# Parallel Computing Assignment Report

## Overview

This project implements and benchmarks 1D/2D matrix addition, matrix multiplication, and Laplace solver using CUDA, OpenCL, and CPU (single-threaded and OpenMP) approaches. The goal is to compare performance and correctness across platforms, and to explore optimizations such as shared/local memory and multi-device computation.

## Hardware Used

- CPU: [Fill in your CPU model, e.g., Intel Core i7-9700K]  
- GPU: [Fill in your GPU model, e.g., NVIDIA RTX 3060]  
- RAM: [Fill in your RAM, e.g., 16GB DDR4]  
- OS: Linux

## Methodology

- All implementations use C++ for host code, with CUDA and OpenCL for GPU kernels.  
- Each algorithm is run on large matrices (e.g., 1024x1024) for fair benchmarking.  
- Correctness is checked by comparing the sum of differences between CPU and GPU results (should be zero or within tolerance).  
- Performance is measured in milliseconds using a high-resolution timer.  
- Plots are generated using Python/matplotlib.

## Results

### Matrix Addition (1D/2D)

[Insert matrix\_addition\_performance.png here]

### Matrix Multiplication

[Insert matrix\_multiplication\_performance.png here]

### Laplace Solver

[Insert laplace\_solver\_performance.png here]

## Observations

- Speedup: GPU implementations (CUDA/OpenCL) show significant speedup over CPU, especially for large matrices. OpenMP provides a good boost on multi-core CPUs.  
- Correctness: All GPU results match CPU results within a small tolerance (checked by sum of differences and per-index checks).  
- Workgroup/Block Size: 16x16 is chosen for CUDA/OpenCL as it balances occupancy and memory access efficiency.  
- Shared/Local Memory: Using shared/local memory in Laplace solver and matrix multiplication further improves GPU performance.  
- Multi-device (OpenCL): Heterogeneous computing is possible and can further reduce runtime if multiple devices are available.

## How to Run

See the main README.md for build and run instructions. All results and plots are saved in the results/ directory.

## Conclusion

This assignment demonstrates the power of parallel computing using GPUs and multi-core CPUs. CUDA and OpenCL both provide substantial speedups, and further optimizations (shared/local memory, multi-device) can push performance even higher.