CS 4110 - High Performance Computing

Deliverable 1 Report

Ibraheem Farooq 23i-0816
Rayyan Waqar 23i-0531
Github: https://github.com/ibbi1020/HPC_i230816_i230531

BCS-HPC-A

1. Introduction

The Kanade-Lucas-Tomasi (KLT) algorithm tracks points of interest across multiple frames in a video. Its goal is to follow small distinctive patches (called features) through time, enabling applications such as object tracking and motion analysis.

Although conceptually simple, KLT involves a series of repeated computations on each frame and feature. This makes it computationally heavy, especially when tracking large numbers of points or processing high-resolution video. This complex computation problem can be broken down into a series of 'embarrassingly parallel problems' which can be offloaded to GPU's for optimized computation. The purpose of this report is to analyze which parts of KLT consume the most resources and how much benefit can realistically be expected from such parallelization.

2. Structure of the Algorithm

KLT operates in three main steps:

- **Feature Selection**: Picking suitable features that are easy to track.
- **Feature Tracking**: Following those features between frames using iterative refinement.
- **Image 'Pyramid' Processing**: Handling different scales of the image to track large movements.

Each of these steps contains functions with varying computational weight. Profiling results show that feature tracking dominates execution time **(≈85%)**, while feature selection accounts for ≈12%. Supporting tasks, such as storing feature lists, are minimal.

3. Profiling and Complexity Analysis

Based on the profiling results, the majority of computation occurs in a few functions:

- **Image Convolution** (convolveImageHoriz, convolveImageVert): ~70% combined.
- **Interpolation** (_interpolate): ~18%.
- **Tracking Loop** (_trackFeature): ~18% cumulatively
- **Gradient Calculations** (_KLTComputeGradients): ~46% when viewed at sub-function level.

These functions represent the "hot spots" of the algorithm. Most of them process pixels or features independently, which makes them highly parallelizable.

From a complexity standpoint:

- Feature selection is roughly proportional to the image size and the search window.
- Feature tracking grows with the number of features, the number of pyramid levels, and the number of iterations needed for convergence.

Overall: O(n * m * L * k * w²) where n*m is image resolution, L is pyramid levels, w is window size for feature tracking and k is number of iterations.

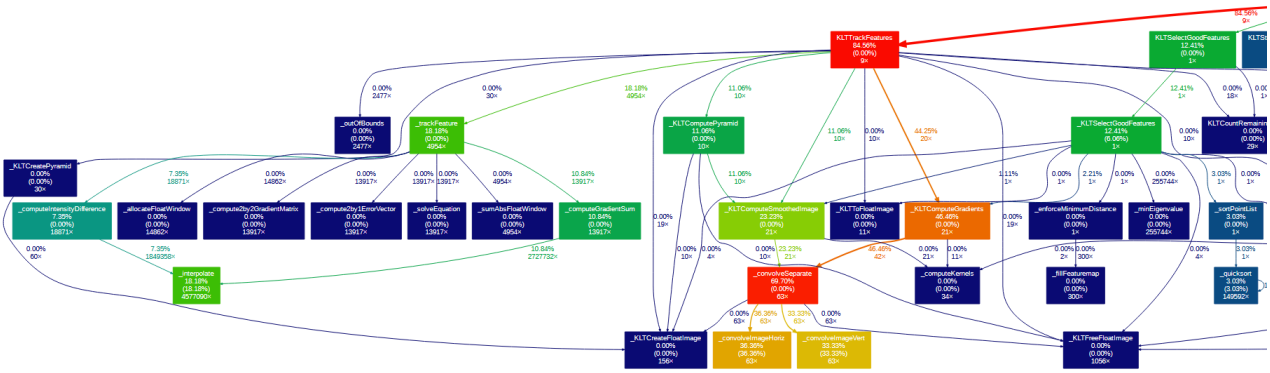

Fig 1: KLT profiling, run on example 3

4. Parallelization Potential and Amdahl's Law

Iterative updates for each feature and sequential pyramid construction must remain serial. However, approximately **85%** of the computation is parallelizable (convolutions, gradient calculations, interpolation, and per-feature operations).

According to Amdahl's Law, with 85% parallelizable and 15% serial, speedup approaches **6.7x**.

5. Conclusion

KLT is computationally demanding but well-suited for parallelization. Profiling shows that a handful of functions dominate execution, especially convolutions and interpolation. These can be efficiently offloaded to GPUs to achieve large reductions in runtime.