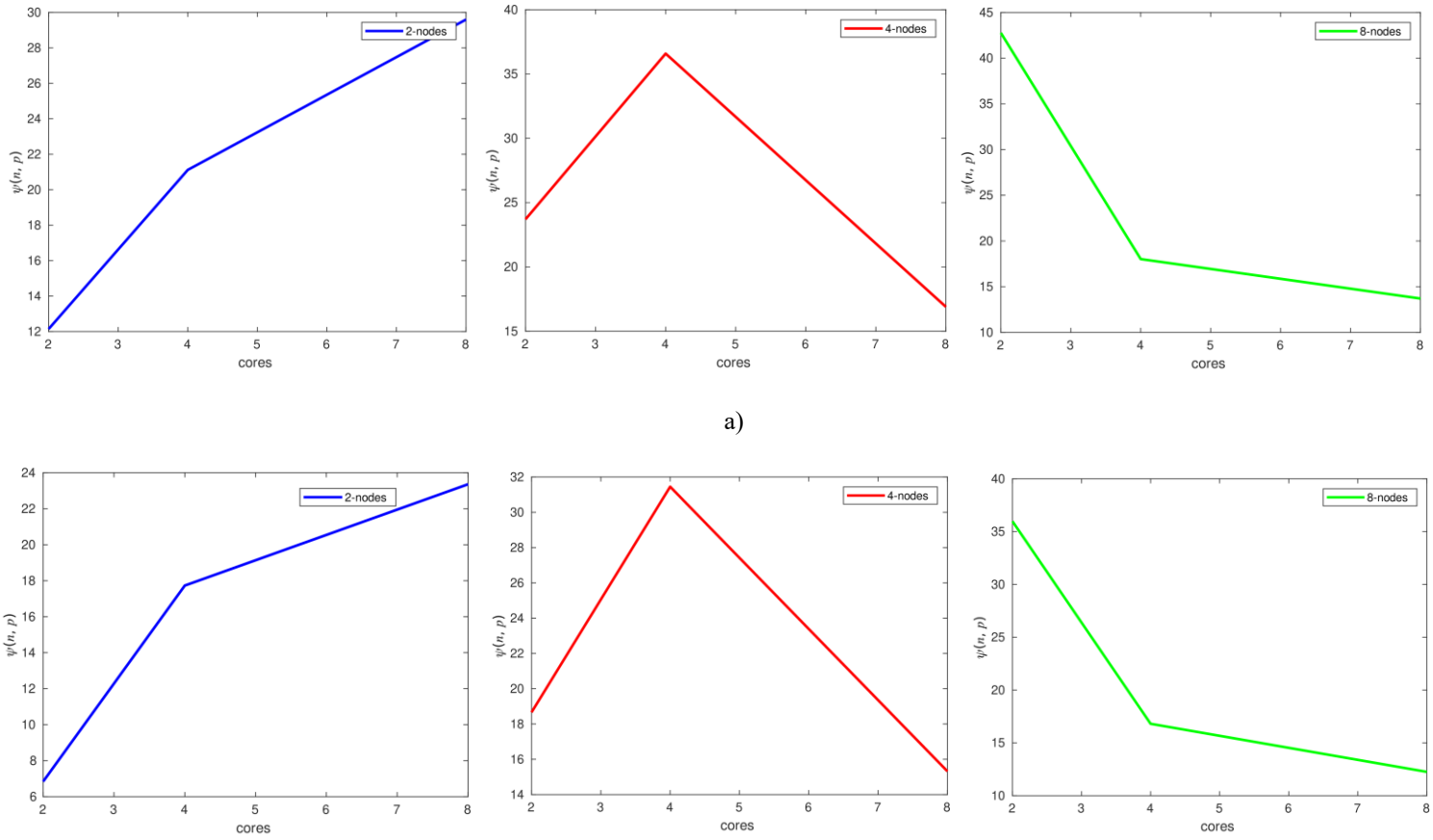
Fig.10. shows plots of speedup vs number of processes obtained for OpenMP and MPI solvers of 2D and 3D computational domain.



a) b)

Fig.10. Speedup vs number of processes for OpenMP and MPI solver for a) 2D domain b) 3D domain

Fig.11. shows plots of speedup vs number of processes obtained for Hybrid solver of 2D and 3D computational domain.



a)

b)

Fig.11. Speedup vs number of processes for Hybrid solver for a) 2D domain b) 3D domain

CONCLUSION

Unsteady steady heat conduction is solved for both 2D and 3D domain using FDM. Developed serial code is parallelized using different methods like OpenMP, MPI and Hybrid (MPI + OpenMP). Codes are validated by plotting temperature contours. Based on the results obtained it is concluded that for 2D and 3D domains: In OpenMP as we increase the number of cores time taken for execution is reduced. For MPI as number of nodes are increased, time taken for execution is decresing. But MPI takes less time compared to OpenMP for execution. In Hybrid solver different combinations of nodes (MPI parallelization) & cores (OpenMP parallelization) are being executed and it is observed that for a particular combination of nodes and core, it is taking less time compared to pure MPI and OpenMP solvers by taking advantage of Hybrid combination .In 2D and 3D domain , hybrid combinations of nodes (2,4,8) and cores (2,4,8) are executed and observed that combination of 8 nodes and 2 cores is taking comparatively less time than any other solver. For performance evaluation of solvers, parameters like execution time , speedup as a function of number of processes is being plotted.

REFERENCES

1. An Introduction to Parallel Programming - Peter S. Pacheco
2. Heat & Mass Transfer – Yunus A. Cengel & Afshin J. Ghajar
3. Numerical Recipes in FORTRAN90 – Second Edition