ID5130 - Parallel Scientific Computing

HYBRID PARALLELISATION OF UNSTEADY HEAT CONDUCTION

R Sai Ashwin
Department of Mechanical Engineering
Indian Institute of Technology
Madras, Tamil Nadu 600036
Email: me20b154@smail.iitm.ac.in

Karthik Sriram
Department of Metallurgical and Materials Engineering
Indian Institute of Technology
Madras, Tamil Nadu 600036
Email: mm20b032@smail.iitm.ac.in

ABSTRACT

This study explores the synergy between MPI (Message Passing Interface) and OpenMP (Open Multi-Processing) paradigms to tackle the 3D unsteady heat conduction problem. The motivation behind hybrid parallelization lies in leveraging the strengths of both MPI and OpenMP: MPI for inter-node communication and OpenMP for intra-node parallelism.

1 Introduction

High performance computing generally makes use of multiple cores/nodes, with the task at hand parallelised by a shared-memory scheme or a distributed memory scheme. In case of shared-memory parallelisation scheme, multiple cores of a single machine are used by spawning multiple threads which are then scheduled to be run on the cores. This is done with the help of libraries like OpenMP or POSIX Thread. Distributed memory parallelisation is achieved by coordinating message passing between all participating nodes. This eliminates the need for shared memory between nodes. This is achieved by the MPI standard, and many implementations of the standard are prevalent since. Each implementation has drawbacks. Shared memory parallelisation suffers from being constrained by resources present on the current node. While shared memory parallelisation will allow this to be mitigated, there exists overhead in passing messages between nodes. The constraint imposed by having only one node is exemplified in the development of compute resources Compute resources have begun scaling by adding multiple nodes with interconnects for data transfer. Such architectures favour parallelisation schemes that leverage a hybrid approach, that seeks to eliminate the communication overhead wherever possible, by utilizing shared memory parallisation schemes within a node, and the MPI standard across nodes. The main objective of this study is to demonstrate the benefit of such an architecture. The investigation centers on solving the unsteady heat conduction equation in 3D domains, where both spatial and temporal discretization play pivotal roles. The heat conduction equation, a fundamental partial differential equation governing heat transfer, will be discretized using finite difference methods.

2 Model

$$\frac{\partial}{\partial x}\left(\kappa\frac{\partial T}{\partial x}\right)+\frac{\partial}{\partial y}\left(\kappa\frac{\partial T}{\partial y}\right)+\frac{\partial}{\partial z}\left(\kappa\frac{\partial T}{\partial z}\right)+q_v=\rho c_p\frac{\partial T}{\partial t}$$

Where,

$$\kappa = \text{Thermal Conductivity}$$
$$\rho = \text{Density}$$
$$c_p = \text{Heat capacity}$$

In our analysis, we have considered the material of the volume under consideration to be Aluminium.

$$\kappa = 237W/m^2K$$
$$\rho = 2700g/cc$$
$$c_p = 0.9J/g-K$$

We consider the above equation in the particular case where there is no heat generation within the element and the material

is homogeneous and isotropic. This reduces the equation to the following:

$$\left(\frac{\partial^2 T}{\partial x^2}\right)+\left(\frac{\partial^2 T}{\partial y^2}\right)+\left(\frac{\partial^2 T}{\partial z^2}\right)=\frac{\rho c_p}{\kappa}\frac{\partial T}{\partial t}$$

Using the taylor series expansion to discretise the equation over our domain, we can express the discretised equation in the bulk of the material as:

$$\begin{aligned}&T_{n+1}[i][j][k]=T_n[i][j][k]+\\&\frac{\kappa\Delta t}{\rho c_p}\left[\frac{T_n[i+1][j][k]-2T_n[i][j][k]+T_n[i-1][j][k]}{\Delta x^2}\right.\\&+\frac{T_n[i][j+1][k]-2T_n[i][j][k]+T_n[i][j-1][k]}{\Delta y^2}\\&\left.+\frac{T_n[i][j][k+1]-2T_n[i][j][k]+T_n[i][j][k-1]}{\Delta z^2}\right]\end{aligned}$$

