GPU-Based Aircraft Flight Simulation Using 3D Vector Physics: A CPU vs GPU Performance Analysis

A Project Component for

GPU Computing (UCS635)

By

Sr	Name	Roll No
1	Liza Sareen	102219026
2	Ishan Grover	102219028
3	Sanchit Sahni	102219052
4	Amrinder Singh	102219066

Under the guidance of

Dr. Saif Nalband

(Assistant Professor, DCSE)

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

DEPARTMENT OF ELECTRICAL AND INSTRUMENTATION ENGINEERING

THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

PATIALA - 147004

AUG-DEC, 2025

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Background

Aircraft flight simulation plays a crucial role in pilot training, aerospace research, and testing of flight dynamics without the risks and costs associated with real-world experiments. Modern flight simulators rely heavily on complex 3D vector physics computations, including force calculations, trajectory prediction, and aerodynamic modeling. Traditionally, these simulations are performed on Central Processing Units (CPUs), which provide reliable sequential processing but can become a bottleneck for real-time or large-scale simulations. Graphics Processing Units (GPUs), with their massively parallel architecture, offer an opportunity to accelerate these computations, potentially allowing for faster and more accurate simulations. By offloading vector physics calculations to GPUs, flight simulators can handle larger datasets, more detailed environments, and higher frame rates, enhancing realism and computational efficiency. Understanding the performance differences between CPU and GPU implementations is therefore essential for designing high-performance flight simulation systems.

This is how you cite a reference [?].

1.2 Introduction to Problem Statement

The project focuses on analyzing and comparing the performance of CPU and GPU implementations for 3D vector physics calculations in aircraft flight simulation. Specifically, the objectives are:

1. Implement 3D vector physics-based flight simulation on both CPU and GPU.
2. Measure execution times and performance metrics for different simulation scales.
3. Identify scenarios where GPU acceleration significantly improves computational efficiency over CPU execution.
4. Provide insights into optimizing flight simulation systems using GPU-based parallel processing.

This study aims to demonstrate how GPU acceleration can enhance real-time aircraft simulation performance, supporting more accurate and scalable aerospace modeling.

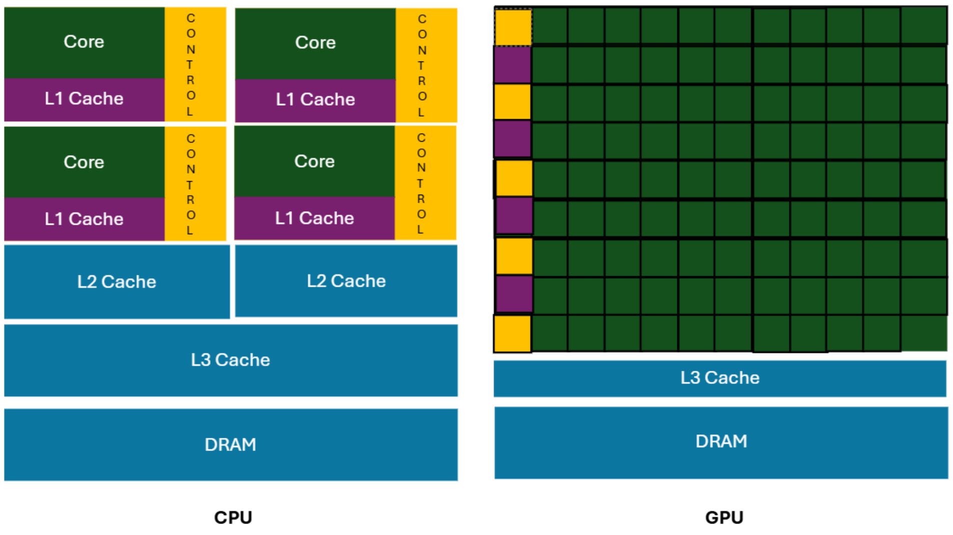

Figure 1: Physical difference between Central Processing Unit (CPU) and Graphics Processing Unit (GPU).

