MULTISCALE THERMAL PROBLEM

Project-I (ME47601) report submitted to

Indian Institute of Technology Kharagpur

in partial fulfilment for the award of the degree of

Bachelor of Technology in Mechanical Engineering

by

Akella Kalyan Lakshmi Sriniwasa

(21ME10010)

Under the supervision of

Dr. Somnath Roy

Department of Mechanical Engineering

Indian Institute of Technology Kharagpur

Autumn Semester, 2024-25

November 14, 2024

DECLARATION

I certify that

1. I have done the work in this report under the guidance of my supervisor. The work has not been submitted to any other Institute for any degree or diploma.
2. The work has not been submitted to any other Institute for any degree or diploma. I have conformed to the norms and guidelines given in the Ethical Code of Conduct of the Institute.
3. I have conformed to the norms and guidelines given in the Ethical Code of Conduct of the Institute.
4. Whenever I have used materials (data, theoretical analysis, figures, and text) from other sources, I have given due credit to them by citing them in the text of the thesis and giving their details in the references. Further, I have taken permission from the copyright owners of the sources, whenever necessary.

Date: November 14, 2024 Akella Kalyan Lakshmi Sriniwasa

Place: Kharagpur 21ME10010

DEPARTMENT OF HUMANITIES AND SOCIAL SCIENCES

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

KHARAGPUR – 721302, INDIA

CERTIFICATE

This is to certify that the project report entitled "**Multi Scale Thermal Problem**" submitted by **Akella Kalyan Lakshmi Sriniwasa** (Roll No. 21ME10010) to the Indian Institute of Technology Kharagpur towards partial fulfilment of requirements for the award of the degree of **Bachelor of Technology in Mechanical Engineering** is a record of Bonafide work carried out by him under my supervision and guidance during Autumn Semester, 2024-25.

Professor Somnath Roy

Date: November 04, 2024

Department of Mechanical Engineering

Place: Kharagpur

Indian Institute of Technology Kharagpur

CONTENTS

Title Page		i
Declaration		ii
Certificate by the Supervisor		iii
Contents		iv
Abstract		1
Nomenclature		2
Chapter 1	Introduction	3
Chapter 2	Review of Literature	4
Chapter 3	Methodology	6
Chapter 4	Equations Used	7
Chapter 5	Brief Description of Computational Tools	8
Chapter 6	Results and Discussions	9
Chapter 7	Conclusion	15
Reference		17

ABSTRACT

Code parallelization is an essential tool for enhancing computational fluid dynamics (CFD) by reducing execution times and enabling complex simulations to run more efficiently. This project investigates the potential of OpenACC, a directive-based parallelization framework for C/C++, to optimize code performance across various scientific applications. As an initial benchmark, we applied OpenACC to a 2D heat equation model, which provided a preliminary reduction in runtime of approximately 65%. This experiment highlighted OpenACC's ability to simplify parallel processing for relatively straightforward calculations, laying the groundwork for more complex CFD problem-solving efforts.

Our primary objective then shifted toward ArtraCFD, a robust CFD solver. Through extensive profiling, we identified that ArtraCFD's computational load is concentrated in the *Solve* phase, specifically within the *EvolveSolution* component. Further analysis pinpointed that the *EvolveFluidDynamics* function accounted for the majority of this runtime, with the *DiscretizeTime* function and its nested *RungeKutta3* and *LLLU* functions contributing the most to the delay. Profiling also revealed that additional sub-functions within *LLLU* did not execute in the current test case, marking the profiling depth limit for this analysis. This directed our focus toward parallelizing the iterative loop in *EvolveSolution* that contains *EvolveFluidDynamics*, as it represents the highest potential for time reduction.

Initial results from the heat equation provide valuable insights into OpenACC's parallelization capabilities, while the findings from ArtraCFD profiling reveal critical sections for targeted optimization. These insights will guide continued parallelization efforts in BTP 2, where we aim to fully integrate OpenACC into ArtraCFD to achieve significant speedup for resource-intensive CFD applications. Together, these two stages of the project underscore the potential of OpenACC to drive efficiency gains in both basic and advanced scientific computations, aligning with the overarching goal of scalable, high-performance solutions in CFD.

Nomenclature

Symbol / Term	Description
CFD	Computational Fluid Dynamics, the field of study involving the simulation of fluid flows.
OpenACC	A directive-based parallel computing model for accelerating code on GPUs.
ArtraCFD	The CFD solver used in this project, designed for complex flow simulations using immersed boundary methods.
SIMPLE Algorithm	Semi-Implicit Method for Pressure-Linked Equations, a method for incompressible flow in CFD.
MAC Algorithm	Marker-and-Cell algorithm, designed for handling complex fluid interfaces in incompressible flows.
IBM	Immersed Boundary Method, a CFD technique for simulating flows around complex, irregular boundaries without conforming grids.
EvolveSolution	A function in ArtraCFD handling the main iterative solution process in the solver.
EvolveFluidDynamics	Sub-function in *EvolveSolution* responsible for calculating fluid dynamics updates.
DiscretizeTime	Function in ArtraCFD for time-stepping in simulations.
CFL Condition	Courant-Friedrichs-Lewy condition, a stability criterion for numerical simulations.
RK3, WENO	Runge-Kutta third-order and Weighted Essentially Non-Oscillatory, numerical methods for temporal and spatial discretization, respectively.
Inviscid Flow	Fluid flow with negligible viscosity, assuming no internal friction.
2D Heat Equation	The partial differential equation governing heat distribution in two dimensions.
Contour Plot	A graphical representation showing variable values as contours, often used for visualizing simulation results.
Jacobi Method	An iterative method used in solving of linear equations, used for solving heat and Laplace equations.

