CS4110 - High Performance Computing with GPUs

Deliverable 03:

KLT Feature Tracking

(Naive 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

17 October, 2025

1 Executive Summary

This report presents the results of the GPU offloading implementation for the Kanade-Lucas-Tomasi (KLT) feature tracker. The primary functions offloaded to the GPU were the convolution operations (_convolveImageHoriz and _convolveImageVert), as well as the feature tracking functions (_computeGradientSum and _computeIntensityDifference). The testing was conducted on three datasets: the provided dataset (d0), and two larger datasets we created (d1 and d2). The results reveal significant differences in performance based on the dataset size, with the GPU performing better on larger datasets and worse on smaller datasets due to overhead.

Datasets Used:

- d0: 10 frames, 76 KB per frame
- d1: 250 frames, 396 KB per frame
- d2: 250 frames, 8 MB per frame

For each dataset, we compared the performance of the CPU and GPU versions of the KLT tracker. The tests were conducted using the same codebase, with the GPU version using CUDA kernels for the functions previously identified for offloading.

Heads Up: After offloading, we found our initial hotspots were not accurate for larger datasets. The tracking functions (_computeGradientSum and _computeIntensityDifference) became insignificant as dataset size increased, while convolutions took most of the time.

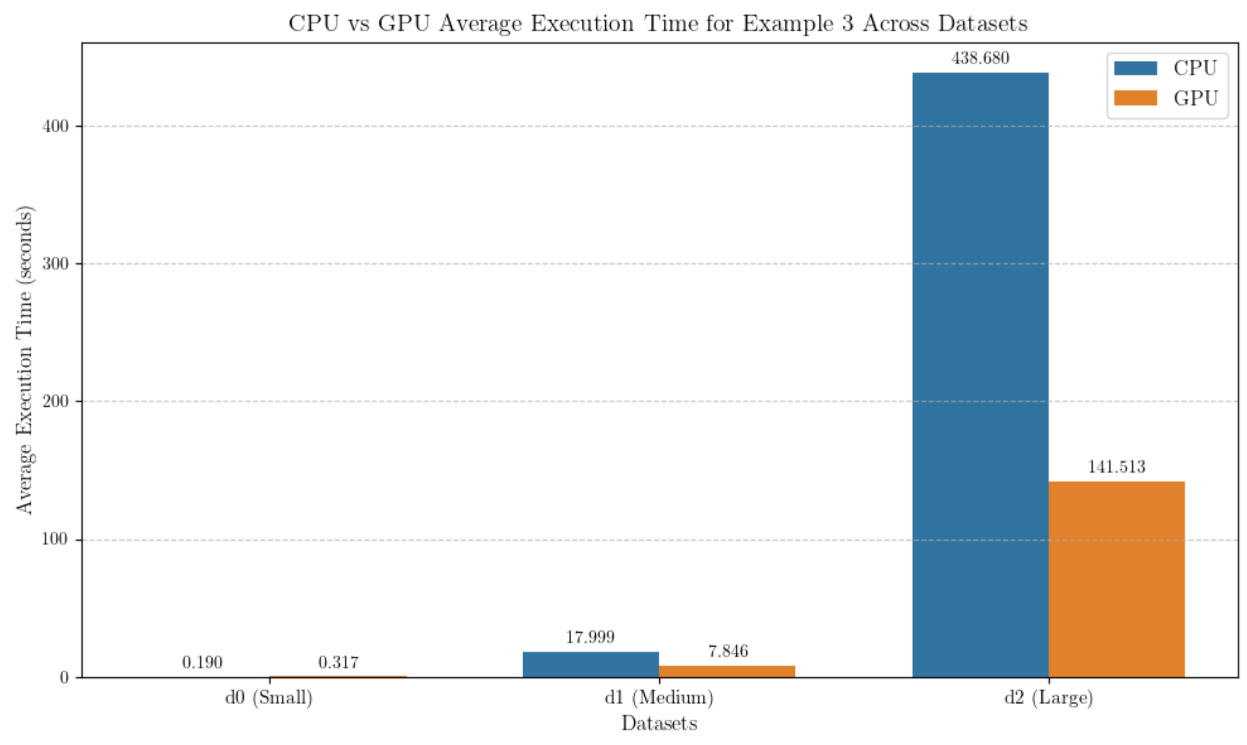
2 Implementation Details

We created CUDA kernels for the functions:

- **Convolution:** Horizontal and vertical kernels where threads compute pixels. The grid matches the image size.
- **Intensity difference:** Kernel parallelizes window interpolations.
- **Gradient sum:** Kernel handles window reductions in parallel.

3 Results

The detailed timings for each example are shown in the `src/v2/findings.md` file. Here we will demonstrate our findings using only example 3 because it is the most meaningful example.



