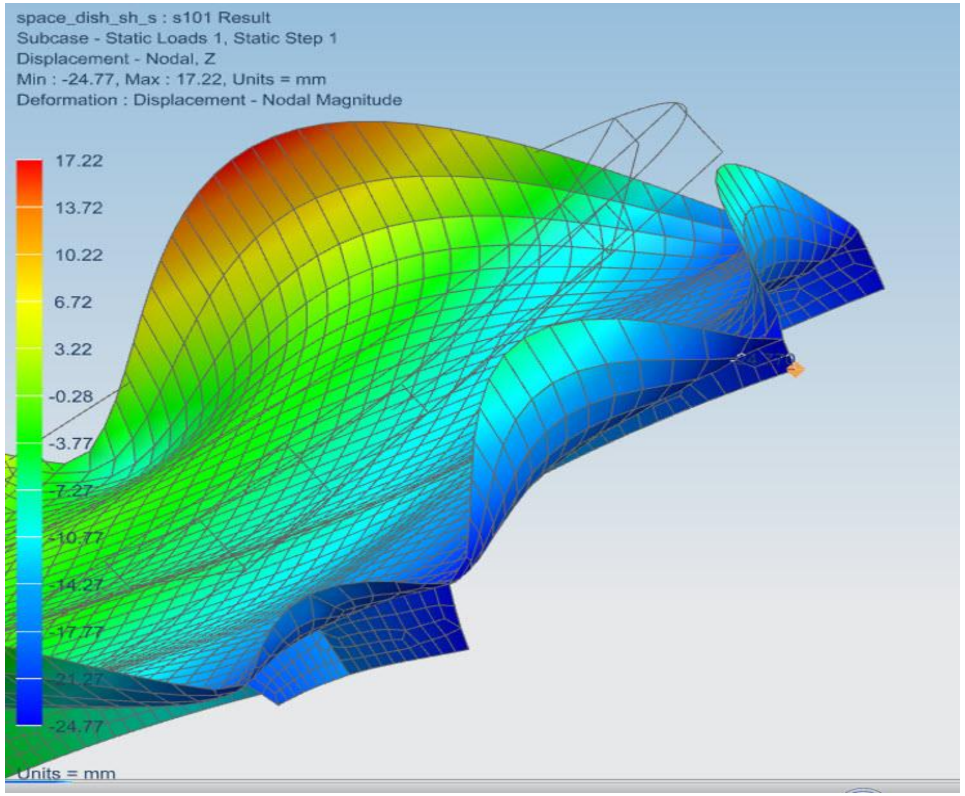
DSC 520-01: High Performance Scientific Computing

Parallelizing Particle Diffusion in 3D Space Using Monte Carlo Simulations

Presented by

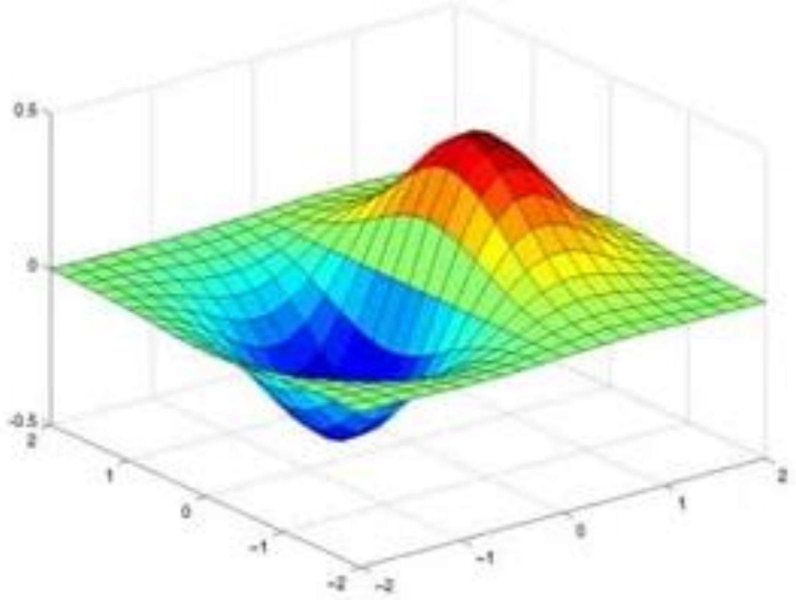
Trina Xavier

(02102403)



Introduction

- **Monte Carlo Simulations**: Monte Carlo simulation is a numerical method to approximate the behavior of systems governed by stochastic or probabilistic processes, like particle diffusion in 3D space.
- **Diffusion Equation**: The **diffusion equation** is a partial differential equation (PDE) that describes how a quantity, such as heat, particles, or chemicals, spreads over time due to random motion.
- **Monte Carlo for PDEs**: By sampling probability distributions, Monte Carlo efficiently approximates PDE solutions where analytical methods fail.
- **Practical Applications**: Widely used in engineering, healthcare, and finance, such as optimizing bridge safety, medical imaging, and financial risk assessment.