At the boundaries, we have a convective boundary condition:

$$q=hA(T_a-T)$$

In order to accomodate the above boundary condition, we analyse our problem in a control volume in the shape of a cube: By

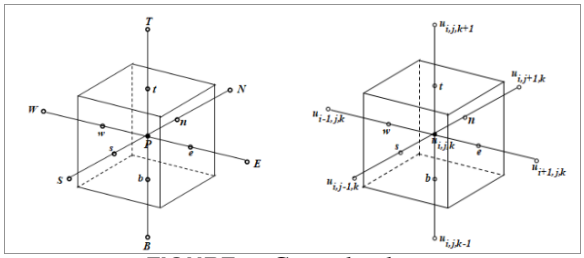

FIGURE 1: Control volume

equating the net heat flux entering the control volume with the heat responsible for its temperature rise, we can write:

$$\begin{aligned}&\frac{\partial T}{\partial t}\rho c_p\Delta x\Delta y\Delta z=\\&\kappa\left[(\frac{\partial T}{\partial x})_e-(\frac{\partial T}{\partial x})_w\right]\Delta y\Delta z+\\&\kappa\left[(\frac{\partial T}{\partial y})_n-(\frac{\partial T}{\partial y})_s\right]\Delta x\Delta z+\\&\kappa\left[(\frac{\partial T}{\partial z})_t-(\frac{\partial T}{\partial z})_b\right]\Delta x\Delta y\end{aligned}$$

Where,

$(\frac{\partial T}{\partial n})_i$ = Temperature gradient along axis 'n' at the face 'i' shown in Figure 1

Δn = Size of the cubic volume along the n-axis

At the boundaries, we may replace one or more temperature gradient terms with the corresponding convective heat transfer term $hA(T_a-T)$ and solve for the temperature at the next timestep. We may use a forward/backward difference scheme depending on which boundary we are dealing with. Since we would have 3 different types of boundaries in our volume, writing separate conditions for each of these boundaries would be cumbersome. The above approach wherein each term could be replaced by a convective heat transfer term greatly simplifies our code. The temperature gradient terms are multiplied by the results of a comparison that is true if the current unit has a boundary along each corresponding axis. Useful macros are defined to carry out this process for us, simplifying our expressions greatly. The macros are shown in Figure 3

```
36 //-Useful helper macros
37 #define pd2x(T,i,j,k,in) ((in==0) ? 0 : (T[i+1][j][k] - 2*T[i][j][k] + T[i-1][j][k])/dx_sq)
38 #define pd2y(T,i,j,k,in) ((in==0) ? 0 : (T[i][j+1][k] - 2*T[i][j][k] + T[i][j-1][k])/dy_sq)
39 #define pd2z(T,i,j,k,in) ((in==0) ? 0 : (T[i][j][k+1] - 2*T[i][j][k] + T[i][j][k-1])/dz_sq)
40 #define pdex(T,i,j,k) ((i == 0) ? ( (T[i+1][j][k] - T[i][j][k])/dx ) : ( (i == nx-1) ? ( (T[i-1][j][k] - T[i][j][k])/dx ) : 0 ))
41 #define pdey(T,i,j,k) ((j == 0) ? ( (T[i][j+1][k] - T[i][j][k])/dy ) : ( (j == ny-1) ? ( (T[i][j-1][k] - T[i][j][k])/dy ) : 0 ))
42 #define pdez(T,i,j,k) ((k == 0) ? ( (T[i][j][k+1] - T[i][j][k])/dz ) : ( (k == nz-1) ? ( (T[i][j][k-1] - T[i][j][k])/dz ) : 0 ))
43
44 #define isin(idx, nidx) ( idx !=0 && idx != nidx-1)
45
46 #define needs_boundary(i,j,k) (i == 0 | j== 0 | k == 0 | i == nx-1 | j == ny-1 | k == nz-1)
```

