CS4110-High Performance Computing with GPUs

V3

Rayyan Waqar 23i-0531
Ibraheem Farooq 23i-0816
CS-5A

1. Introduction

This report presents the **final performance results**, profiling insights, and optimization achievements for **V3** of the CUDA-accelerated KLT Feature Tracker. The goal of D3 was to deliver a high-performance GPU implementation, compare it to the baseline CPU version, and demonstrate efficiency gains through compute and memory optimizations. All work is based on the provided baseline sequential implementation and optimized according to HPC principles taught in the course.

The KLT tracker involves corner detection, pyramid construction, and iterative optical-flow estimation across multiple frames. GPU acceleration is challenging due to the algorithm's mixture of **dense operations** (e.g., convolution, subsampling) and **irregular per-feature calculations**. The main performance bottleneck is typically communication overhead between CPU and GPU, particularly during pyramid uploads.

2. Summary of Optimization Strategy

The V3 implementation uses a set of CUDA-based optimizations focused on reducing memory transfers, improving locality, and exploiting GPU parallelism. The following strategies were implemented and validated:

2.1 Persistent GPU Buffers

GPU memory buffers for image pyramids and intermediate structures were pre-allocated once and reused for all frames. This removed repeated `cudaMalloc/cudaFree` operations—previously a major overhead.
Impact: *50 s → 15 s (3.3× speedup)*

2.2 Shared Memory Tiling for Convolution

Separable 2D convolution kernels were rewritten to cache tiles in shared memory, minimizing redundant global memory accesses. This optimization greatly improved pyramid smoothing performance.
Impact: *15 s → 9 s* (additional *1.67× speedup*)

2.3 GPU Subsampling Kernel

A custom 2× downsampling kernel was introduced to generate pyramid levels entirely on the GPU. While it does not provide a direct speedup by itself, it is critical for enabling the final caching optimization.
Impact: Support for full pipeline locality on GPU

2.4 Pyramid Caching (Eliminating 99.9% Transfers)

This is the most important optimization in V3. The image pyramid for each frame is uploaded once and reused during all interpolation operations, eliminating more than **130,000 cudaMemcpy calls**.
Impact:

- cudaMemcpy calls: *133,221 → ~100*
- Transfer time: *21.8 s → <0.3 s*
- Kernel execution improved due to reduced latency
- Overall runtime: *9 s → <3 s* (>3× speedup)

This optimization also required correcting a critical **sequential-mode bug** in the original implementation related to pyramid swapping, ensuring correctness on the GPU path.

3. Performance Results

3.1 Final Speedup Summary

Version	Runtime	Speedup (vs Naive GPU)	Notes
Naive CUDA	50 s	–	No optimizations
After Persistent Buffers	15 s	3.3×	Major overhead removal
After Shared Memory	9 s	5.6× total	Memory locality improvements
Final V3 (with Pyramid Caching)	<3 s	16× total	Eliminated 95–99% bottleneck
CPU Baseline	25 s	–	Improved dataset baseline

As required in D3, **GPU performance is faster than CPU performance**, with the optimized GPU version achieving approximately:

GPU Speedup over CPU ≈ 7–8×

4. Profiling and Analysis

Performance analysis was conducted using **Nsight Systems**, **Nsight Compute**, and **nvprof** to evaluate both kernel efficiency and memory-transfer behavior. The following metrics were examined:

4.1 Kernel-Level Metrics

- Thread occupancy
- Memory throughput
- Shared memory utilization
- Instruction efficiency
- Warp divergence

Shared memory tiling for convolution significantly improved **global memory load efficiency**, while eliminating many kernel launches reduced **launch overhead**.

4.2 Transfer and Bandwidth Metrics

- Total host-device transfers
- cudaMemcpy call counts
- Transfer bandwidth per frame

The profiling timeline clearly showed that the naive GPU implementation spent the majority of time in memory transfers, confirming that **communication optimizations are essential**—a key learning objective of CLO-4. After pyramid caching, the bandwidth graph flattened almost completely.

4.3 Correctness Validation

KLT outputs were validated using:

- Error tolerance checks on tracked feature coordinates
- Visual inspection of feature trajectories
- Frame-to-frame drift comparison vs CPU

The GPU version matches CPU results within numerical tolerance

5. Technical Achievements

The most significant accomplishments in V3 are:

1. **95–99% bottleneck removal** through pyramid caching.
2. **3× convolution speedup** via shared memory tiling.
3. **16× overall GPU improvement** from naive to optimized.
4. **GPU outperforms CPU** (3.5 s vs 25 s).
5. All CUDA improvements are cleanly isolated using **#ifdef USE_CUDA**, ensuring maintainability.
6. Advanced debugging and profiling tools were used as encouraged in the CCP instructions.

Link to Github Repository

https://github.com/ibbi1020/HPC_i230816_i230531.git