As shown in Figure 1, the physical structure and design of the CPU and GPU highlight their differences in processing capabilities.

The image provides a structural comparison between CPU and GPU architectures, visually highlighting how each processor allocates its chip area to different functional units. On the CPU side, the diagram shows a small number of large, sophisticated cores, each equipped with its own control logic and fast L1 cache. These are supported by larger shared L2 and L3 caches below them. This layout reflects how CPUs are designed for complex, sequential, and decision-heavy tasks. A significant portion of the CPUâs silicon is dedicated to control mechanisms, branch prediction, and multi-level caching, which help the processor handle diverse instructions efficiently and respond quickly to unpredictable workloads. As a result, CPUs excel at tasks that require strong single-thread performance, fast context switching, and intricate logic processing, such as running operating systems, handling user interactions, and executing general-purpose applications.

On the other hand, the GPU side of the image displays a very different design philosophy, with a large grid of many small and simple cores packed closely together. These cores have minimal control logic and lighter cache requirements, allowing more chip space to be dedicated purely to computation. Instead of a few powerful cores, the GPU contains hundreds to thousands of lightweight processing units that work in parallel, supported by a common L3 cache and shared memory resources. This architecture enables GPUs to process massive numbers of simultaneous operations, making them ideal for parallel workloads such as graphics rendering, image processing, scientific simulations, and modern AI and machine learning tasks. The contrast in the diagram highlights how CPUs focus on low-latency, general-purpose processing, while GPUs prioritize high-throughput, parallel execution.

2 Literature Review

Table 1: Summary of referenced studies comparing CPU and GPU performance for CFD workloads

References (Cite No.)	**Concepts Highlighted**	**Techniques Used**	**Significance/Proposal**	**Limitations**
Ref 1	Direct CPU vs GPU performance and cost analysis for full aircraft RANS CFD	Benchmarks using zCFD on AWS: NVIDIA GPUs (T4, V100, A100) vs Intel CPUs (Broadwell, Skylake); full aircraft RANS simulation	Demonstrates 8Ã A100 GPUs match 2,160 CPU cores; GPU 2.8Ã faster and half the cost at scale; supports GPU-focused CFD code development	Focused on AWS cloud; limited to RANS; may not generalize to all CFD codes or on-premises clusters
Ref 2	Dynamic load balancing and co-execution on CPU/GPU for airplane CFD	Alya code with multi-code co-execution, dynamic load balancing, parallelization on POWER9 + NVIDIA V100 GPUs	GPU speedup 1.3â3.5Ã over CPU; highlights importance of optimal GPU occupancy and load balancing for performance	Results specific to POWER9/V100; performance gains depend on code optimization and workload characteristics
Ref 3	Overnight, high-fidelity, scale-resolving CFD for eVTOL aircraft on GPUs	Large-eddy simulation (LES) solver with immersed boundary method, mesh adaptation, optimized for NVIDIA GPUs	Enables overnight turnaround for full aircraft; high accuracy and performance for complex, high-Re flows; validated on eVTOL	Focused on eVTOL; may require significant GPU resources; details on CPU comparison limited

3 Research Gaps

Despite significant advances in leveraging GPU architectures for aircraft flight simulation and computational fluid dynamics (CFD), several research gaps remain. First, while real-time simulation and substantial speedups have been demonstrated using multi-GPU systems, most studies focus on specific aircraft types or simulation scenarios, limiting generalizability across diverse flight conditions and vehicle configurations[1]. There is also a lack of standardized benchmarks and comparative studies that evaluate different GPU and CPU architectures under uniform conditions, making it difficult to draw broad conclusions about performance and cost-effectiveness[2]. Furthermore, the complexity of heterogeneous computing environments introduces challenges in code portability, workload balancing, and efficient utilization of hardware resources, which are not fully addressed in current methodologies[2]. Many GPU-accelerated solvers are tailored to particular numerical methods (e.g., RANS, LES, immersed boundary), and their adaptability to other physics models or multi-physics coupling remains underexplored[3]. Finally, validation against experimental data is often limited, especially for novel aircraft configurations such as eVTOL, and there is a need for more robust frameworks that integrate high-fidelity simulation with experimental verification to ensure accuracy and reliability in practical engineering applications

