

CS4110 - High Performance Computing with GPUs



Deliverable 03:

KLT Feature Tracking

(Optimized GPU Implementation)

Presented By:

Ali Aan Khowaja [23I-0708]

Wajih-Ur-Raza Asif [23I-0819]

Ismail Sheikh [23I-2524]

Submitted to:

Dr. Imran Ashraf

National University of Computing & Emerging Sciences

FAST, Islamabad

30th October, 2025

Abstract

This report details the performance results of our optimized GPU implementation (V3) for the Kanade-Lucas-Tomasi (KLT) feature tracker. Building upon the naive GPU version (V2), we focused optimizations on the convolution operations (`_convolveImageHoriz` and `_convolveImageVert`), which were identified as the dominant hotspots in larger datasets during V2.

Optimizations Applied

- Tuned launch configurations for improved occupancy
- Communication optimizations for reduced host-device data transfers
- Memory hierarchy improvements using shared memory for stencil computing for minimized global memory accesses
- Reducing memory allocation overhead by using a single `cudaMalloc` and `cudaFree` for all frames instead of per-frame allocations
- Using constant memory for convolution kernels instead of global memory for faster access

Testing Results

Testing was conducted on the Modest Dataset and Jumbo Dataset, detailed in the table below.

Dataset Name	Number of Frames	Frame Resolution	Size per Frame
Modest Dataset	250	848x478	396 KB
Jumbo Dataset	250	3840x2160	8 MB

Table 1: Datasets used for testing

Note: These correspond to the d1 and d2 datasets from previous reports, renamed for clarity.

For accurate benchmarking, timings were measured specifically for the offloaded GPU convolution segments during full application runs. These timings exclude high-intensive write operations performed by the CPU, which were not part of the parallelized workload. The results compare V3 (optimized GPU) against V1 (purely CPU) and V2 (naive GPU), focusing on example 3 as the most representative workload. All timings represent averages over multiple executions where applicable, to isolate the impact of optimizations.

Performance results on Jumbo Dataset

On the Jumbo Dataset, the optimized GPU (V3) achieved an average timing of approximately **15.388 seconds** for the offloaded convolutions, compared to the CPU’s (V1) total execution time of **396.4 seconds** for the same dataset. This demonstrates a substantial speedup of about **24.47x** for the parallelized components, primarily due to the communication optimizations.

This represents a significant improvement over the naive GPU implementation (V2), which took approximately **35.986 seconds** on the same Jumbo Dataset, yielding roughly a **10x increase** from V2 to V3

through targeted optimizations. These results highlight the GPU’s efficiency for large-scale data, where overheads are minimized and parallelism is most effective.

Performance on Modest Dataset: Multiple Kernel Launch Configurations

For the Modest Dataset, we evaluated different launch configurations to optimize thread utilization and occupancy:

Kernel Launch Config	Total Execution Time
16x16	6.923 seconds
32x32	7.118 seconds
32x8	6.955 seconds
32x16	6.937 seconds
16x32	6.957 seconds

Table 2: Kernel launch configuration performance on Modest Dataset (V3)

The **16x16** configuration performed best overall, with minimal variation across options, indicating a good balance between occupancy and thread efficiency.

Conclusion

This compares to the CPU’s total execution time of 396 seconds for the dataset. The **24x speedup** for the convoluted portions stems largely from communication optimizations and shared memory stencils, which efficiently handled the large frame sizes by reducing memory traffic.

The V3 optimizations markedly improved performance on larger datasets, where convolutions dominate. The combination of tuned launch configurations, reduced communication, and shared memory for stencils scales well with data size, making GPU ideal for high-resolution workloads. For smaller datasets, overheads may still favor CPU, but Modest Dataset and Jumbo Dataset results affirm the value of these techniques.

GitHub Repo Link:

<https://github.com/aliamkhowaja/KLT-Using-CUDA>