KLT Feature Tracker Acceleration on GPUs

Complex Computing Problem

CS 4110 - High Performance Computing with GPUs

This Complex Computing Problem (CCP) deals with the GPU acceleration of well-known feature tracker application, namely KLT feature tracker. You will gain hands-on experience in GPU programming, parallel computing, and performance optimization. By the end of the project, you should be able to apply HPC concepts to accelerate real-world computer vision efficiently. You are provided with a baseline implementation of KLT feature tracking application. Use the Makefile to compile and execute this application. You are supposed to work in a team of 2-3 members to optimize this implementation for GPU using CUDA. Although you have to work in a team, still you need to clearly claim what part was your contribution in the project. This should also be evident from the regular Github repository commits.

Target CLOs

CLO2. Identify application hotspots based on the application profile using realistic workload. [Analyze, C4]
CLO3. Develop data-parallel solutions using appropriate programming model. [Create, C6]
CLO4. Analyze performance of an application running on an HPC system to improve compute and memory performance. [Evaluate, C5]

Complex Computing Problem Attributes

No	Attribute	Check	Justification
1	Range of conflicting requirements	✓	Solution involves range of conflicting issues in the optimization process.
2	Depth of analysis required	✓	Solution requires in depth analysis of application profile to make decision of partitioning parts of application running on CPU and GPU.
3	Depth of knowledge required	✓	Solution also requires depth of the domain knowledge (CV) and an in depth understanding of GPU architectures to do efficient application mapping.
4	Familiarity of issues		
5	Level of problem		
6	Extent of stakeholder involvement and level of conflicting requirements		
7	Consequences		
8	Interdependence		
9	Requirement identification		

CCP Deliverables

No	Deadline	Deliverable	Evaluation
CCP-D1	6th Oct, 2025	1. V1 complete source-code in a zip file which can be compiled and executed on our GPU server. Do not include temporary files as well as the dataset in this zip. Makefile should have rules which you used to generate the profile. 2. 2-page report on decisions regarding which functions will be ported to GPUs and why. Mention link to the private Github repository.	Implementation 30% Report 30% Presentation 40%
CCP-D2	17th Oct, 2025	1. V2 complete source-code in a zip file which can be compiled and executed on our GPU server. Do not include temporary files as well as the dataset in this zip. Makefile should have rules to execute application on CPU as well as on GPU. 2. 2-page report on performance results.	Implementation 30% Demo 20% Report 20% Presentation 40%
CCP-D3	31st Oct, 2025	1. V3 complete source-code in a zip file which can be compiled and executed on our GPU server. Do not include temporary files as well as the dataset in this zip. Makefile should have all the required rules to compile and execute the code out of box. 2. 2-page report on performance results.	Implementation 15% Demo 15% Speedup ranking 20% Report 20% Presentation 30%
CCP-D4	14th Nov, 2025	1. V4 complete source-code in a zip file which can be compiled and executed on our GPU server. Do not include temporary files as well as the dataset in this zip. Makefile should have all the required rules to compile and execute the code out of box. 2. Report (not more than 6 pages formatted as research paper) which documents your work.	Implementation 15% Demo 15% Speedup ranking 20% Report 20% Presentation 30%
CCP-D5	2nd Last week of the course, exact date/time TBA	1. Final Presentation	Slides Content 20% Slides Quality 20% Speech 30% Q&A 30%

Assessment Rubrics

CCP-Deliverable	CLO	Exemplary 15-20	Proficient 10-15	Developing 5-10	Worst Performance 1-5
CCP-D1	2	Students were able to generate the realistic profile of the application and they were clear about the role of the workload. Students were also able to defend their analysis and figure out the hotspots. Moreover students also discussed the role of the CPU-GPU communication in making the analysis.	Students were able to able to generate the realistic profile of the application and they were clear about the role of the workload. Students were also able to defend their analysis and figure out the hotspots.	Students generated the profile of the application but could not identify the hotspots.	Students were unable to generate the application profile and could not identify the hotspots.
CCP-D2, CCP-D3, CCP-D4	2(50%), 3(50%)	Students developed highly optimized V3 as well as efficient V4 utilizing OpenACC directives. The results are also well defended and presented in the deliverables. Students were able to compare the tradeoffs of all the versions especially V3 and V4.	Students developed the GPU solution and applied fair number of optimizations in terms of launch configuration, shared memory access, streams and other advanced optimizations discussed in the course. Advanced V3 optimizations.	Students developed GPU based parallel solution and applied some basic optimizations in terms of launch configuration, occupancy etc. Basic V3 optimizations.	Students developed some GPU code but they were not able to gain any performance improvement. Basically V2 is complete.
CCP-D5	2(20%), 3(40%), 4(40%)	Apart from sound profile guided application analysis, students also performed the analysis with the peak performance and produced results in the form of scalability graphs.	Utilized the profiler to the application analysis and performed informed decisions to improve performance in terms of compute and memory/communication efficiency.	Utilized profiler and occupancy calculator to at least guide the launch configuration analysis.	No use of any profiler. Just use intuition to analyze and improve performance.

