CS4110-High Performance Computing with GPUs

Complex Computing Problem Deliverable 2

Rayyan Waqar 23i-0531
Ibraheem Farooq 23i-0816

CS-5A

1. Introduction

This report presents initial GPU acceleration results for the Kanade-Lucas-Tomasi (KLT) feature tracker. Building upon the CPU baseline (V1), we ported four functions to CUDA: `_interpolate`, `_calculateMinEigenvalues`, `convolveHorizontal`, and `convolveVertical`. The V2 implementation uses naive parallelization with straightforward thread-to-pixel mappings and per-kernel data transfers. We expected moderate speedup to establish a working GPU pipeline before advanced optimizations in V3.

2. Experimental Setup

Hardware: NVIDIA GPU (Tesla/V100 class), x86_64 CPU. (Using Google Colab)
Software: CUDA 11.x/12.x, nvcc/gcc, Linux, Nsight Systems profiling.
Dataset: 10 frames, 640×480 grayscale PGM images, tracking 300 features.

3. Performance Results

Version	Runtime	Speedup	Notes
V1 (CPU)	1.903s	1.00×	Sequential baseline
V2 (GPU)	30.298s	0.063×	**15.9× slower**

CUDA API Breakdown (37.6s total):

- `cudaMemcpy`: 17.3s (45.9%) - 280,287 calls
- `cudaMalloc`: 9.9s (26.2%) - 280,287 calls
- `cudaFree`: 9.2s (24.6%) - 280,287 calls
- `cudaLaunchKernel`: 1.2s (3.3%) - 93,429 calls

Kernel Execution (0.28s total): `naiveInterpolateKernel` 282ms (99.9%), `minEigenvalueKernel` 0.19ms

Memory Transfers: 48.3 GB Host-to-Device (4.4s), 19.5 MB Device-to-Host (0.15s)

Time Distribution: Memory operations 97.6%, API overhead 1.7%, Kernel execution 0.7%

4. Analysis & Discussion

The naive GPU implementation is 15.9× slower likely due to catastrophic memory management. With 280,287 allocations/frees consuming 19+ seconds (96.7% of GPU time) and 48.3 GB of transfers for a ~30 MB dataset, the implementation allocates/frees memory for every operation instead of reusing buffers. Kernel launch overhead (13.3 µs) exceeds computation time (3 µs) by 4.4×. The CPU wins by maintaining cache-friendly data without transfer or allocation overhead. For small 640×480 images, CPU caching and SIMD outperform naive GPU parallelism.

The other most likely reason why this is failing is that we are using the original 9 image dataset that we were provided with, since the dataset is so small, the GPU cannot use its full capability and ends up being slower than the CPU. Convolutions work best when your data is big.

V3 Optimizations: We plan to optimize this memory allocation to achieve a much higher level of speedup than the CPU.

Link to GitHub Repository: https://github.com/ibbi1020/HPC_i230816_i230531.git

Link to Active Google Colab: i230816_i230500_D2