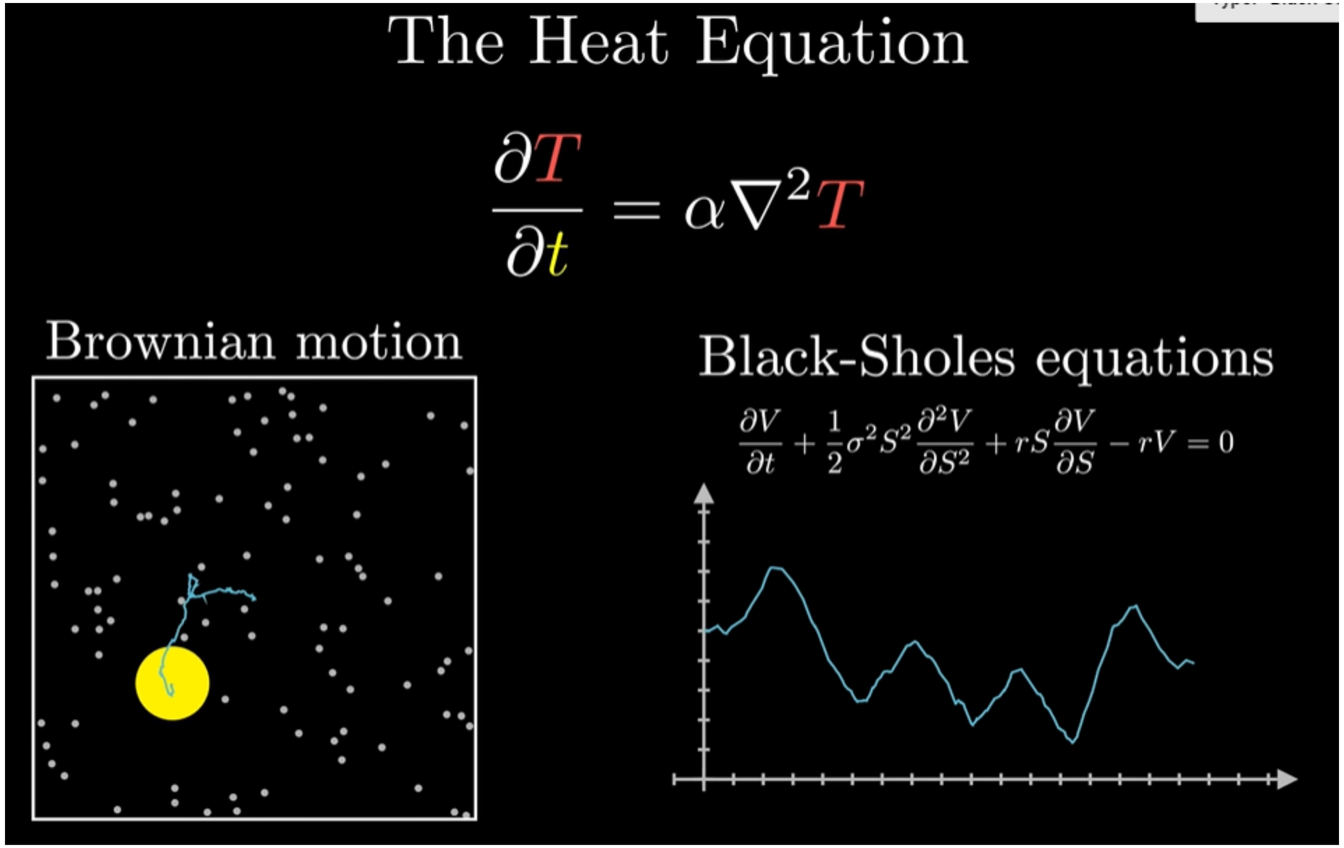
Project Overview

- **Objective:**

✓ Solve the **3D diffusion equation** using Monte Carlo simulations.

✓ Compare results with the **analytical solution** to validate accuracy.

✓ Leverage **GPU acceleration** for efficient large-scale simulations.