INTRODUCTION

The need for faster, more efficient simulations is increasing across industries that rely on complex modeling, particularly in fields like engineering, climate science, and aerodynamics. Computational fluid dynamics (CFD) plays a vital role in modeling fluid flows, but these simulations are often computationally expensive and time-consuming. This highlights the need for parallelization, which can distribute tasks across multiple processors, thus reducing computation time and enabling more efficient use of resources.

OpenACC, a directive-based parallelization framework for C/C++ and Fortran, offers a simple approach to parallelizing code. By using annotations, developers can parallelize existing programs with minimal changes, enabling efficient computation on both GPUs and CPUs. This project explores OpenACC's potential for accelerating two types of simulations: a 2D heat equation and ArtraCFD, a CFD solver.

The heat equation served as a starting point for applying OpenACC, providing valuable insights and achieving a 65% reduction in runtime. This provided an initial validation of OpenACC's effectiveness and helped familiarize us with directive-based parallelization. With this experience, the project moved on to ArtraCFD, where the goal was to identify bottlenecks for parallelization. Profiling revealed that most of ArtraCFD's runtime was concentrated in the Solve phase, particularly within the EvolveSolution function. Further analysis showed that the EvolveFluidDynamics function contributed significantly to this delay, with the DiscretizeTime and RungeKutta3 functions accounting for the most time. Additionally, profiling showed that the nested LLLU function did not run in the test case, which marked the profiling limit. These insights have highlighted key areas for parallelization, specifically within the EvolveSolution loop.

In the next phase (BTP 2), the focus will be on parallelizing the EvolveFluidDynamics function and optimizing the iterative loop in the EvolveSolution phase. The profiling work completed this semester has provided the foundation for these efforts and will guide further optimizations in ArtraCFD.

In summary, this project demonstrates OpenACC's ability to enhance both basic and complex scientific simulations. The results suggest that directive-based parallelization can significantly improve the performance and scalability of CFD solvers, offering a practical solution to the computational challenges of modern simulations. Future work will continue to build on these

insights, refining parallelization to meet the increasing demands for fast, resource-efficient CFD simulations.

REVIEW OF LITERATURE

Computational Fluid Dynamics (CFD) has become essential in engineering and scientific fields for simulating complex fluid flows. Methods such as the SIMPLE (Semi-Implicit Method for Pressure-Linked Equations) and MAC (Marker-and-Cell) algorithms provide foundational frameworks for solving the Navier-Stokes equations governing fluid motion, while parallelization strategies enable efficient simulations.

The SIMPLE algorithm solves incompressible fluid flows by decoupling the pressure and velocity fields, facilitating iterative solutions. Initially, a guessed pressure field is used to compute a preliminary velocity field. Through iterative correction, pressure and velocity adjustments satisfy the continuity equation, ensuring mass conservation. Using a finite volume approach, SIMPLE divides the computational domain into control volumes, updating flow variables systematically. The correction mechanism allows for convergence toward balanced pressure and velocity fields, delivering accurate results while reducing computational overhead.

The MAC algorithm, in contrast, is designed to handle complex multi-phase flows and fluid interfaces, such as free-surface flows. By using a staggered grid where velocity components are stored at cell faces and pressure at cell centers, MAC prevents artifacts like checkerboard pressure distributions. To track boundaries accurately, marker particles move with the fluid, marking interfaces and tracking their evolution. This structure enables MAC to capture complex behaviors like wave formation and separation, which are critical in free-surface flows. The MAC algorithm's iterative nature enables it to capture detailed flow characteristics accurately in dynamic fluid applications, such as turbulent flows or free-surface interactions.

Complementing these algorithms, the immersed boundary method (IBM) facilitates simulations around complex, moving geometries without needing body-conforming grids. IBM avoids the computational demands of grid deformation in domains with dynamic boundaries, incorporating boundary forces at fluid-solid interfaces. Initially developed for biological flows, IBM has proven broadly applicable, such as in flows involving rotating objects and other moving boundaries. The IBM approach demonstrated by Kumar et al. (2016) uses a dynamic search algorithm to tag immersed nodes, minimizing computation on solid nodes that do not affect fluid dynamics. By selectively tagging nodes, computational overhead is reduced while maintaining accuracy. Modified IBM implementations have shown promise in fields such as arterial flow and turbulent wakes where traditional methods are prohibitive.

