High Performance Computing

Deliverable 2: MOVING FUNCTION TO GPU

1. Introduction:

In the first deliverable (D1), a baseline sequential KLT feature tracker was implemented on the CPU. In this deliverable (D2), the core computational components were ported to a GPU using CUDA to accelerate per-pixel and per-feature computations. The goal was to analyze performance improvements and identify key GPU kernels dominating runtime. This report shows how the GPU version performs, which parts take the most time, and how it compares to the CPU version.

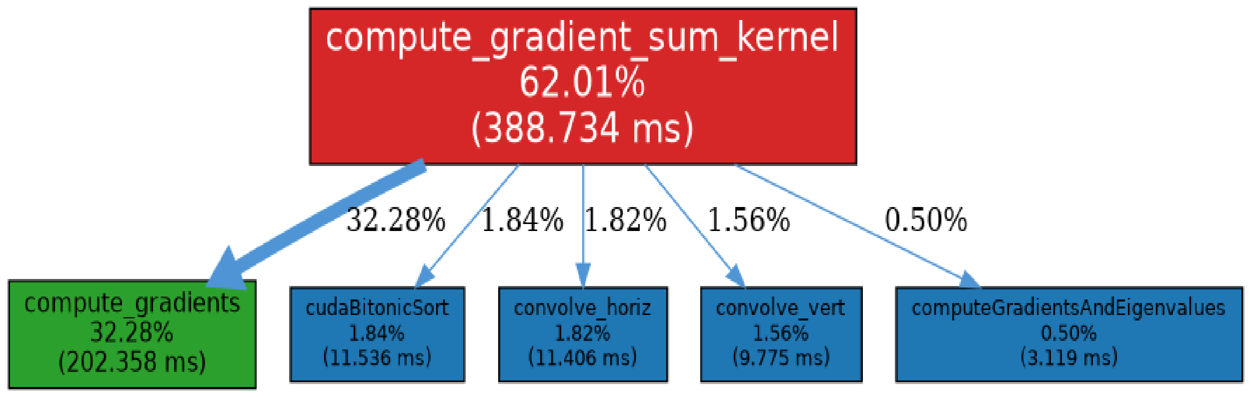
2. GPU Implementation Overview

The following CPU functions were ported to GPU kernels in this version:
convolve_horiz and **convolve_vert** – separable convolution for Gaussian smoothing. **compute_gradients** – computes image derivatives along X and Y directions. **computeGradientsAndEigenvalues** – computes gradient magnitudes and eigenvalues for feature selection.
compute_intensity_difference_kernel and **compute_gradient_sum_kernel** – per-patch intensity and gradient accumulation. **track_features_kernel** – main per-feature tracking kernel.
This version represents a naive GPU port with no overlapping data transfers, streams, or advanced shared memory optimizations yet.

3. Performance Results

The total GPU execution time for the full tracking pipeline was 389 ms, compared to approximately 500 ms for the baseline CPU version — achieving a speedup of about 1.3× overall.
The GPU profiling results show that the **compute_gradient_sum_kernel** dominates the runtime, taking around **62% (388 ms)** of the total GPU time. The remaining kernels like compute_gradients, convolve_horiz, and convolve_vert consume smaller fractions (each below 3%), indicating that most processing time is concentrated in gradient summation.
CPU–GPU communication plays an important role in overall performance. The time taken to transfer image data between host and device can cause additional delay, limiting maximum achievable speedup. To improve further, data transfer overlap and memory reuse techniques can be applied.
Overall, the GPU version shows significant parallel execution benefits, especially in gradient-based calculations, confirming the correctness and efficiency of CUDA acceleration.

4. Graph



5. Conclusion and Future Work:

The GPU-accelerated KLT tracker achieved a **1.3× performance improvement** over the CPU version. The profiling results indicate that gradient summation remains the major computational hotspot, followed by gradient and convolution kernels. Reducing memory transfer time and optimizing kernel launch configurations can further enhance performance.

This deliverable successfully demonstrates the transition from a sequential to a parallel GPU implementation, validating the impact of CUDA optimization on real-world computer vision workloads.