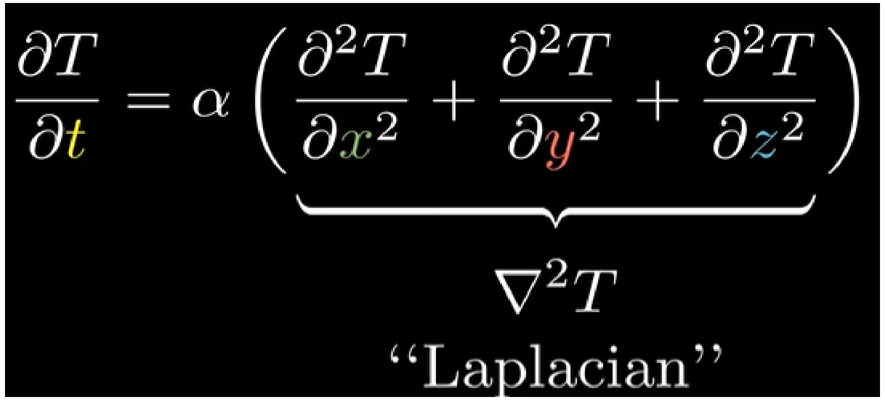
3D Diffusion Equation

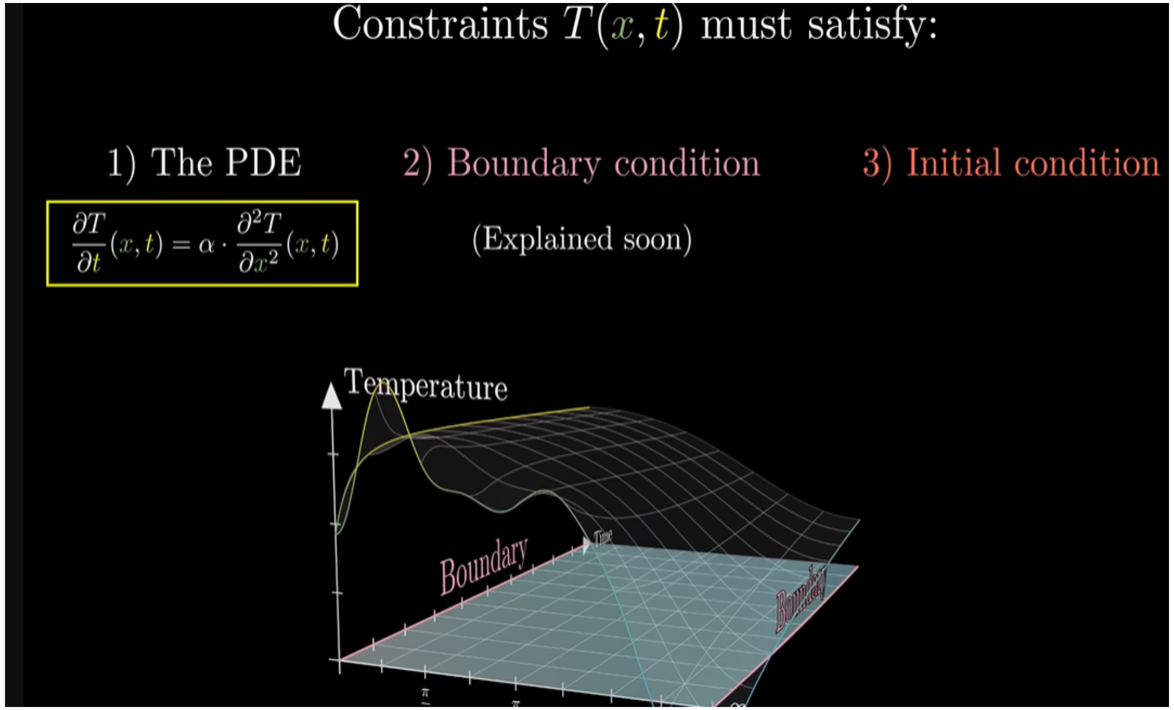
- **Mathematical Formulation**: ∂u∂t=D∇2u

1) u(x,y,z,t): Concentration at time t and position (x,y,z).
2) D: Diffusion coefficient (controls diffusion rate).
3) ∇2u: Laplacian operator in 3D.

- **Analytical Solution** for point-source initial condition:
- u(x,y,z,t)=1(4πDt)3/2exp(−x2+y2+z24Dt)

Gaussian-like distribution spreading over time.





Monte Carlo Simulation

- Monte Carlo simulation is **numerically approximating the solution** to the **3D diffusion equation** by simulating the random motion of particles (random walks) over time. This **stochastic method** allows us to model the **physical diffusion process** computationally.
- **Represents Diffusion as Random Walks**:

➢ Each particle represents a "unit" of concentration or mass.

➢ At each time step, the particles take random steps in the x, y, and z directions, with the step size proportional to the square root of two times D times the time step.

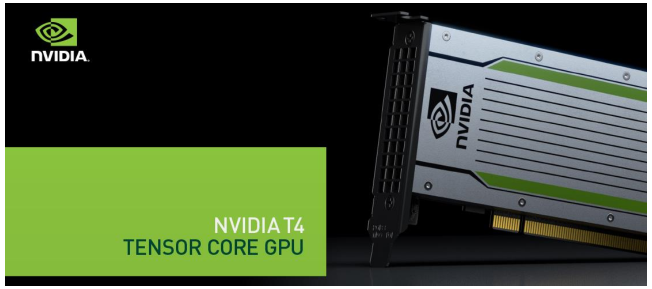
- **Approximates the Density Distribution**:

➢ By simulating the movement of a large number of particles, we compute the density at each grid cell by counting the number of particles in that cell

➢ This particle-based density distribution serves as an approximation to the solution of the 3D diffusion equation.