Parallelization has transformed CFD by handling the high computational demands of detailed simulations. The IBM structure adapts well to parallel processing as its non-conforming grid enables independent updates across fluid and solid regions. GPU-based parallelization, explored by Raj et al., accelerates IBM by leveraging GPU cores, allowing efficient task distribution for high-resolution simulations. The MAC algorithm's staggered grid layout also supports parallelization, as velocity and pressure updates are minimally dependent on neighboring cells. The SIMPLE algorithm's iterative correction framework further benefits from parallelization, executing each iteration concurrently across distributed grids. Parallelization enables faster, larger-scale simulations, significantly enhancing efficiency.

The SIMPLE and MAC algorithms each offer distinct strengths. SIMPLE's decoupling of pressure and velocity fields supports steady or mildly unsteady flows but faces limitations with dynamic interfaces. Meanwhile, MAC's staggered grid and marker structure provide greater precision in capturing fluid boundaries, making it suitable for free-surface flows despite being more computationally intensive. IBM complements these methods with its efficient handling of complex geometries and moving boundaries, reducing computational costs. While IBM's high solid-to-fluid node ratio initially presented challenges, strategies like selective re-tagging and localized updates have expanded its utility.

In summary, IBM, SIMPLE, and MAC algorithms are powerful CFD tools, enhanced by parallel computing to support faster and more scalable simulations. Algorithm refinements and parallelization advancements promise to further enhance CFD's applicability and efficiency across a range of engineering, physical sciences, and industrial applications. Together, these methods offer adaptable and reliable ways to model complex fluid interactions across diverse scenarios.

METHODOLOGY

The experimental procedures began with simulating a 2D heat diffusion equation on a 200x200 grid with a diffusion constant of 0(steady state heat equation). The boundary conditions were to T(x,0)=0, T(0,y)=0, T(1,y)=1 and T(x,1)=1. Then, the values of the cells were initialized to 0 and computed using the Jacobi method. The code is run with both CPU (no parallelization) and GPU (with OpenACC parallelization), for number of steps varying from 100000 to 1000000.The runtime was measured for each test case on both a standard CPU and an OpenACC-optimized version utilizing loop collapse for parallelization. Each output was also plotted to verify the parallelized code's accuracy. This method aimed to observe whether OpenACC significantly reduces execution time and to assess performance scalability across time and space domains, thereby providing a baseline for subsequent optimizations applied to ArtraCFD.

The second experiment involved using ArtraCFD to simulate the flow of an inviscid fluid over a stationary cylinder, with an initial horizontal velocity of 2.41981 m/s along the x-axis, a density of 3.67372 kg/m³, and a pressure of 9.04545 Pa. Profiling focused particularly on the *Solve* function to identify high-computation sections for later parallelization. This setup allowed for an in-depth analysis of the solver's performance and flow behavior, with the results visualized through contour plots of pressure and velocity distributions around the cylinder, validating the model's accuracy in simulating inviscid flow. This experiment's findings will guide optimization efforts for enhanced performance in future stages. The results obtained at the end of the simulation were plotted on contour plots along the surface. The parameters that were plotted were u, v P, ρ and T.

EQUATIONS USED

Immersed Boundary Method- ArtraCFD

The immersed boundary method developed in this paper uses a **three-step flow reconstruction scheme** to handle complex, irregular, and moving boundaries in fluid simulations.

1. **Prediction Step:** For each boundary point on an object, an initial guess for flow properties (like velocity or pressure) is made using inverse distance weighting from nearby fluid points. This weighting favors closer points, giving an estimated value at the boundary.

$$\varphi_I{}^* = \frac{\sum \omega(d_N)\varphi_N}{\sum \omega(d_N)}$$

Where $\varphi_I{}^*$ is the predicted value of flow at the image point I, d_N is the distance from boundary point I to fluid node N, ω is the weighting factor based on d_N and φ_N is the known value of the flow variable at node N.

2. **Boundary Condition Enforcement:** This guess is corrected based on physical boundary conditions. For instance, if no fluid movement is allowed (no-slip), the boundary value is fixed (Dirichlet condition); for smooth flow, gradients are adjusted (Neumann condition).

$$\varphi_O = \varphi_I - ||x_I - x_O|| \frac{\partial \varphi_O}{\partial n}$$

Where φ_O is corrected value of flow variable at boundary point O, φ_I is predicted value at boundary image point I $x_I \; and \; x_O$ are the positions of points I and O, and n is the normal direction, relevant in Neumann conditions.

3. **Correction Step:** A final value combines the prediction and boundary-enforced correction, ensuring both the initial guess and physical conditions are respected.

$$\varphi_I = \frac{\sum \omega(d_N)\varphi_N + \sum \omega(d_O)\varphi_O}{\sum \omega(d_N) + \sum \omega(d_O)}$$

Where φ_I is the final value at image point I after correction and the other variables are the same as previously mentioned. In all equations, φ is any fluid property.

