Real-Time Visual SLAM Frontend and 3D Mapping on CUDA GPUs

Team: Suchir Vemulapalli (suchirv) & Kaeshav Mandyam (kaeshavm)

Github Repository: https://github.com/SuchirVemulapalli/Parallel_Computing_Final_Project

Summary
We will implement a real-time Visual SLAM frontend on CUDA, including parallel FAST feature detection, ORB descriptor extraction, GPU-accelerated feature matching, and a minimal 3D mapping + camera trajectory pipeline.

Background
Visual SLAM (Simultaneous Localization and Mapping) estimates a camera's 3D motion while reconstructing a sparse 3D map. SLAM is used in many real world applications like robotics and autonomous driving for real time map creation of a surrounding space.

SLAM frontends have two major tasks:

- Feature detection + description (e.g., FAST + ORB)
- Feature matching + geometric verification (e.g., Hamming matching + PnP)

FAST corner tests involve independent pixel comparisons.

1. Iterate over pixels and perform the following computations
2. Iterate over ring of pixels surrounding reference pixel
3. Compute whether the pixel is darker or brighter than the current pixel
4. If N or more consecutive pixels are all brighter or all darker than the reference pixel, assign that pixel as a corner.

Traditional FAST does not calculate orientation, but to create a descriptor for each feature that is orientation invariant, we need to measure where each corner is facing. To do this, we compute the moments of a patch surrounding the keypoint via the surrounding points' intensity. From these moments, we can calculate the centroid of the patch, calculate the vector from the corner to the centroid, and then rotate the patch to a canonical orientation before calculating the descriptor.

ORB descriptors run 256 binary comparisons per keypoint in order to create a unique "fingerprint" for a keypoint. Creation of a 256 bit binary descriptor is as follows:

1. Sample 2 points uniformly at random in a 16x16 patch surrounding the keypoint.
2. If point 2 is darker than point 1, append 1 to our binary descriptor
3. Otherwise append 0

With the ORB descriptor we need the skew matrix provided by the previous FAST kernel, this will allow us to match keypoints between frames even when there is rotation.

Matcher computes Hamming distances between all keypoint pairs to match keypoints in a frame with its previous frame.

After matching, we compute 3D map points via triangulation and a camera trajectory using PnP.

The Challenge
ORB-SLAM does not just consider all gestures in a frame. It chooses a small, balanced number of keypoints per region and drops many detected corners. Techniques like non-max suppression and RANSAC dropping need to be implemented to obtain a truncated list of points. Since these techniques are not inherently embarrassingly parallel, we will need to figure out how to implement them efficiently to maximize the hardware. Additionally, ORB-SLAM is inherently a sequential pipeline that feeds data from a processed frame into the next frame. We will need to figure out how to optimize computation through the use of pipelined parallelism or by preprocessing frames with computations that are independent between frames and only go sequential when necessary. The workload also involves many irregular memory accesses and divergent execution patterns since features can be centered around certain key patches within a frame. Since GPUs run warps of threads in lockstep, the divergence between pixel computations will prove a challenge for mapping the workload to the hardware.

Resources
We will be developing on the GHC machines as they have GPUs and already have CUDA installed. For our serial version to compare against, we will use the OpenCV CPU implementations as a baseline.

Goals and Deliverables
50%
- Implement FAST corner detection kernel in CUDA
    - Implement rotation invariance
    - Implement non-max suppression
- Implement ORB descriptor kernel in CUDA

75%
- Implement hamming matcher and integrate full FAST, ORB, matching pipeline to detect keypoint tracking across frames

100%
- Implement map creation and triangulation

125%
- Implement RANSAC optimization (more robust pose estimation by removing outliers)

150%
- Implement loop closure (recognize previously visited locations to prevent long-term drift in pose estimation)

We will include a demonstration of the pipeline working in our report using a sample input.

Platform Choice
We are choosing to implement our project in CUDA on the GHC NVIDIA GPUs due to the data parallel thread block and warp architecture being particularly well suited for computer vision and

image processing tasks. Additionally, the high memory bandwidth of GPUs allows for very fast image processing functions on high resolution frames.

Schedule

Week	
11/17-11/23	● Implement FAST corner detection kernel in CUDA ● Work on ORB descriptor and hamming matcher if time permits
11/24-11/30	● Implement ORB descriptor kernel in CUDA ● Implement hamming matcher and integrate all kernels into pipeline ● Continue working on triangulation and map creation if time permits
12/1-12/7	● Implement triangulation and map creation ● Attempt 125%, 150% goals if time permits ● Work on final report

Work Distribution
Kaeshav: FAST kernel, RANSAC (125% goal)
Suchir: ORB descriptor kernel, hamming matcher, loop closure (150% goal)
Both: Triangulation and map creation, kernel pipeline integration