Parameter	Value
Grid Size	101
Number of Particles	10^7 (10 million)
Time Steps	1000
Diffusion Coefficient D	1.0
Time Step dt	0.1
Simulation Domain	3D Cartesian Grid
Computation	GPU-accelerated (CuPy)

Implementation Details



- **Tools**: Python, CuPy (GPU computations), Matplotlib (visualization)
- **Grid Size**: 101
- **Particles**: 10^7 (10 million)
- **Time Steps**: 1000
- **Diffusion Coefficient:** D=1.0
- **Platform:**

  GPU: NVIDIA Tesla T4 (Google Colab)

  CPU: Intel i5-1135G7 (2.42 GHz, 4 cores)

Platform	Execution Mode	Details
CPU (Intel i5-1135G7)	**Serial**	Single-thread execution with Python.
	Parallel (Threads)	Multi-threading in Python (multi-processing, threading).
GPU (NVIDIA Tesla T4)	**Serial**	Single CUDA thread processes particle diffusion.
	Parallel (CuPy)	Thousands of CUDA threads accelerate diffusion.

Parallelization for Speedup and Accuracy in Monte Carlo Simulation

Parallelization was implemented for **speedup** and improved **accuracy** when dealing with large-scale stochastic processes:

1. **CPU Parallelization**:
    - Used Python’s multiprocessing library to split the total workload (particles) across multiple CPU cores.
    - Each core independently simulated a subset of particles performing **random walks**, reducing execution time compared to a sequential approach.

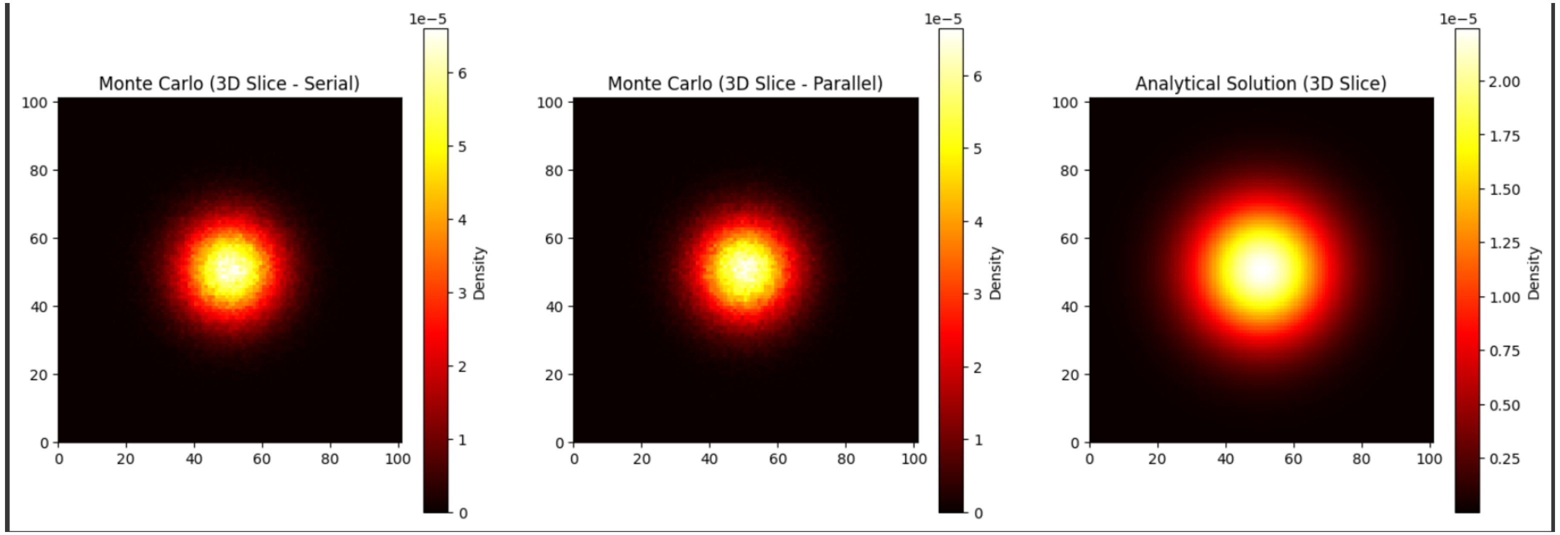
2. **GPU Parallelization**:
    - Leveraged the **CuPy library** to perform parallel computations on thousands of GPU threads.
    - Particle positions were updated simultaneously using **vectorized operations**, eliminating sequential loops.

Solving the 3D Diffusion Equation on CPU

Execution Time:
•Serial Execution: **1080.75 seconds**
•Parallel Execution: **869.95 seconds**

Parallelization Benefits:
•CPU parallelization reduced simulation time by 24%.