This approach is validated on fluid scenarios with stationary, moving, and curved boundaries, showing accuracy in reproducing phenomena like shock waves and vortex preservation.

2-Dimensional Heat Equation

The 2-Dimensional heat equation at any point is given as

$$\frac{\partial T}{\partial t} = \alpha(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T)}{\partial y^2}$$

Where T(x,y,t) is the temperature of the region, and α is the thermal diffusivity constant. In our experiments, $\alpha = 0$.

Navier Stokes equation in 2-Dimensions

$$\frac{\partial \boldsymbol{u}}{\partial \mathrm{x}} + (\boldsymbol{u}.\nabla)\boldsymbol{u} = \frac{-1}{\rho}\nabla\rho + \vartheta\nabla^2\boldsymbol{u} + \boldsymbol{f}$$

In the case of our experiment, we have $\vartheta = 0$, i.e. the flow is inviscid.

BRIEF DESCRIPTION OF COMPUTATIONAL TOOLS

OpenACC: OpenACC is a parallel programming model that provides a set of compiler directives for offloading computations from the CPU (host) to the GPU (accelerator) in a way that is high-level and accessible. Designed for C/C++ and Fortran, OpenACC enables developers to add parallelism to existing code with minimal restructuring, making it particularly useful for porting legacy software. With similarities to OpenMP, OpenACC uses a host-directed execution model, where the CPU controls program flow and manages memory transfer, while intensive computations are handled by the GPU. It includes data clauses such as `copy`, `copyin`, `copyout`, and `present` to manage data transfer between the host and device. Developers can define parallel regions with `parallel` or `kernels` directives, offering a choice between manual and automatic work-sharing. Unlike CUDA or HIP, which offer direct access to GPU hardware features and optimizations, OpenACC provides an easier, more portable option that can compile the same code to both CPU and GPU. Extensive resources are available at http://openacc.org.

ArtraCFD: ArtraCFD is a Computational Fluid Dynamics (CFD) solver, which uses **RK2** and **RK3** temporal discretizations, **WENO3** and **WENO5** (convective fluxes) + 2nd order central scheme (diffusive fluxes) spatial discretizations, and uses the Immersed Boundary Method and operator splitting for boundary treatment and fluid solid interactions respectively. Please refer to the GitHub repository https://github.com/mohuangrui/ArtraCFD/tree/master for further details on the solver.

RESULTS AND DISCUSSIONS

2D Heat Equation Solution

The results of the plots are given in the following figures.