FIGURE 2: Helper Macros

3 Convergence

While testing the programs built using the above model, results were divergent. Further analysis showed that the equations written in code were consistent with the model described in Section 2. We discovered that Δt needs to be sufficiently small in

order for our program to achieve convergence. The analysis described in [1] shows the maximum value Δt can assume in order for the program to converge, using Von-Neumann Stability theory.

$$\Delta t \leq \frac{1}{2}\left(\frac{1}{\Delta x^2}+\frac{1}{\Delta y^2}+\frac{1}{\Delta z^2}\right)^{-1}$$

4 Parallelisation Schemes

We present three schemes used to parallelise the 3-D unsteady heat conduction problem and compare the speedup obtained. As seen in Section 2, the equation for all interior points has no spacial data dependence. The time stepping is done in a serial manner, while the temperature of each node for the next timestep is calculated in a parallel manner.

4.1 Shared Memory Only - OpenMP

Since the problem is quite easy to parallelise, we may collapse our $O(nx*ny*nz)$ nested loop using an openmp directive to be distributed between threads

```c
#pragma omp parallel for collapse(3) default(none) shared(T, T_new, t, DEBUG )\
    private(i,j,k) num_threads(num_threads)

        for(i=0;i<nx;i++)
            for(j=0;j<ny;j++)
                for(k=0;k<nz;k++)
                {
                    // Check if we are on the boundary
                    if(needs_boundary(i,j,k) == 0){
                        T_new[i][j][k] = T[i][j][k] +
                            kappa*dt*(pd2x(T,i,j,k,1) +  pd2y(T,i,j,k,1) + pd2z(T,i,j,k,1))/(rho * cp);
                    }else{
                        // If we are in the boundary, the heat balance equation changes. We can write a generalised equation for the heat flux
                        // using selectors that would use the index to determine whether each term in the sequence is active or not. This would
                        // simplify coding edge cases greatly
                        T_new[i][j][k] = kappa*dx*dy*dz*( pd2x(T,i,j,k,isin(i,nx)) + pd2y(T,i,j,k,isin(j,ny)) + pd2z(T,i,j,k,isin(k,nz)) )
                            + h*(Ta - T[i][j][k])*( dy*dz*(!isin(i,nx)) + dx*dz*(!isin(j,ny)) + dx*dy*(!isin(k,nz)) )
                            + kappa*(dy*dz*pdex(T,i,j,k) + dx*dz*pdey(T,i,j,k) + dx*dy*pdez(T,i,j,k));
                        T_new[i][j][k] /= rho*cp*dx*dy*dz/dt;
                        T_new[i][j][k] += T[i][j][k];
                        if(DEBUG){
                            printf("DEBUG: T[%d][%d][%d] @ t = %lf: \t %+.3lf\n", i ,j, k, t, T[i][j][k]);
                            printf("DEBUG: pdex(T,%d,%d,%d) @ t = %lf: \t %+.3lf\n", i, j, k, t, pdex(T,i,j,k));
                            printf("DEBUG: pdey(T,%d,%d,%d) @ t = %lf: \t %+.3lf\n", i, j, k, t, pdey(T,i,j,k));
                            printf("DEBUG: pdez(T,%d,%d,%d) @ t = %lf: \t %+.3lf\n", i, j, k, t, pdez(T,i,j,k));

                        }
                    }
                }

        memcpy(T, T_new, nx*ny*nz*sizeof(double));
```

FIGURE 3: OpenMP Implementation

Here, we have used the default schedule for distributing chunks of the collapsed for loops between threads. It is worthy to note that the optimal schedule would be to distribute "cuboids" of the volume under consideration to different threads, as each thread would benefit from the spatial locality of each chunk that it has to work on. This is due to the presence of caches on modern CPUs. In case the entire problem size does not fit in the L1 cache of the cpu (512 kB in our case), each thread would benefit if a majority of its chunk can be placed within the L1 cache (each CPU core has its own L1 cache). Therefore, we aim to distribute the working chunk in this manner to exploit the cache. In our case, an L1 cache is filled for a problem size of 40x40x40. We posit that the greatest speedup due to this approach would be for a problem size of 80x80x160 since the maximum number of concurrent threads on our machine is 16.