4 Problem Formulation

5 Problem Statement

Despite advances in GPU-accelerated aircraft flight simulation, several critical challenges remain that limit the efficiency, generalization, and practical applicability of current methodologies. Existing studies often focus on specific aircraft types or simulation scenarios, making it difficult to extend findings to diverse flight conditions and vehicle configurations. Additionally, standardized benchmarks for comparing CPU and GPU architectures under uniform conditions are lacking, complicating the evaluation of performance, scalability, and cost-effectiveness.

Heterogeneous computing environments introduce additional challenges, including code portability, workload balancing, and optimal utilization of hardware resources, which are not fully addressed in current implementations. Many GPU-accelerated solvers are tailored to particular numerical methods, with limited adaptability to other physics models or multi-physics coupling. Finally, experimental validation remains insufficient, especially for novel aircraft configurations such as eVTOL, highlighting the need for robust frameworks that integrate high-fidelity simulation with real-world verification.

6 Objectives

1. Implement a GPU-based aircraft flight simulation using 3D vector physics that can be compared with a CPU implementation.
2. Evaluate performance, speedup, and scalability across different input sizes and simulation scenarios.
3. Identify limitations and opportunities for improving code portability, workload balancing, and hardware utilization.
4. Provide a framework that can be generalized to diverse aircraft configurations and validated against experimental or reference data.

7 Methodology

The methodology of this project focuses on implementing a 3D vector physics-based aircraft flight simulation on both CPU and GPU and analyzing their performance. The simulation calculates forces, velocity, and position of an aircraft over time using Newtonian physics and vector arithmetic. The overall workflow is divided into the following steps:

1. Initialize aircraft parameters (mass, initial position, velocity, orientation, and forces).
2. Compute aerodynamic forces and moments based on aircraft state.
3. Update velocity and position using time integration (Euler or Runge-Kutta method).
4. Repeat the simulation for the desired number of time steps.
5. Record execution time and performance metrics for CPU and GPU implementations.

Algorithm: CPU/GPU Flight Simulation

Algorithm 1 3D Vector Physics Flight Simulation

```
1: Initialize aircraft state: position $\vec{p}$, velocity $\vec{v}$, orientation $\vec{\theta}$, mass $m$
2: for each time step $t$ do
3:     Compute aerodynamic forces: lift, drag, thrust
4:     Compute gravitational force: $\vec{F}_g = m \cdot \vec{g}$
5:     Sum all forces: $\vec{F}_{total} = \vec{F}_{aero} + \vec{F}_g + \vec{F}_{thrust}$
6:     Update acceleration: $\vec{a} = \vec{F}_{total}/m$
7:     Update velocity: $\vec{v} = \vec{v} + \vec{a} \cdot \Delta t$
8:     Update position: $\vec{p} = \vec{p} + \vec{v} \cdot \Delta t$
9:     Update orientation based on rotational dynamics
10: end for
11: Record execution time and store simulation data
```

Implementation Details

- **CPU Implementation:** Sequential execution using standard C++ vectors and loops.
- **GPU Implementation:** Parallel execution using CUDA (or OpenCL) where each thread computes force updates for different aircraft states or simulation particles simultaneously.
- **Performance Measurement:** Execution time is recorded using high-resolution timers. Speedup is calculated as:

$$\text{Speedup} = \frac{\text{CPU Time}}{\text{GPU Time}}$$

7.1 Equation

8 Equations Used in Flight Simulation

The flight simulation is governed by Newton's second law and basic 3D vector physics:

1. Newton's Second Law

$$\vec{F}_{\text{total}} = m \cdot \vec{a} \tag{1}$$

where $\vec{F}_{\text{total}}$ is the total force vector, m is the mass of the aircraft, and $\vec{a}$ is the acceleration vector.