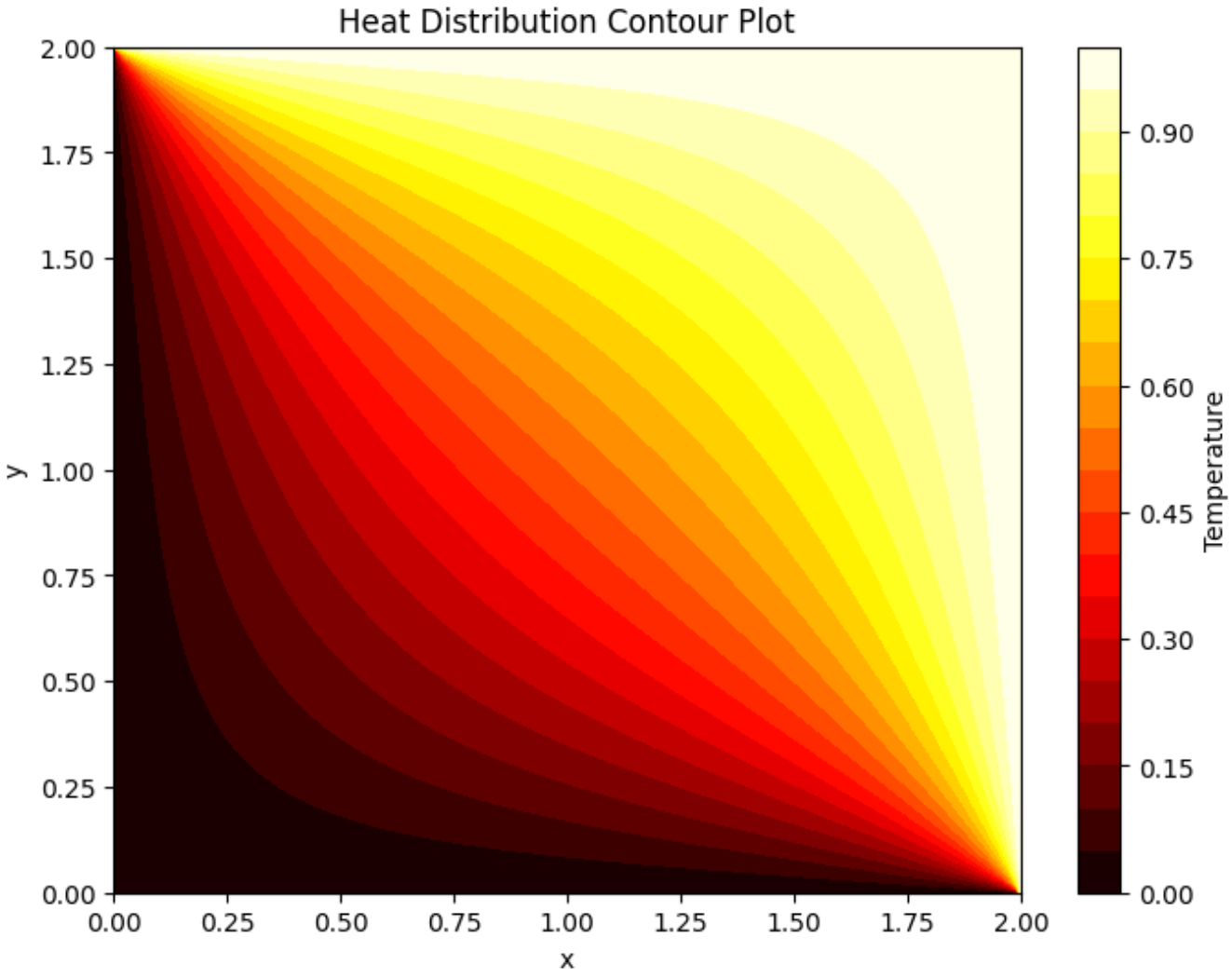

Figure 1: Contour plot of Temperature for the equation when solved with no parallelization.

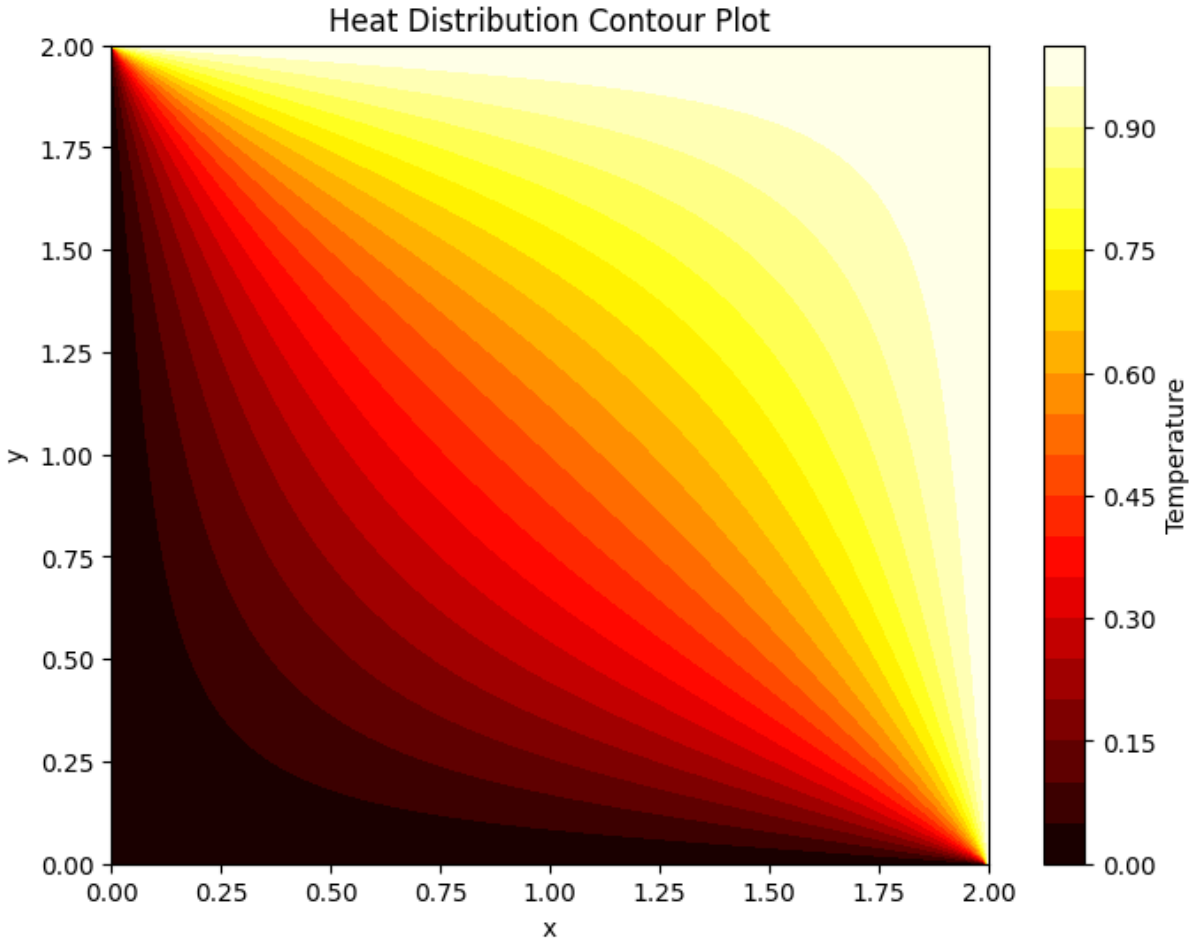

Figure 2: Temperature distribution of heat equation solved with OpenACC parallelization.

