CS4110 - High Performance Computing with GPUs

CCP Deliverable 1 - Report

Menahil Ahmad (23I-0546)
Hiyam Rehan (23I-0039)
HPC-A

Github Repository:

https://github.com/mena-aq/KLT-CCP

Introduction

The Kanade–Lucas–Tomasi (KLT) feature tracker is an algorithm used to detect and follow motion across sequential image frames by estimating 2D translation and scale variations of tracked points. The objective of this task is to parallelize KLT tracking to reduce computational overhead. This is achieved by profiling the existing implementation to locate performance hotspots and offloading the most time-consuming functions from CPU to GPU for accelerated processing.

Methodology

Three datasets were used for performance analysis—two recorded videos and one provided example, each containing fewer than 100 frames. The code was executed using compiled libraries and example3.c, followed by profiling with *gprof*. The profiling output was visualized using *gprof2dot* to generate call graphs showing execution time and call frequency per function. The timing data was logged and averaged across all datasets to identify the primary computational bottlenecks suitable for GPU parallelization.

Findings

From our observations, the main hotspots of the code are:

- **_convolveImageVert()** and **_convolveImageHoriz()**: Dominated execution time (~67% combined), primarily due to repeated matrix convolution operations applied to each pixel in every frame.
- **_interpolate()**: Accounted for ~20% of total runtime, not because of computational heaviness per call, but due to its high frequency of execution across features and frames (within *computeGradientSum()* and *computeIntensityDiff()* in *_trackFeature()*).

Remaining functions contributed negligibly (~0–4%) to total runtime and were excluded from further optimization (e.g., PPM write operations).

Decisions

Keeping these hotspots in mind, we propose implementing separate CUDA kernels for vertical and horizontal convolutions (*_convolveImageVert() and _convolveImageHoriz()*). Each kernel would parallelize the convolution at the pixel level, with each thread processing a single pixel. This fine-grained parallelism is well suited to the GPU execution model. Additionally, we propose a *_trackFeature()* kernel, where each block processes a single feature. Within each block, threads cooperatively perform interpolations over the feature's support window, thereby parallelizing *computeGradientSum()* and *computeIntensityDiff()*. This design exploits both inter-feature and intra-feature parallelism.

To estimate the theoretical performance upper bounds, we applied Amdahl's Law under the assumption that the kernelised functions execute with negligible runtime on the GPU. Our results are summarised in Table 2.1.

Kernel	**% time**	**Maximum Application Speedup**
convolveImageVertKernel()	0.3735	1.6x
convolveImageVertKernel() + *convolveImageHorizKernel()*	0.6739	3.07x
convolveImageVertKernel() + *convolveImageHorizKernel()* + *_trackFeatureKernel()*	0.8758	8.05x

Table 2.1 *– Upper Bounds of Application Speedup achieved with infinite speedup of proposed kernels*

Limitations

Several limitations must be accounted for in applying this optimisation strategy:

- **Iteration-dependent Workload Imbalance:** The number of iterations needed per feature position refinement is data-dependent and may lead to load imbalance.
- **Host-device Communication Overhead:** Transferring large image data between CPU and GPU memory can become a dominant cost, potentially diminishing the effective speedup.
- **Kernel Launch and Resource Contention:** Multiple large kernel launches may introduce overhead and resource contention issues.

Conclusion

In this stage of our work, we gained insight into the major hotspots of our KLT feature tracking application, and identified convolution and interpolation routines as the dominant hotspots, accounting for over 85% of execution time. To address these bottlenecks, we propose a GPU kernelisation strategy, to exploit fine-grained parallelism at a pixel level. Amdahl's Law predicts a theoretical upper speedup of around 8×, but in practice the effects of host–device communication and workload imbalance will likely limit the actual improvements. However, these results suggest that GPU offload is a promising direction for accelerating the KLT tracker, but careful optimization beyond kernelization will be necessary to achieve substantial real-world gains.