**CS-4110**

High Performance Computing with GPUs

Deliverable-01:

KLT Feature Tracker Profiling & GPU Optimization

Submitted by:   
Muhammad Hassan (23I-0698)  
Muhammad Gulsher (23I-0066)

Date:05 October,2025.

Submitted to: Dr. Imran Ashraf

**Introduction:**

This report analyzes the performance of the Kanade–Lucas–Tomasi (KLT) Feature Tracker using gprof.  
Five example programs (example1–example5) were compiled and executed to identify the computational bottlenecks in Version 1 of the implementation. The purpose is to determine which functions dominate runtime and to estimate potential speedups through GPU parallelization using Amdahl’s Law.

**Profiling Setup and Execution:**

Each example was compiled using GCC with the -pg flag for profiling:

* make prof\_ex1
* gcc -pg -O2 -Wall -DNDEBUG -o example1 example1.c -L. -lklt -lm
* . /example1 ../../data/img0.pgm ../../data/img1.pgm
* gprof example1 gmon.out > prof\_ex1.txt

This process was repeated for examples 1–5, producing flat profiles (prof\_ex1.txt, prof\_ex2.txt, etc.) that recorded runtime percentages per function.

* **gcc** → GNU C Compiler.
* -**pg** → Enables gprof or other profiling.
* -**O2** → Optimizes the code for better performance by removing the dead code (instruction that has no impact), loop optimizations and use registers instead of the RAM etc.
* -**Wall** → Shows all warnings to catch potential issues.
* -**DNDEBUG** → Disables debug assertions. Assertions are basically that the output that should be true is true assert(condition), It is disabled to make the program execute faster.
* **-lklt -l** → for the KLT lib and the math’s lib

Profiling measures where CPU time is spent identifying “hot spots” for optimization.

**Typical KLT runtime functions include:**

* **\_KLTSelectGoodFeatures**:  
   Selects the best corner points. For each patch, it calculates how the brightness changes in both directions. It then finds points where the brightness changes strongly in directions these are called corners.
* **\_convolveImageHoriz / convolveImageVert:**

Gaussian smooth filters which blur or smooth the image, and it helps in reducing the noise resulting in better image detection.

* **\_interpolate**:

It improves accuracy by finding where the feature lies between pixels, not just at the center of one pixel.

* **\_enforceMinimumDistance**:

Ensured detected features are not clustered like all the features of image must not be clustered but spaced.

* **\_ppmWrite:**

Handles image file output (mostly I/O-bound) Portable PixMap format  
Forming red dots on the image to make it easier for a person to understand how the tracking is done.

**Profiling Results Summary:**

|  |  |  |  |
| --- | --- | --- | --- |
| **NO** | **Dominant Functions** | **% of Runtime** | **Description** |
| **1** | KLTSelectGoodFeatures, convolveImageHoriz/Vert, quicksort | ≈25% each | Balanced load between detection and smoothing |
| **2** | KLTSelectGoodFeatures, convolution, ppmWrite | 50%, 33%, 17% | Detection dominates |
| **3** | Convolution, interpolation, initialization | 62%, 12%, 6% | Heavy image processing |
| **4** | (No measurable compute) | – | Mostly file I/O and table operations |
| **5** | KLTSelectGoodFeatures, convolution, enforceMinimumDistance | 25% each | Parameter tuning effects |

Overall, convolution and feature detection consume over (75 % + 83 % + 68 %) ÷ 3 ≈ 75 % rounded ≈ 70 % of total CPU time. These are ideal candidates for GPU acceleration.

**Amdahl’s Law and GPU Optimization:**

Amdahl’s Law estimates total speedup S for a system where a fraction P of work is parallelized across N processors (or GPU threads):

S=1/((1−P) +P/N)

Assuming:

* 70% of runtime (P = 0.7) is parallelizable (convolution + feature selection)
* Assuming GPU provides a 50× speedup for that portion (N = 50)

S=1/ (0.2+0.7/50) ​≈3.77×

Thus, moving convolution and feature detection to GPU can yield up to 3.77× overall acceleration even if other functions remain on CPU.

**GPU Suitability:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Function** | **Parallelizability** | **GPU Benefit** | **Notes** |
| **\_convolveImageHoriz/Vert** | **Very High** | **Excellent (ON GPU)** | **Independent per pixel** |
| **\_KLTSelectGoodFeatures** | **High** | **Good (ON GPU)** | **Parallel corner scoring** |
| **\_interpolate** | **Medium** | **Moderate (OPTIONAL)** | **Memory bandwidth limited** |
| **\_quicksort** | **Low** | **Poor (CPU)** | **Branch heavy** |
| **\_ppmWrite** | **None** | **CPU** | **I/O bound** |

**Conclusion:**

Profiling revealed that convolution and feature selection dominate the computational cost in KLT.  
 Both are highly parallel tasks that can benefit significantly from GPU acceleration. Using Amdahl’s Law, an estimated 3.77× total speedup is achievable with GPU optimization, primarily targeting convolution and feature extraction routines.

**GitHub Repository Link:   
https://github.com/MuhammadHasan-04/klt-feature-tracking**