As can be seen from the figures1 and 2, the plots are identical, which indicates that the code from parallelization is just as accurate as the code without parallelization. The runtimes are as stated below.

1. 2D Heat Equation: Number of Time Steps vs Runtime

The first experiment, where the grid size was kept constant at 0.01x0.01, showed a 4-5% speedup in code runtime when using OpenACC for parallelization. The performance improvement indicates the effectiveness of optimizing the loop structure through OpenACC, enhancing the simulation's overall efficiency.

Number of Steps	No parallelization Runtime	Parallelization Runtime
100000	25.06 sec	8.99 sec
200000	50.7 sec	17.73 sec
250000	61.42 sec	22.02 sec
500000	125.03 sec	43.7 sec
750000	184.96 sec	68.57 sec
800000	199.61 sec	69.52 sec
1000000	247.45 sec	86.97 sec

Table 1: Runtimes for varying number of time steps for both the codes(constant grid size=200)

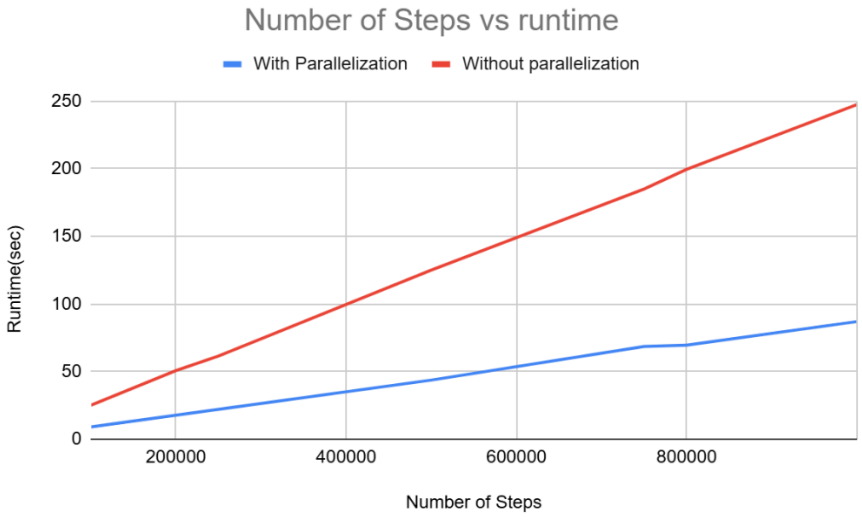

Figure 3: Number of Steps vs Runtime comparison of the code without parallelization and parallelized code.

The graph above shows how parallelization reduces the runtime of the code. On average, the code speeds up by around 65%.

Grid Size vs Runtime

Parallelization time	**No Parallelization time**	**Grid Spacing(same in both X and Y)**
0.16 sec	0.36 sec	20
0.42 sec	1.41 sec	40
1.5 sec	4.24 sec	80
2.24 sec	6.53 sec	100
4.3 sec	12.59 sec	140
5.7 sec	16.32 sec	160
8.99 sec	25.06 sec	200

Table 2: Runtime for code with parallelization and no parallelization for various grid sizes(constant number of steps=100000)