Background

Feature Tracking

In computer vision (CV), feature tracking refers to the process of detecting distinctive points, patterns, or regions in an image (called features) and then following their movement across subsequent frames in a video sequence. Features are typically chosen for their uniqueness and stability, such as corners, edges, or texture patches, because these are easier to recognize consistently over time.

For example, algorithms like the Kanade-Lucas-Tomasi (KLT) tracker detect corner points in an image and then track their displacement frame by frame. This allows the system to estimate motion, recognize objects, or reconstruct 3D structures from 2D video.

Applications include

- Object tracking: following a moving car in a traffic video.
- Motion estimation: measuring how fast and in what direction objects move.
- Augmented reality (AR): anchoring virtual objects to stable features in the real-world scene.
- Robotics: enabling robots to navigate by tracking landmarks in their environment.
- In short, feature tracking is a fundamental tool that enables computers to understand and interpret dynamic scenes by following how visual elements evolve over time.

KLT Feature Tracking

KLT detects good corner-like features and tracks them across video frames using local optical flow estimation, refined with pyramids for robustness. Some applications of KLT are

- Video stabilization
- Object tracking (e.g., tracking cars, humans)
- Structure-from-motion (3D reconstruction from video)
- Augmented reality (sticking virtual objects to moving points)

KLT is one of the most widely used feature tracking algorithms in computer vision. It combines good feature selection with an efficient optical flow method to track them frame by frame.

Feature Detection (Tomasi–Kanade "Good Features to Track")

- First, the algorithm detects strong, distinctive points in the image (often corners).
- A good feature is one where the image intensity changes significantly in both directions (x and y).
- Mathematically, this is measured using the eigenvalues of the image gradient matrix (from the structure tensor).
- Features with large eigenvalues are chosen, because they are less ambiguous and easier to track.

Example: Corners of a window or checkerboard are “good features,” while flat regions or straight edges are not.

Optical Flow Assumption

KLT assumes that between two consecutive frames:

- Pixel intensities of small patches remain nearly constant.
- Motion between frames is small.
- Neighboring pixels in a patch move in roughly the same way (coherence).

This is called the brightness constancy assumption.

Lucas–Kanade Method for Tracking

- For each detected feature (corner), the algorithm takes a small window/patch around it.

- It computes how this patch has shifted in the next frame by solving a set of linear equations using image gradients.
- The displacement vector gives the new position of the feature.

Pyramid Implementation (for Large Motion)

- If the motion is too big, a multi-scale image pyramid is used.
- Start tracking at a coarse (downsampled) scale → refine at finer scales.
- This makes the tracker robust to larger displacements.

Tracking Over Time

- The tracked points are updated in every frame.
- If a feature drifts too much, becomes occluded, or moves out of frame, it is dropped.
- New features can be detected to keep tracking fresh points.

Evaluation Notes

- Correctness and accuracy of the GPU-accelerated implementation.
- Performance improvement over the baseline CPU implementation.
- Quality and efficiency of CUDA optimizations.
- Clarity and depth of performance analysis in the final report.
- Code readability, modularity, and documentation quality.
- Your familiarity with the code as per demo and presentation
- Your contribution as an individual in the code development as evident by Github repo commits. Keep your Github repository private.

Version Naming Notes

You should name your implementation as:

1. V1 : The given sequential implementation
2. V2 : A naive GPU implementation
3. V3: An optimized GPU implementation in which you should try at least the following optimizations
    a. Launch configuration
    b. Occupancy
    c. Communication optimizations
    d. Memory optimizations (properly use memory hierarchy)
4. V4: OpenACC pragma based optimized implementation

Github Repo Notes

- Each version should be in a separate directory
- Work in Github repository from the start
- Keep your Github repository private
- The Gihub repo as well as the deliver should have the following directory structure
    1. src
        a. V1
        b. V2
        c. V3
        d. V4
    1. data (which contains images that you used)
    2. report
    3. slides
    4. Readme.md file describing your project, instructions to compile execute it as well as other relevant instructions