4.2 Distributed Memory only - MPI

While implementing a parallel solver using MPI, we will greatly benefit by simply decomposing the volume under consideration along the X-axis and distributing the resultant quanta of work amongs all of the MPI processes. Each MPI worker will have to send information about temperatures to its neighbours. Therefore, it makes logical sense for such a distribution.

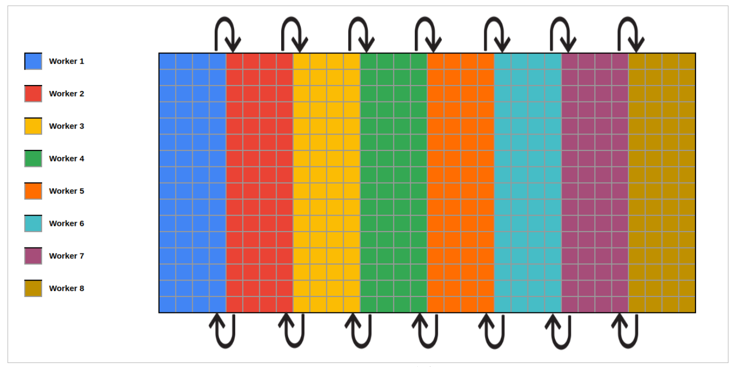

FIGURE 4: MPI Architecture

4.3 Hybrid OpenMP - MPI

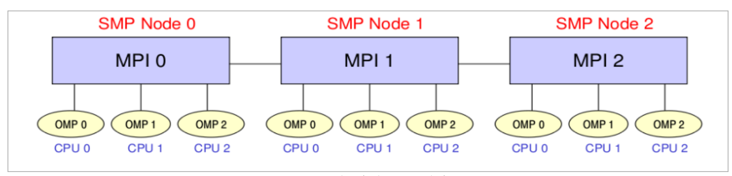

FIGURE 5: Hybrid Architecture

Figure 5 representes a typical setup where we would benefit from a Hybrid architecture. In Section 4.1, we reasoned about the problem size that would benefit greatly from the proposed parallelisation scheme. Similarly, any multiple of the above could be distributed in 80x80x160 chunks to each node that is similar to the one described. Within each node, we can create threads to process 40x40x40 chunks. To implement MPI in a multithreaded environment, we have to ensure that there are a minimal number of MPI transfers. In fact, MPI provides an API [2] for the same. To ensure that only 1 thread per MPI node carries out MPI layer data transactions, we have to use MPI_Init_thread() instead of MPI_Init(), with an MPI_THREAD_SINGLE argument to force MPI to ignore MPI calls from all threads except 1.

5 Results

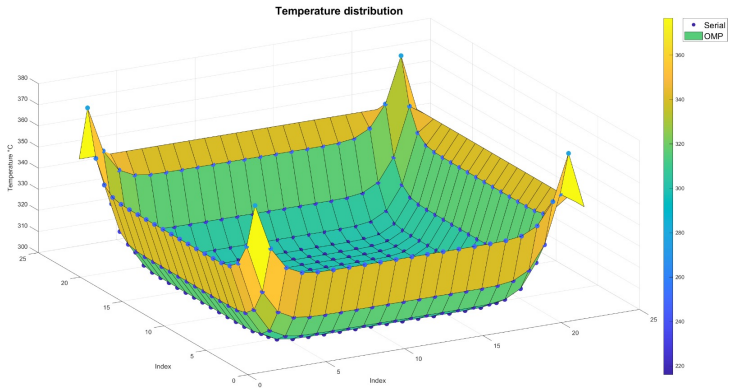

FIGURE 6: Temperature distribution: Omp vs Serial

5.1 Verification of parallel algorithms

To verify the correctness of our parallel solvers, we plot the solution from parallel solvers at $x = 0.5$. The results can be seen in Figure 6 and Figure 7. In the figures, the coloured circular markers represent the serial solution and the surface represents OMP/MPI. It can be seen that there is little to no deviation from the solution of our parallel solver