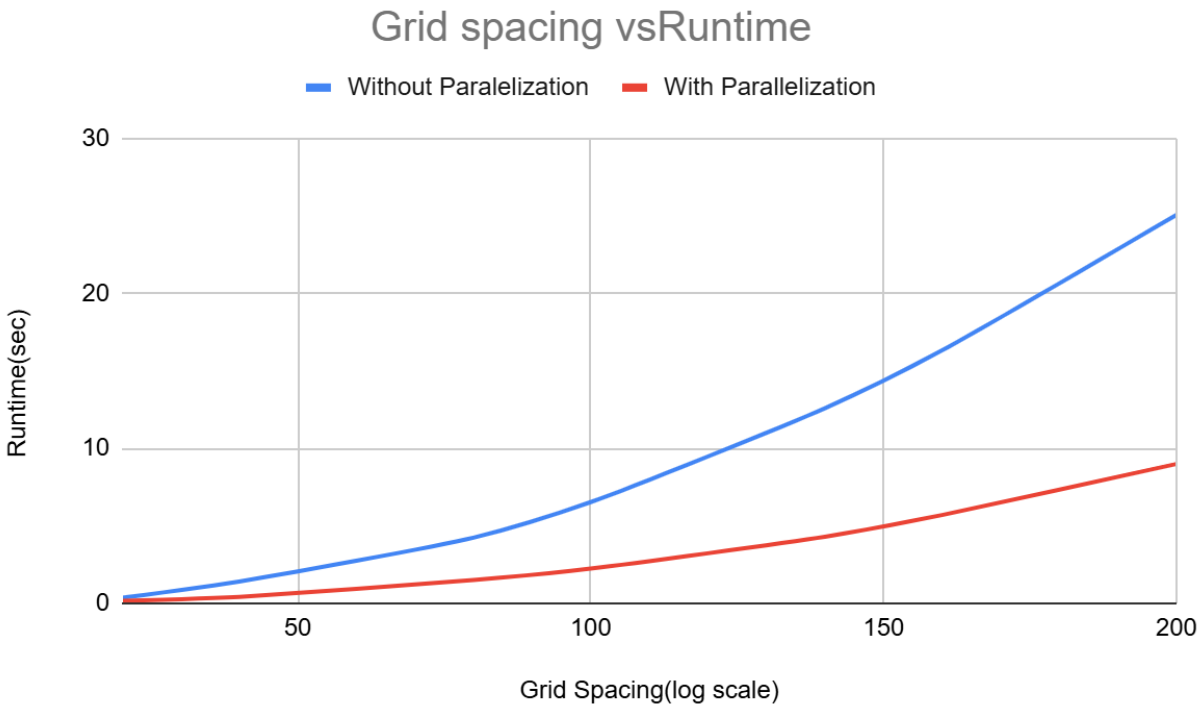

Figure 4: Graph of grid size vs runtime(sec)

2. ArtraCFD: Profiling Results

The results of the simulation are plotted in the contour plots below. The runtime profiling of the ArtraCFD solver revealed the following results. The plots of u, v and T along the grid have been shown in the figure

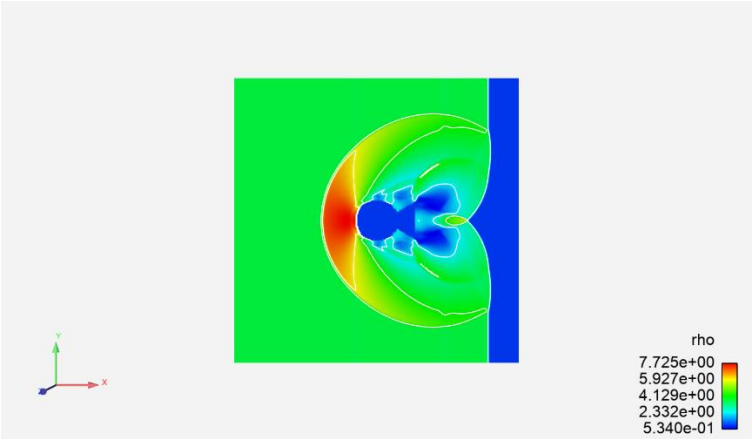

Figure 5:Contour plot showing density distribution

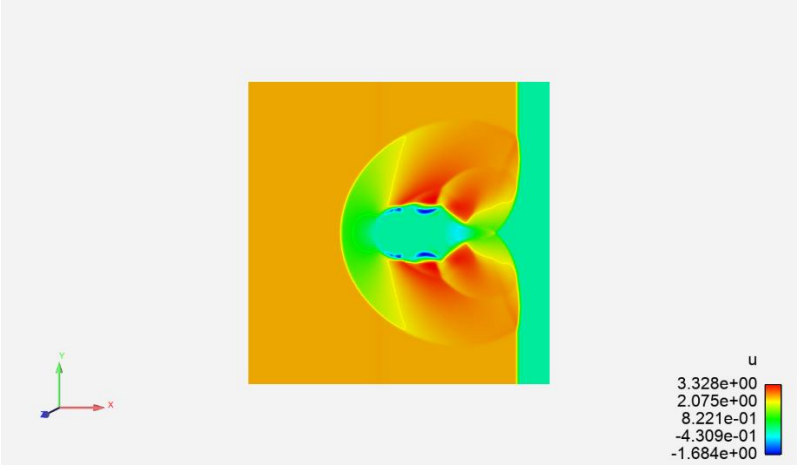

Figure 6: Contour plot showing the velocity profile along X direction(u)