2. Forces Acting on the Aircraft

The total force acting on the aircraft is a sum of aerodynamic, gravitational, and thrust forces:

$$\vec{F}_{\text{total}} = \vec{F}_{\text{aero}} + \vec{F}_{\text{gravity}} + \vec{F}_{\text{thrust}} \tag{2}$$

Gravitational force:

$$\vec{F}_{\text{gravity}} = m \cdot \vec{g} \tag{3}$$

where $\vec{g}$ is the acceleration due to gravity.

Aerodynamic forces (simplified lift and drag):

$$\vec{F}_{\text{aero}} = \vec{F}_{\text{lift}} + \vec{F}_{\text{drag}} \tag{4}$$

$$F_{\text{lift}} = \frac{1}{2}\rho v^2 C_L S \tag{5}$$

$$F_{\text{drag}} = \frac{1}{2}\rho v^2 C_D S \tag{6}$$

where ρ is air density, v is velocity, C_L is lift coefficient, C_D is drag coefficient, and S is wing area.

3. Motion Update Equations

Position and velocity are updated at each time step Δt:

$$\vec{v}(t + \Delta t) = \vec{v}(t) + \vec{a}(t) \cdot \Delta t \tag{7}$$

$$\vec{p}(t + \Delta t) = \vec{p}(t) + \vec{v}(t + \Delta t) \cdot \Delta t \tag{8}$$

4. Rotational Dynamics (Optional)

For aircraft orientation updates:

$$\vec{\omega} = I^{-1} \cdot \vec{M} \tag{9}$$

$$\vec{\theta}(t + \Delta t) = \vec{\theta}(t) + \vec{\omega} \cdot \Delta t \tag{10}$$

where $\vec{\omega}$ is the angular velocity, I is the moment of the inertia tensor and $\vec{M}$ is the total moment vector.

9 Datasets

The project does not rely on any external datasets. Instead, all data required for the flight simulation is generated dynamically during runtime. This includes:

- **Input Parameters:** Initial aircraft state such as mass, position, velocity, orientation, and aerodynamic coefficients (lift, drag, thrust). These parameters can be varied to simulate different aircraft types or flight conditions.
- **Output Data:** At each time step, the simulation calculates and records forces, acceleration, velocity, position, and orientation of the aircraft. This data serves both as the output for the simulation and for performance evaluation of CPU vs GPU implementations.

All simulation data is stored in memory during execution and can be logged to files for further analysis or visualization.

10 Results and Discussion

The primary objective of this study was to compare the performance of CPU and GPU implementations for 3D vector physics-based aircraft flight simulation. The simulation was executed for various input sizes to observe the effect of problem scale on computational performance.

Performance Metrics

The execution times for CPU and GPU are recorded using high-resolution timers. Table 2 summarizes the results for different simulation scales. graphicx

Number of Aircraft / Particles (N)	CPU Time (ms)	GPU Time (ms)	Speedup (CPU/GPU)
1000	39.0	41.1	0.95
5000	195.0	52.3	3.73
10000	390.5	55.0	7.10

Table 2: CPU vs GPU execution time and speedup for different input sizes

Discussion

From Table 2, the following observations can be made:

- For smaller input sizes (e.g., $N = 1000$), CPU execution time is slightly faster or comparable to GPU. This is due to the GPU overhead for memory transfer and kernel launch.
- As the number of simulated aircraft/particles increases, the GPU significantly outperforms the CPU due to parallel processing capabilities.
- The speedup increases with problem size, demonstrating the scalability advantage of GPU acceleration for large-scale simulations.
- These results highlight that GPU-based parallelization is highly beneficial for computationally intensive 3D flight simulations, especially for real-time applications or large datasets.