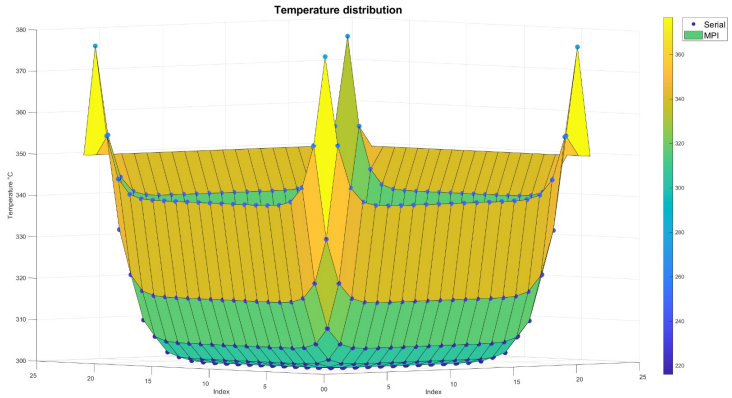

FIGURE 7: Temperature distribution: MPI vs Serial

5.2 Timings

The timings have been calculated by running the solvers on 2 different machines. They have been tabulated. Table 1 contains results from profiling on a personal laptop

TABLE 1: Statistics for solvers
Results from laptop: AMD Ryzen 7 6800HS
$\Delta x = 0.05 \quad \Delta t = 0.0001 \quad T_{\max} = 10.0$

Threads	4	8	16
Serial	19.021		
OMP	5.911	4.445	4.284
Hybrid (2 procs)	9.544	9.533	9.425
Hybrid (4 procs)	6.228	6.095	6.122
Hybrid (8 procs)	4.170	4.260	4.299
Num_procs	2	4	8
MPI	9.379	5.928	4.251

Table 2 contains results from profiling on the aqua HPC cluster at IITM.

TABLE 2: Statistics for solvers
Results from Aqua cluster at IITM
$\Delta x = 0.025 \quad \Delta t = 0.0001 \quad T_{\max} = 10.0$

Threads	2	4	8	16
Serial	37.419			
OMP	20.434	11.599	7.275	5.159
Hybrid (2 procs)	20.343	12.185	13.611	20.779
Hybrid (4 procs)	8.947	10.078	11.494	15.119
Hybrid (8 procs)	5.803	5.966	6.311	6.998
Num_procs	2	4	8	16
MPI	22.566	12.133	6.868	4.536

5.3 Hybrid solve temperature distribution

In Figures 8 through 13, we show the temperature evolution of the volume. The figures show the distribution of temperatures at intervals between 10 to 100 seconds, for a problem size of 25x25x25 elements. A Δt = 0.0001 has been used.