Limitations:
•Speedup is limited on CPUs for large-scale simulations.




```
Normalization of Analytical Solution: 0.9989360910365087
Normalization of Monte Carlo Solution (Serial): 0.9999999999999999
Normalization of Monte Carlo Solution (Parallel): 0.9999999999999996
Mean Squared Error (Monte Carlo vs Analytical) - Serial: 5.89464362543277e-12
Mean Squared Error (Monte Carlo vs Analytical) - Parallel: 5.889227937947 1064e-12
```

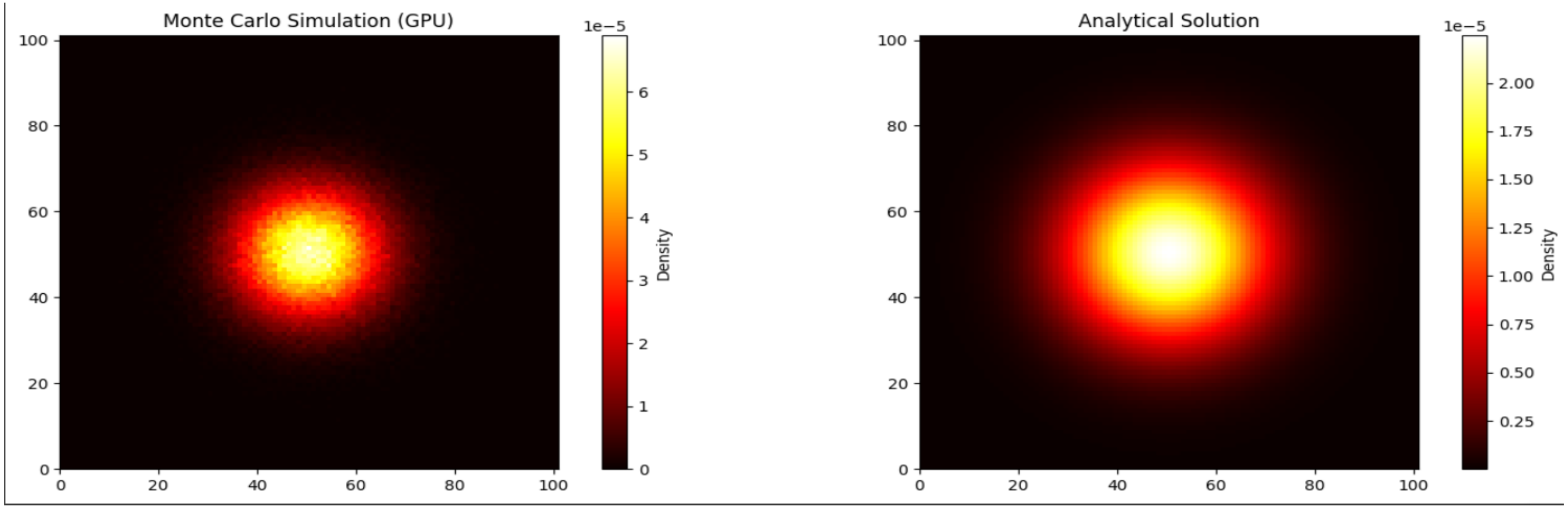
GPU Execution – Overview

•**Normalization**:

•Analytical Solution: **0.9989**
•Monte Carlo Solution (Parallel GPU): **1.000**

•**Mean Squared Error (MSE)**:
•Parallel GPU: 5.88×10^-12

Execution Type	Execution Time	Speedup
Serial GPU Execution	218.79 seconds	Reference
Parallel GPU Execution	22.35 seconds	9.8x



Results:

Execution Mode	Time (seconds)	Speedup
Serial CPU	1080.75	1.0x
Parallel CPU	869.95	1.24x
Serial GPU	218.79	4.9x
Parallel GPU	22.35	49x