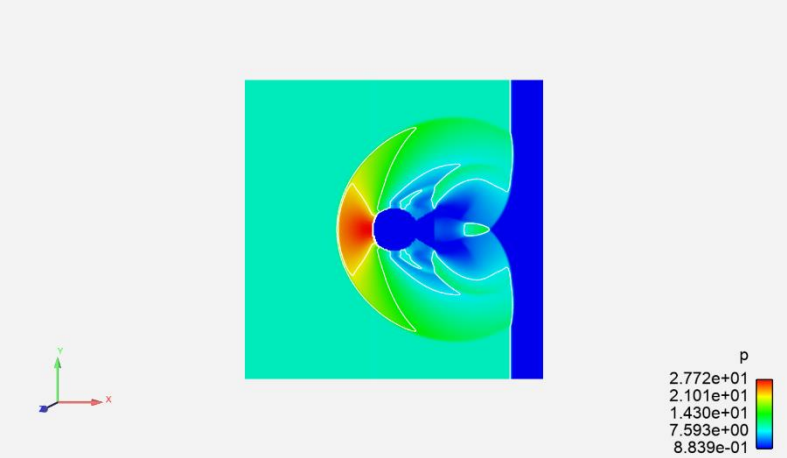

Figure 7: Contour plot showing the pressure variation

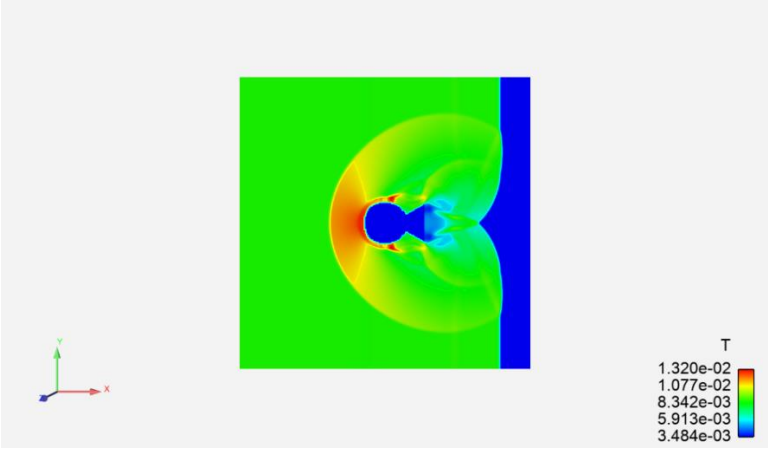

Figure 8: Contour plot showing the temperature distribution profile

The main function's most time-consuming operation was the *Solve* function, taking 77.338 seconds. Within the *Solve* function, the *EvolveSolution* function was the bottleneck, accounting for 76.527 seconds. Delving deeper into *EvolveSolution*, the *EvolveFluidDynamics* function was identified as the next performance hotspot, consuming 0.423 seconds. Within *EvolveFluidDynamics*, the *DiscretizeTime* function ran multiple times, with four instances in the sample test case, each averaging 0.96 seconds.

DiscretizeTime calls the *IntegrateTime* function, which in turn invokes *RungeKutta3*. While *IntegrateTime* and *RungeKutta3* ran for approximately 0.96 and 0.086 seconds, respectively, the longest runtime within *RungeKutta3* was taken by the *LLLU* function, consuming 0.034 seconds. Importantly, *LLLU* did not invoke additional functions during the profiling test case, leading to the conclusion that profiling could go no further.

CONCLUSIONS

In BTP 1, significant progress was made in profiling and initiating optimization efforts on both a 2D heat equation and the ArtraCFD solver, using OpenACC to accelerate computations. The first experiment, involving the heat equation, achieved a notable 65% reduction in runtime with OpenACC. This experiment demonstrated the potential of OpenACC to enhance performance through parallelization, while also revealing opportunities for more refined optimization, particularly in memory management, which will be addressed in future stages.

In the case of ArtraCFD, profiling provided detailed insights into the code's performance bottlenecks, specifically identifying the *EvolveSolution* function, and within it, *EvolveFluidDynamics*, as primary areas for optimization. Additionally, functions such as *DiscretizeTime* and *IntegrateTime* were pinpointed as major contributors to runtime, offering clear targets for parallelization in the next phase, BTP 2. By focusing optimization efforts on these identified functions, the project is well-positioned to achieve greater efficiency and performance gains. The insights obtained from profiling not only guide the upcoming optimization but also underscore the potential of OpenACC to streamline computationally intensive processes in CFD solvers.

Despite these achievements, certain limitations were noted. The profiling was conducted on a single test case, which limits the generalizability of the results across different flow conditions or

scenarios. Future work will need to address this by testing additional cases to ensure robustness and reliability. Furthermore, optimization in BTP 1 was constrained by initial challenges, such as memory management and implementation issues, which limited the full potential of speedups in the OpenACC parallelization. Addressing these limitations will be central to achieving the project's goals in BTP 2.

REFERENCES

Apurva Raj, Piru Mohan Khan, Md. Irshad Alam, Akshay Prakash, Somnath Roy,

A GPU-accelerated sharp interface immersed boundary method for versatile geometries,

Journal of Computational Physics, Volume 478, 2023, 111985, ISSN 0021-9991,

https://doi.org/10.1016/j.jcp.2023.111985

Manish Kumar, Somnath Roy, Md Sujaat Ali, An efficient immersed boundary algorithm for simulation of flows in curved and moving geometries, Computers & Fluids, Volume 129, 2016,

Pages 159-178, ISSN 0045-7930, https://doi.org/10.1016/j.compfluid.2016.02.009

Huangrui Mo, Fue-Sang Lien, Fan Zhang, and Duane S. Cronin. An immersed boundary method for solving compressible flow with arbitrarily irregular and moving geometry. International Journal for Numerical Methods in Fluids 88, no. 5 (2018): 239-263.

Mohuangrui. (n.d.). *ArtraCFD/README.md at master · mohuangrui/ArtraCFD*. GitHub. https://github.com/mohuangrui/ArtraCFD/blob/master/README.md

Introduction to OpenACC — OpenACC/CUDA for beginners. (n.d.). https://enccs.github.io/OpenACC-CUDA-beginners/1.02_openacc-introduction/