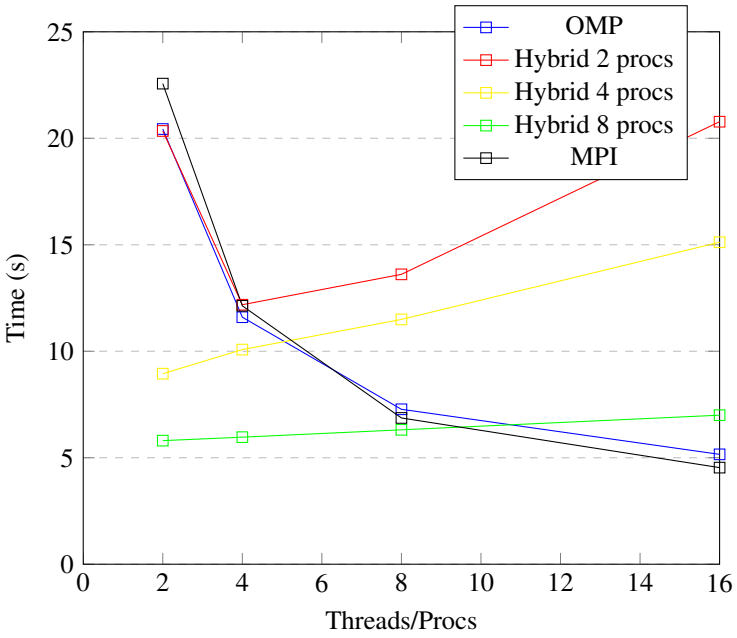

FIGURE 14: Time vs Threads/Procs

6 Conclusion

This study explored the feasibility of employing a hybrid architecture utilizing both MPI and OpenMP frameworks to tackle the intricate dynamics of 3D unsteady Heat Conduction problems. By leveraging the complementary strengths of MPI for inter-node communication and OpenMP for intra-node parallelism, we aimed to optimize computational efficiency.

Through numerical simulations conducted on a 1m x 1m x 1m cube subjected to varying ambient temperatures, we employed the fundamental 3D unsteady Heat Conduction equation to analyze heat transfer phenomena.

Comparative analyses were conducted across three distinct parallel architectures: exclusive utilization of OpenMP, exclusive utilization of MPI, and a hybrid MPI+OpenMP configuration. Runtimes were meticulously scrutinized across all architectures to discern performance differentials.

Our findings suggested that the optimal hybrid architecture is very dependent on the system's characteristics that one is using. For instance, our investigation revealed that the most efficient hybrid architecture on our system featured 8 processors paired with 16 threads. Nevertheless, a broader trend emerged, indicating that an increase in the number of processors within the hybrid architecture when accompanied by a higher thread count corresponded with a proportional rise in execution time. Conversely, for a given processor count, lower thread counts exhibited an improvement in performance.

In summation, this research contributes valuable insights into the intricate interplay between parallelization paradigms in computational heat transfer studies. By elucidating the nuanced trade-offs inherent in hybrid MPI+OpenMP architectures, our findings offer pragmatic guidance for optimizing computational efficiency across diverse computing environments.

REFERENCES

[1] E. Tsega, "A numerical solution of three-dimensional unsteady state heat equation," vol. 11, pp. 49–60, 10 2021.

[2] R. Rabenseifner. (2013) Hybrid mpi and openmp parallel programming. [Accessed 2nd May 2024]. [Online]. Available: https://openmp.org/wp-content/uploads/HybridPP_Slides.pdf