3.1 Dataset d0 (10 frames, 76 KB per frame)

For dataset d0, which is relatively small, the GPU performed worse than the CPU. The time spent on transferring data to and from the GPU added overhead that outweighed the parallelization benefits, leading to slower performance compared to the CPU.

Average Timings:

- CPU (Example 3): 0.190s
- GPU (Example 3): 0.317s

The GPU version took significantly longer than the CPU version due to the overhead of data transfer and kernel execution setup. Since the dataset is small, the parallelization benefits provided by the GPU do not outweigh the overhead, making the CPU a better choice for this dataset.

3.2 Dataset d1 (250 frames, 396 KB per frame)

For dataset d1, the GPU showed a noticeable improvement in performance, particularly for the larger functions, as parallelization was able to reduce the execution time substantially.

Average Timings:

- CPU (Example 3): 17.999s
- GPU (Example 3): 7.846s

The GPU version performed better than the CPU, with a 2.3x speedup. The larger dataset allowed the GPU to take full advantage of parallelization, significantly reducing the execution time. This improvement is due to the ability of the GPU to handle multiple operations simultaneously, especially for functions like image convolution and feature tracking.

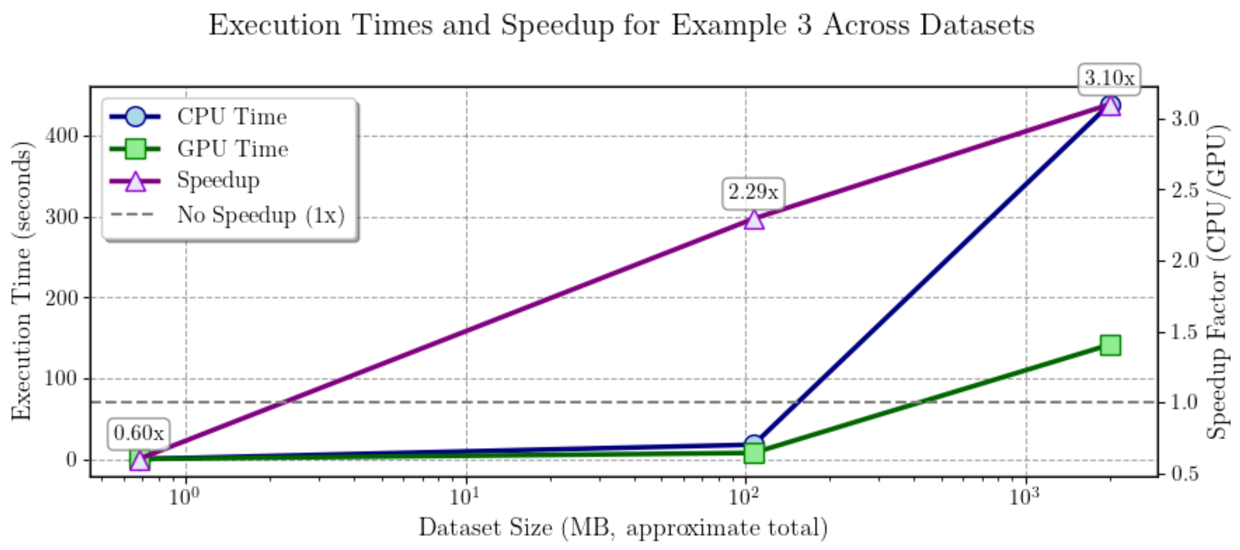
3.3 Dataset d2 (250 frames, 8 MB per frame)

For the largest dataset, dataset d2, the GPU showed the most significant improvement. The large size of the dataset allowed the GPU to fully leverage its parallel processing capabilities, resulting in a substantial reduction in execution time.

Average Timings:

- CPU (Example 3): 438.680s
- GPU (Example 3): 141.513s

The GPU showed a 3x speedup on dataset d2, completing the task in 141.513s compared to 438.680s on the CPU. The large dataset enabled the GPU to outperform the CPU by processing many operations in parallel, drastically reducing the execution time. This demonstrates the GPU's ability to handle large-scale computations more efficiently.



4 Conclusion

The results highlight the varying performance of the GPU depending on the dataset size. On smaller datasets like d0, the GPU suffers from overheads, resulting in slower performance compared to the CPU. However, as the dataset size increases, the GPU provides significant speedup. For dataset d1, we observed a 2.3x speedup, and for dataset d2, the GPU provided a 3x speedup. Based on these findings, the GPU is recommended for larger datasets where the overheads are less impactful, and the parallelization benefits can be fully utilized. For smaller datasets, it is better to stick with the CPU.

GitHub Repo Link: https://github.com/aliaankhowaja/KLT-Using-CUDA