Accuracy Validation:-

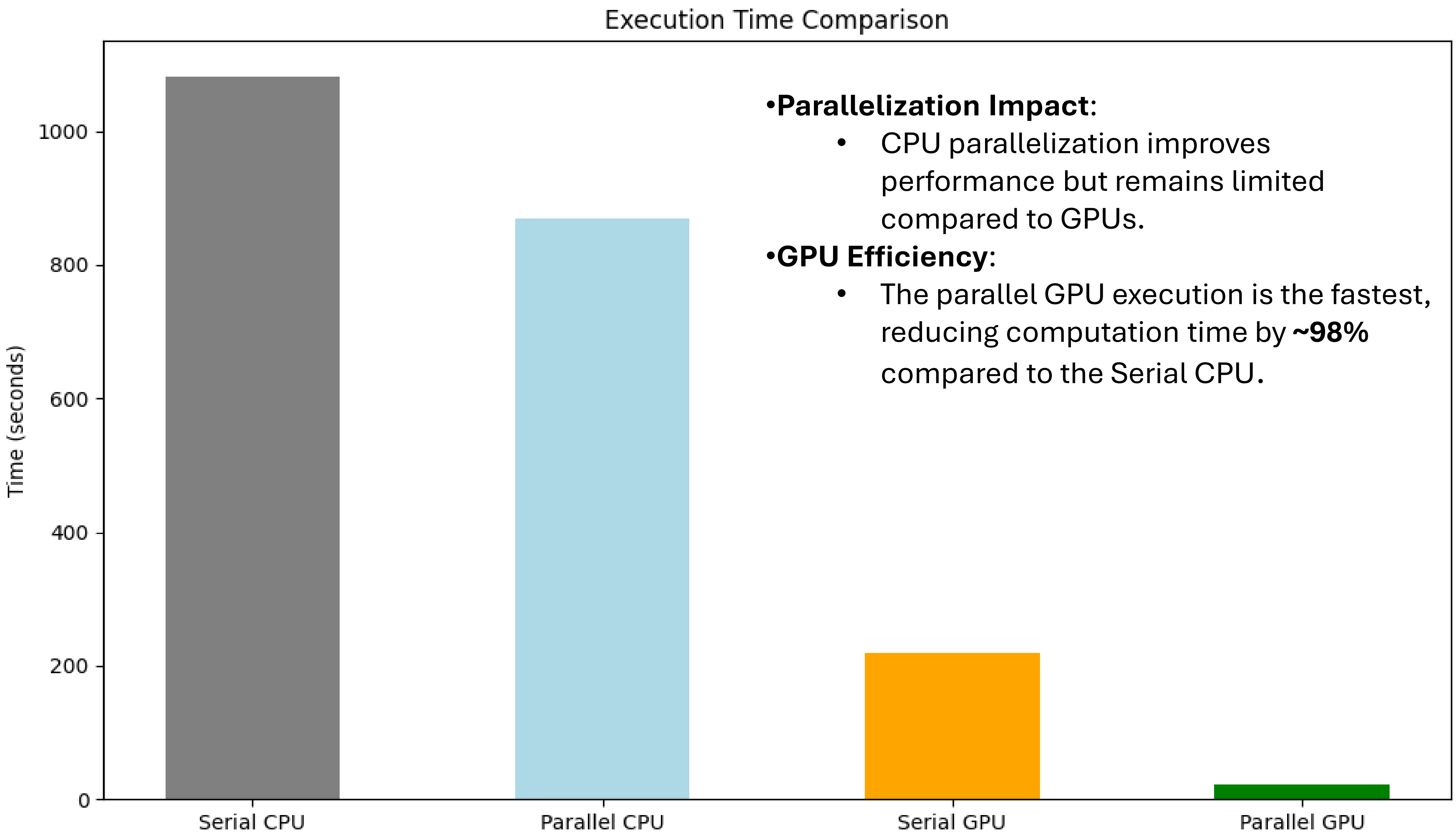
Normalization:

•Analytical Solution: **0.9989**
•Monte Carlo Solution (GPU): **0.9999**

Mean Squared Error (MSE):
•Serial Execution: 5.89×10^{-12}
•Parallel GPU: 5.88× 10^{-12}

Inference:
•The Monte Carlo approximation closely matches the analytical solution, confirming its accuracy.