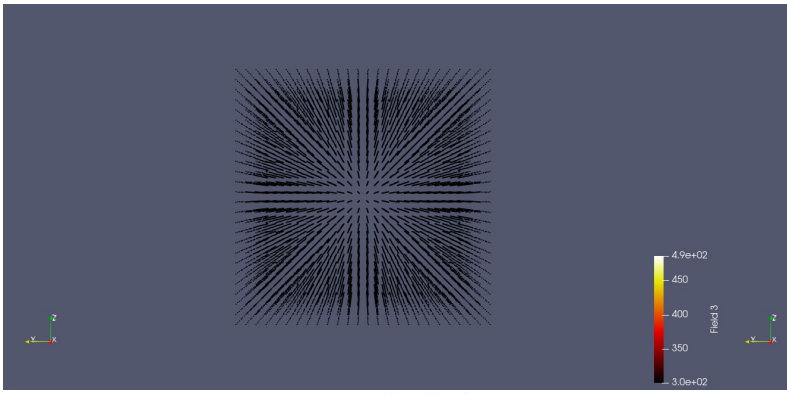

FIGURE 8: Temperature distribution: T = 0s

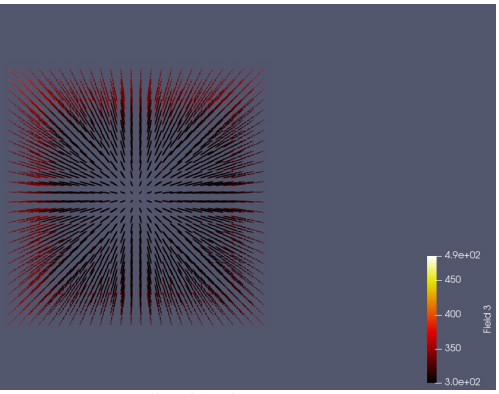

FIGURE 9: Temperature distribution: T = 20s

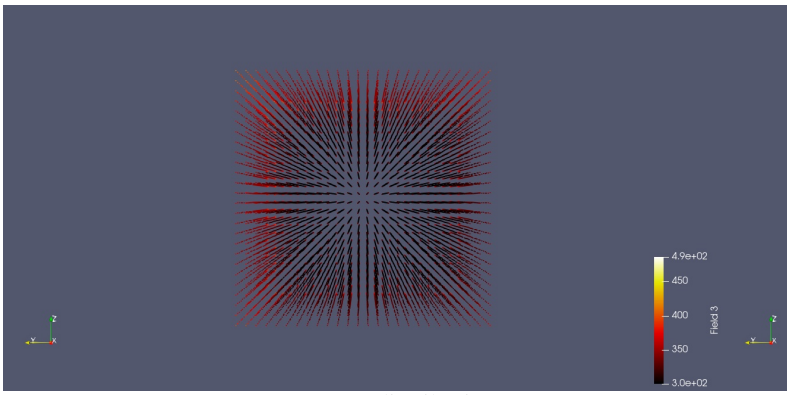

FIGURE 10: Temperature distribution: T = 30s

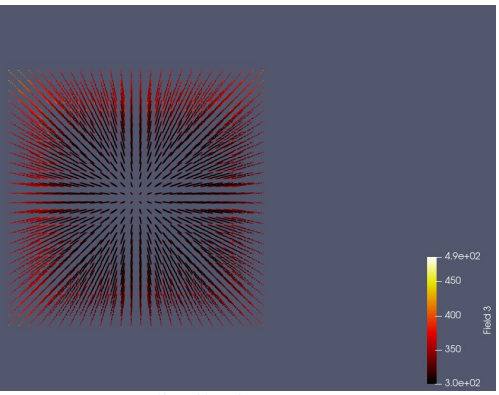

FIGURE 11: Temperature distribution: T = 40s

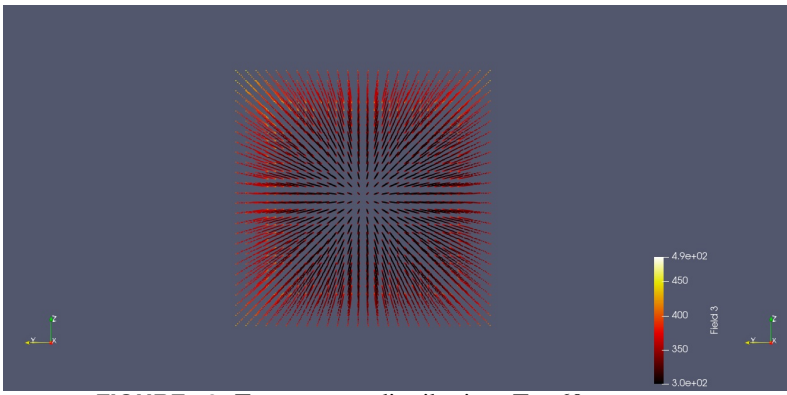

FIGURE 12: Temperature distribution: T = 60s

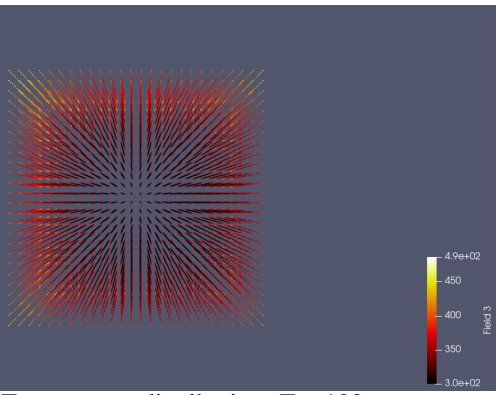

FIGURE 13: Temperature distribution: T = 100s