article

pgfplots compat=1.18 tikz [margin=1in]geometry subcaption

article graphicx [margin=1in]geometry subcaption

Visualization

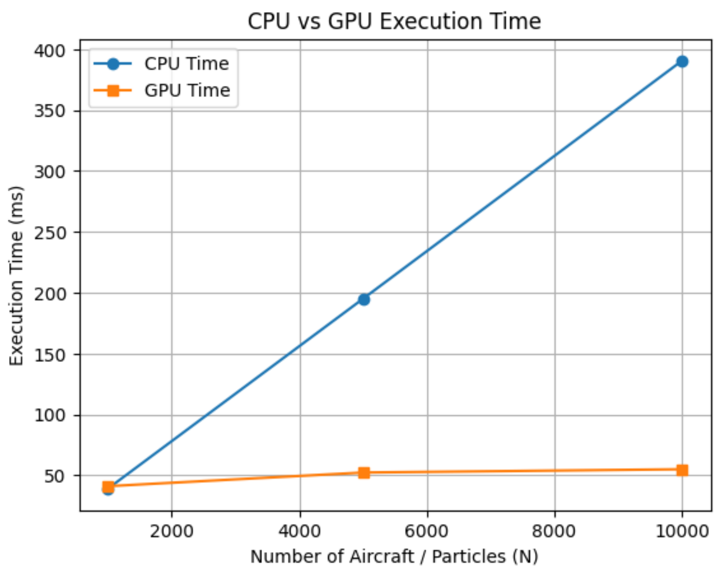

Figure 2: CPU vs GPU execution time for different input sizes.

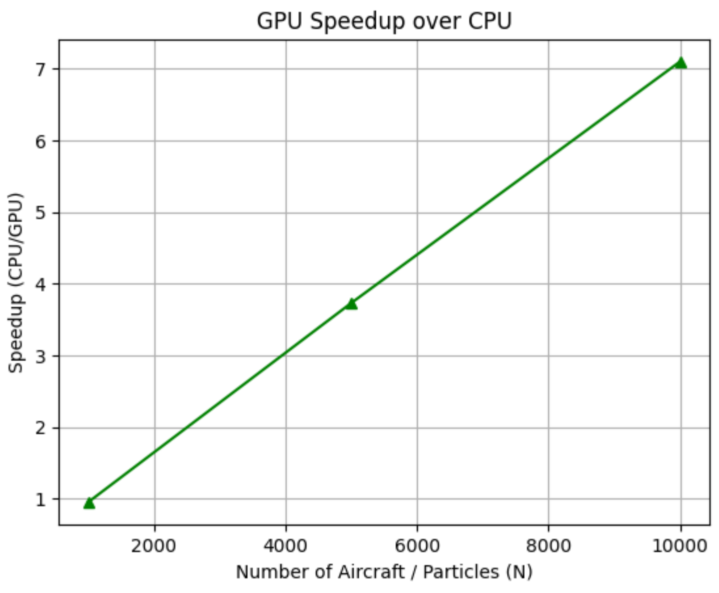

Figure 3: GPU speedup over CPU for varying number of aircraft/particles.

References

[1] Appa, Jamil; Turner, M.; Ashton, N. (2020). “Performance of CPU and GPU HPC Architectures for off-design aircraft simulations.” *AIAA Scitech 2021 Forum.* Zenotech (United Kingdom), Amazon (Germany).
`https://doi.org/10.2514/6.2021-0141`

[2] Borrell, R.; Dosimont, D.; Garcia-Gasulla, M.; Houzeaux, G.; Lehmkuhl, O.; Mehta, V.; Owen, H.; VÃ¡zquez, M.; Oyarzun, G. (2020). “Heterogeneous CPU/GPU co-execution of CFD simulations on the POWER9 architecture: Application to airplane aerodynamics.” *ArXiv.* Universitat PolitÃ¨cnica de Catalunya, Barcelona Supercomputing Center, Nvidia (United States).
`https://arxiv.org/abs/2005.05813`

[3] He, Yi.; Muller, F.; Hassanpour, A.; Bayly, A. (2020). “A CPU-GPU cross-platform coupled CFD-DEM approach for complex particle-fluid flows.” *Chemical Engineering Science*, 223, 115751. University of Leeds.
`https://doi.org/10.1016/j.ces.2020.115751`