15-418 Project Proposal
Kunal Jobanputra (kjobanpu), Ryan Stentz (rastentz)

Parallel Corner Detection

URL
https://kjobanputra.github.io/Parallel-Corner-Detection/

Summary
We are going to implement a parallel-optimized Harris Corner Detector on the NVIDIA GPUs in both the labs and in embedded devices with and without dedicated GPUs such as the NVIDIA Jetson Nano and Raspberry Pi 4. This optimized corner detector will be useful for real-time image feature extraction in embedded devices.

Background
Image feature extraction is an important first step in the pipeline of numerous computer vision algorithms. As such, it is important to be able to extract high quality features quickly. Traditionally, corners have been a sought-after image feature for their invariance to rotation, scale, illumination variation, and noise. The Harris Corner Detector is by far one of the most popular algorithms for corner feature extraction, however, it is relatively computationally intensive. The basic Harris Corner Detector algorithm is outlined below:

1. For each pixel in the image, compute them matrix

$$G = \begin{bmatrix} g_{xx} & g_{xy} \\ g_{xy} & g_{yy} \end{bmatrix}$$

$$g_{xx} = \left(\frac{\delta I}{\delta x}\right)^2 \oplus w$$
$$g_{yy} = \left(\frac{\delta I}{\delta y}\right)^2 \oplus w$$
$$g_{xy} = \left(\frac{\delta I}{\delta x}\frac{\delta I}{\delta y}\right) \oplus w$$

   Where w represents the Gaussian kernel and $\oplus$ represents the convolution operation
2. For all pixels, compute $c(x, y) = \det(G) - k(\text{trace}(G))^2$
3. Threshold each $c(x, y)$ value
4. Perform non-maximum suppression over all $c(x, y)$ to extract the corners

For the Harris Corner Detector, all steps involve repetitive computation that could benefit greatly from parallelism. In the first step, each element of G can be computed in parallel. Additionally,

the convolution operation itself can be parallelized. Both the second and third steps can benefit from parallelizing over pixels. The final step involves scanning a window over each pixel's neighborhood to determine if it is the local maximum.

The Challenge

There are a few challenges associated with parallelizing the Harris Corner Detector. In terms of the workload, each step in the algorithm presented above is dependent on the previous step. The first step involves plenty of convolutions which are known for their high communication to computation ratio. Step 2 has a higher computation to communication ratio as long as the G matrix is stored locally. Step 3 also suffers from a high communication to computation ratio. Finally, step 4 involves divergent execution as some pixels can be eliminated more quickly than others.

We plan on testing our Harris Corner Detector implementation on a NVIDIA Jetson Nano and a desktop-grade NVIDIA GPU. The NVIDIA Jetson Nano has a 128 core GPU with 4 GB of LPDDR4 memory. The memory speed and core count of this GPU do not compare to a standard desktop grade GPU, which typically have around 1000+ CUDA cores and 8 GB of DDR6 memory. With these system limitations, the high communication to computation ratio mentioned above may have a significant effect on performance. Specifically, copying the result to the CPU could prove a serious bottleneck. Additionally, special care must be taken when mapping the convolutions to the limited amount of CUDA cores.

Resources

For resources, we mostly require only hardware. Luckily, we already have access to a Jetson Nano, an RTX 2080 Super + GTX 1070, and a Raspberry Pi 4. In terms of software, the Harris Corner Detector is a well documented algorithm, so we will not be using any specific research for the algorithm itself. Currently, we plan on writing our own serial implementation of the Harris Corner Detector and do not plan on using any starter code.

Goals

Plan to Achieve

- Enable the Harris Corner Detector to run in significantly faster than real time (around twice as fast). For perspective, we hope to run our corner detector with an image stream of 30 frames per second. If we could achieve this, it would free up time to do interesting computations with the result of our Harris Corner Detector in real time
- Keep power consumption low. With the addition of a GPU, we will inevitably use more energy. The hope is to finish the computation quickly so that we are using more energy over less time. If we could achieve this, it could extend the battery life of embedded applications that rely heavily on computer vision algorithms

- Tackle bigger problems. The limited computational abilities of embedded devices typically limits the input resolution to computer vision algorithms. Our hope is that our parallel corner detector will allow for higher resolution inputs to computer vision algorithms to run in real time on embedded devices. If we can achieve this, we could improve the performance of these computer vision algorithms

Hope to Achieve

- If time permits, we would also like to take on the challenge of implementing a parallelized version of a computer vision algorithm that would utilize the results from our Harris Corner Detector. One possible algorithm is homography estimation which takes in matched image features and attempts to find the transformation that relates them. Parallelizing this algorithm would be challenging, as it would require parallelizing singular value decomposition.
- We'd also like to implement a parallel Harris Corner Detector on the Raspberry Pi 4 using OpenMP to utilize its multicore processor. We'd like to do this because the Raspberry Pi is one of the most common embedded devices for hobbyists.

For the poster session, we plan on having a demo of the harris corner detector running with another computer vision algorithm (homography estimation) in real time. As output, we can show a visualization of the harris corner detector overlaid on the live video feed. Additionally, we can show speedup and work size graphs on the NVIDIA Jetson Nano as well as the NVIDIA RTX 2080 Super to compare the performance of our corner detector and demonstrate our ability to downscale.

For our system, we are focused on implementing our corner detector on an embedded system, specifically the NVIDIA Jetson Nano. This system is equipped with a quad-core ARM cpu, and a 128 CUDA core GPU with 4 GB of LPDDR4 memory. It typically consumes around 10 watts. It is hard to estimate exactly what this type of system is capable of without running benchmarks, but we would guess the system is able to run the serial harris corner detector on its CPU at around 10 frames per second. As mentioned in our goals, we hope to run our parallel harris corner detector on the GPU at twice the speed of real time (30 frames per second).

Platform Choice

We have chosen to write our Harris Corner Detector in CUDA and C++, targeting the NVIDIA Jetson Nano. We chose C++ for its speed and relative ease of development (with respect to raw C). In addition, many serial implementations of the Harris Corner Detector already exist in C++.

For the hardware, we chose the embedded NVIDIA Jetson Nano since it is one of the cheapest embedded devices with a dedicated GPU. We chose an embedded device because computer vision algorithms are making their way into many embedded applications such as toys,

household robots, and cell phones. These algorithms are computationally expensive, while embedded computers typically have low computational resources but a high potential for parallelism. Thus, parallelising a corner detector could bring more interesting computer vision algorithms to embedded devices.

Schedule

Week 1 (11/2 - 11/6) - Finish proposal, research hardware, and acquire / set up hardware

Week 2 (11/9 - 11/13) - Develop serial version of the Harris Corner Detector to run on the NVIDIA Jetson Nano

Week 3 (11/16 - 11/20) - Develop and implement a method to parallelize step 1 and 3 of the Harris Corner Detector. Also, prepare for the checkpoint

Week 4 (11/23 - 11/27) - Develop and implement a method to parallelize step 4 of the Harris Corner Detector

Week 5 (11/30 - 12/6) - Run benchmarks comparing the serial and parallel versions of the Harris Corner Detector on the NVIDIA Jetson Nano. Evaluate results. Also look into methods for parallelizing the Harris Corner Detector on popular embedded devices without a dedicated GPU

Week 6 (12/7 - 12/11) - Benchmark a homography computation on the NVIDIA Jetson Nano using both the serial and parallel implementations for the Harris Corner Detector. Compare performance

Week 7 (12/14 - 12/18) - Finish any implementation details that remain from previous weeks. Prepare graphics / demo for final presentation