Parallel Video Object Tracker - 15618 Project Proposal

Eric Li
Carnegie Mellon University
Pittsburgh, PA 15213
shizhuol@cs.cmu.edu

Yanxin Jiang
Carnegie Mellon University
Pittsburgh, PA 15213
yanxinj@andrew.cmu.edu

1 URL

https://github.com/askEric0/Parallel-Video-Object-Tracker.git

2 SUMMARY

We are developing a high-performance visual object tracker implemented in C++ and CUDA. This system aims to achieve high performance on video object tracking, both in speed of processing and GPU utilization.

3 BACKGROUND

Object tracking is a computer vision technique that identifies and follows a specific object across consecutive frames in a video. A tracker typically begins with an initialized bounding box in the first frame, extracts a representation of the object, and then searches for the best matching region in each subsequent frame.

The search and evaluation process for object tracking is computationally intensive. For each frame, the tracker may need to evaluate hundreds of candidate regions. As a result, even relatively simple tracking algorithms can quickly exceed the processing capacity of a CPU when real-time frame rates are required. Motion and noise further increase the difficulty and may require broader search regions, adding more computation per frame.

Most of these computations can be parallelized. Evaluating one candidate regions doesn't depend on others. This independence enables massive spatial parallelism, allowing thousands of threads to process different parts of the search area simultaneously. Since similarity computation dominates the overall runtime, accelerating this component through parallelism provides the greatest performance benefit. Our current idea is shown as below:

```
for each frame:
    for each candidate region (parallelizable):
        evaluate the matching of the every candidate region
    choose location with the highest matching
```

4 THE CHALLENGE

Object tracking is challenging to parallelize because the overall task contains both parallel and sequential components. While the visual similarity computation between the template and the incoming video frames is parallel, the high-level tracking logic is inherently sequential: the bounding box in frame t+1 depends on the position found in frame t. This indicates that the project must separate parallelizable workload from the sequential state update, and hybrid the parallel and sequential desgin.

The workload also places heavy pressure on memory bandwidth. Template matching requires sliding a window across each frame, causing significant global memory access with overlapping regions and low arithmetic intensity. Without careful GPU optimization, such as shared memory tiling, coalesced memory access, and block-level synchronization, the kernel becomes memory-bound and performs poorly. These memory characteristics—large 2D arrays, overlapping reads, and strided access patterns—make this a difficult problem to efficiently map onto CUDA's hierarchical memory architecture. Divergent execution at image borders and the challenge of fitting shared tiles into limited SM memory further increase the complexity of the kernel design.

5 RESOURCES

We plan to take advantage of both resources from the class and personal resources:

5.1 Hardware

We plan to develop and run the project primarily on a Linux machine equipped with an NVIDIA GPU. CUDA programming requires access to NVIDIA hardware, and these GPUs provide more than enough computational capability to run and evaluate the performance of my kernels.

The ideal choice of machine will be the GHC cluster machines for benchmarking and performance testing.

5.2 Software

CUDA 11+ and recent NVIDIA driver support will be required. The machine should be able to run C++ and CUDA, that is, with C++ and nvcc compiler installed.

5.3 Code

The project will be mostly starting and take advantage of OpenCV packages for frame loading and basic image manipulation. No additional external object tracking libraries (e.g., OpenCV Tracker API, DeepSORT, OpenCV DNN) will be used because they would hide the parallel compute parts of the project, and provides no help in batching frames.

5.4 Potential Future Needs

The potential needs include better and more recent high performance GPUs, and may need to increase the quota of AFS storage spaces, if we plan to consitently use GHC machines and AFS.

6 GOALS AND DELIVERABLES

The project goal is to build a GPU paralleled tracking system. It will be a CUDA accelerated single object tracker, that is designed to take advantage of GPU parallelism, memory optimization, and it will improve over CPU baseline and non-optimized GPU baseline. The tracker follows a target object in video across a video sequence by comparing a reference template against incoming video frames and locating the best matching region.

To take advantage of GPU parallelism, the tracker system processes batches of video frames simultaneously on GPU. While object tracking requires sequential updates over stream of frames, the visual similarity comparison if fully independent (between template and frames). Batching frames will maximize GPU utilization and increase throughput.

We plan to improve the speed of video processing with at least $5\times$ from a baseline when tracking an element, and will show a demo video during the day of presentation.

6.1 Milestone Goals (50%, 75%, 100%, 125%, 150%)

50% Goal: Baseline and Infrastructure Completed

- Implement a CPU-based tracker for the reference of correctness.
- Implement a naive CUDA kernel for similarity computation without optimization

75% Goal: Initial GPU Speedup

- Add batched frame processing to improve GPU utilization.
- Implement shared memory tiling and block-level synchronization for CUDA optimization.

100% Goal (Target Completion): Optimized CUDA Tracker

- Achieve about $5\times$ speedup over CPU baseline.
- Demonstrate stable real-time or near-real-time tracking on short videos.

125% Goal: Extended Optimizations and Robustness

- Real-time tracking by tuning batch size and accept streaming.
- Achieve more than $5\times$ speedup on parallel implementation.

150% Goal: Advanced Features Improvements

- Achieve multiple-object tracking by running several trackers concurrently.

7 PLATFORM CHOICE

CUDA on an NVIDIA GPU is an ideal platform for this project because the core workload—template matching and pixel-level similarity—maps naturally to thousands of parallel threads. CUDA gives the control, performance, and parallel capabilities necessary to build an efficient and scalable tracking system.

8 SCHEDULE

The schedule will be roughly divided into 4 weeks' workload. Week 4 is short hence allocated for project report writing.

Week 1:

1. Implement CPU baseline tracker (template matching / NCC) for correctness reference. (Yanxin + Eric)
2. Set up project structure, build system (CMake), and OpenCV I/O + visualization. (Eric)
3. Write a naïve CUDA kernel for template matching (no optimization). (Yanxin + Eric)
4. Verify correctness (Yanxin)

Week 2:

1. Add batched frame processing (process 4–8 frames per kernel launch). (Yanxin)
2. Integrate GPU similarity output back into sequential tracking update. (Yanxin)
3. Implement shared memory tiling + block-level synchronization for CUDA optimization. (Eric)

Week 3:

1. Finalize CUDA optimizations (coalesced reads, minimizing redundant loads). (Yanxin + Eric)
2. Ensure stable real-time or near-real-time tracking on short videos. (Yanxin + Eric)
3. Test on videos, verify correctness. (Eric)

Week 4:

1. Project Report Writing. (Yanxin + Eric)
2. Optional: Real-time tracking by tuning batch size and accept streaming. (Yanxin + Eric)