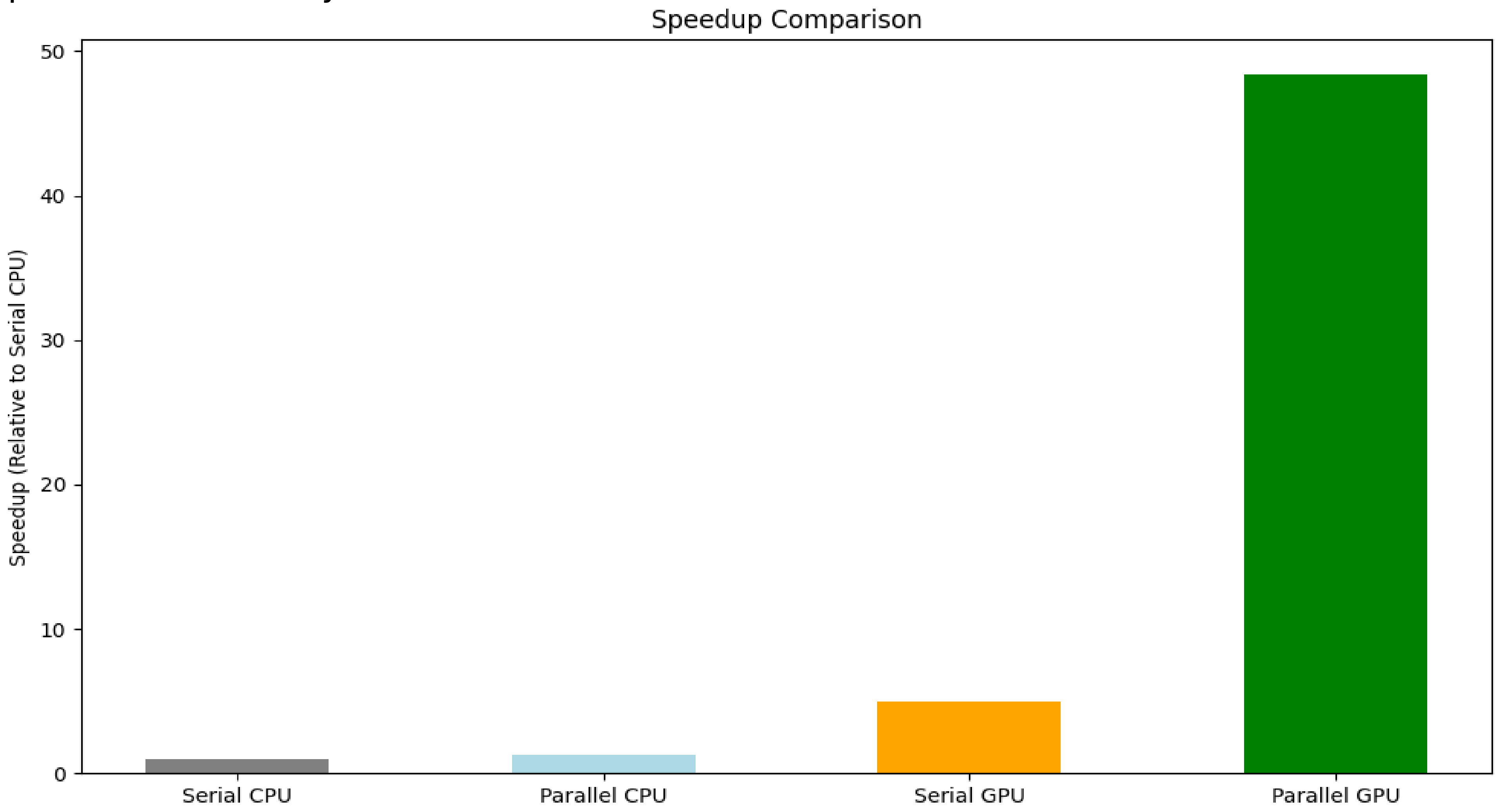
Results :



- **Parallelization Impact**:
  - CPU parallelization improves performance but remains limited compared to GPUs.
- **GPU Efficiency**:
  - The parallel GPU execution is the fastest, reducing computation time by **~98%** compared to the Serial CPU.

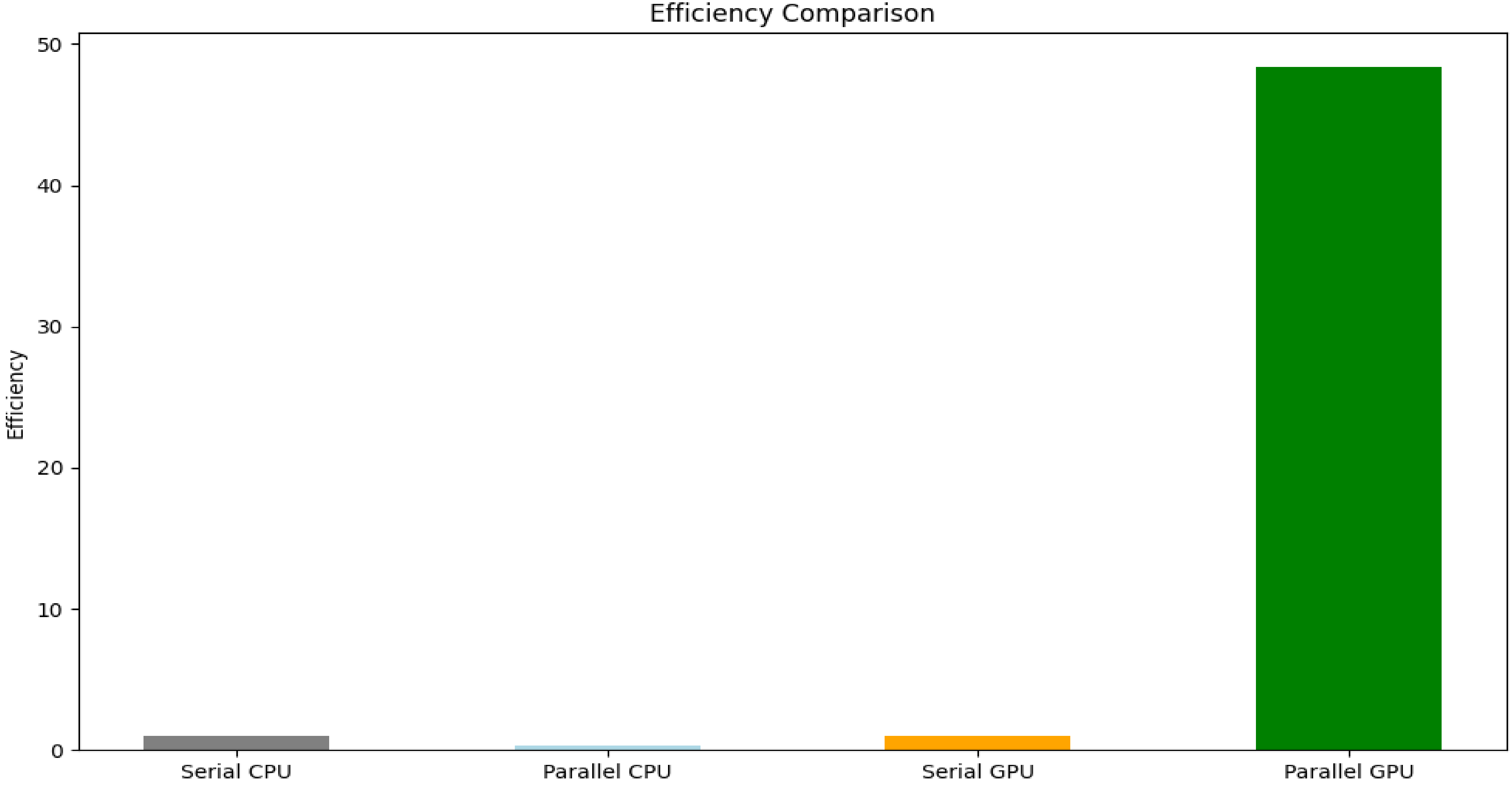
Speedup Comparison:

GPU Parallelization achieves a ~49x speedup over Serial CPU, demonstrating exceptional computational efficiency.



Efficiency Comparison:

Parallel GPU execution achieves the highest efficiency, demonstrating its ability to utilize thousands of threads for massive parallelism



Future Scope and Potential Improvements

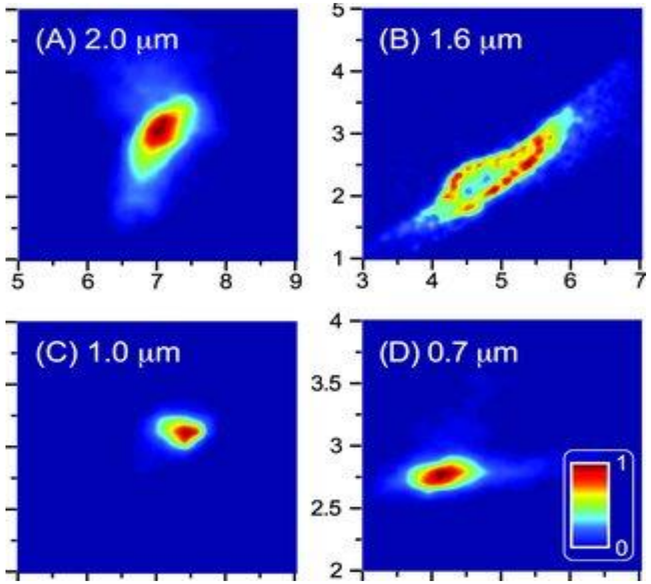
Scalability:
Increase the number of particles (e.g., 10^8) and expand grid resolution for higher precision.

Complex Geometries:
Simulate diffusion in irregular or constrained domains (e.g., porous materials, biological tissues).

Optimization:
Improve GPU performance using advanced techniques like **CuPy JIT kernels** or **mixed precision computing**.

Comparative Methods:
Compare Monte Carlo results with **finite difference** or **finite element methods** to analyze efficiency and accuracy.

Conclusion



- Successfully solved the **3D particle diffusion equation** using **Monte Carlo simulations** to approximate the solution.
- Reduced simulation time from over 1000 seconds on a serial CPU to under 25 seconds on a GPU, demonstrating computational feasibility.
- Monte Carlo methods effectively simulate **particle motion** governed by the diffusion equation, especially where analytical solutions are complex or impractical.
- The method can be extended to model **complex geometries**, higher-dimensional systems, or real-world applications like **pollutant dispersion** and **biological transport**

References:

- Overview and derivation of the diffusion equation: Diffusion Equation - Wikipedia
- Introduction to Monte Carlo methods for partial differential equations: Monte Carlo Method - Wikipedia
- Documentation and tutorials for GPU-accelerated libraries: CuPy Official Documentation
- Multiprocessing module for CPU parallelization: Python Multiprocessing - Official Documentation
- Gaussian solution for the 3D diffusion equation: Gaussian Function - Properties and Applications
- A practical guide to solving PDEs with Python: Solving PDEs with Python - Practical Overview
- Applications of Monte Carlo methods in scientific computing: Monte Carlo Simulations in Science - ResearchGate

Thank You!