GPU-Accelerated Parallelization of Common Computer Vision Applications

Emilio Del Vecchio, Kevin Lin, Senthil S. Natarajan
Department of Electrical and Computer Engineering, Rice University

I. INTRODUCTION

-Graphics Processing Units (GPU's) have recently become popular for their ability to parallelize computations and achieve much greater efficiency as compared to the serial implementation of a CPU.

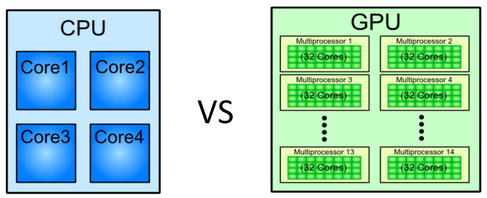

Fig 1. The GPU's parallel architecture allows it to parallelize massive computations.

-**Our objective:** utilize the GPU's ability to accelerate computer vision applications like facial recognition, edge detection, and motion tracking.

-Due to the proliferation of computer vision in many regular tasks, this affords us an opportunity to explore how to make such common algorithms more efficient.

II. FACIAL RECOGNITION

1. Extract HAAR features of a face from a set of positive and negative training images.

Fig 2. Example HAAR features of a random face

2. Use Adaptive Boosting machine learning to train a multi-stage, or cascade, classifier, against positive and negative HAAR features.

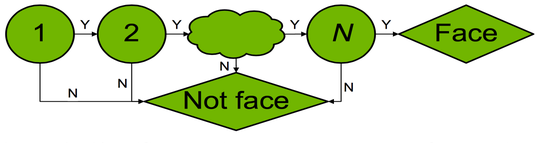

Fig 3. Cascade Classifier training rejects most non-face regions early.

3. Apply final cascade classifier to a loaded image. The OpenCV library provides modules to encapsulate this process.

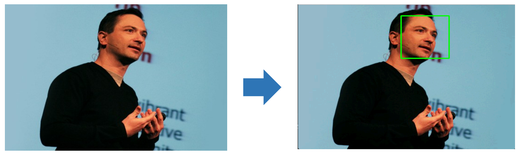

Fig 4. Original sample image (left), detected faces (right)

III. EDGE DETECTION

1. Reduce high frequency noise from the image with a Gaussian blur

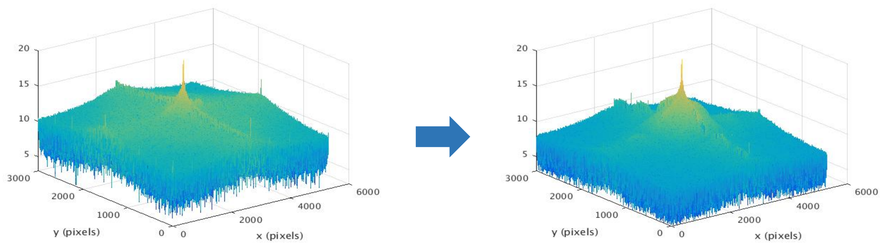

Fig 5. FFT of grayscale noisy image, unfiltered (left) versus filtered (right)

2. Use the Sobel operator to approximate the gradient of the image, then determine the gradient's magnitude and angle at each pixel

$$G = \sqrt{{G_x}^2 + {G_y}^2}$$

$$\theta = \arctan\frac{G_x}{G_y}$$

3. Classify edges by suppressing non-maximum pixels along the gradient direction with a spatial tolerance threshold

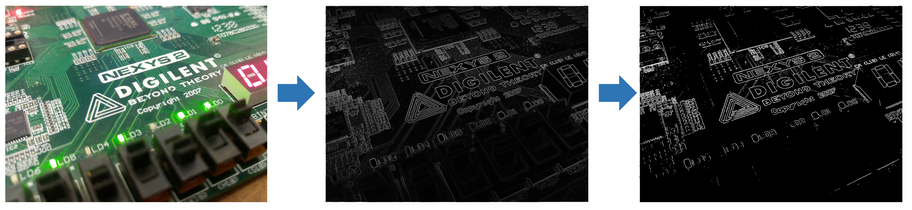

Fig 6. Original image (left), Sobel filter output (center), final edges (right)

IV. MOTION DETECTION

1. Calculate a pixel-by-pixel difference map from the edges of two frames, applying movement tolerance thresholding in the process

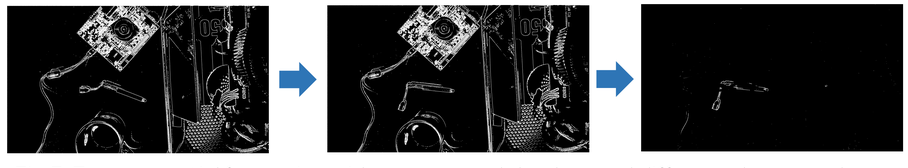

Fig 7. Two sequential frames in a video stream and the detected difference between them

2. Estimate the motion region from a difference density map

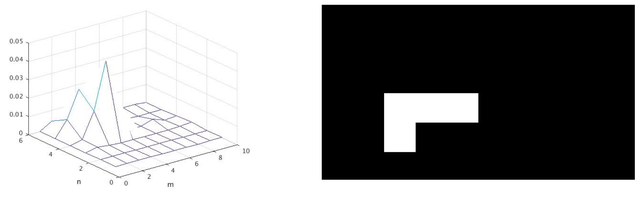

Fig 8. Spatial density map (left), motion area estimation (right)

V. GPU SPEEDUP RESULTS

-The NVIDIA CUDA framework was used to run the parallel implementations of our algorithms.

-The results below were obtained on a **Intel Core i5 4200U** CPU and an **NVIDIA GeForce GTX 860M** GPU.

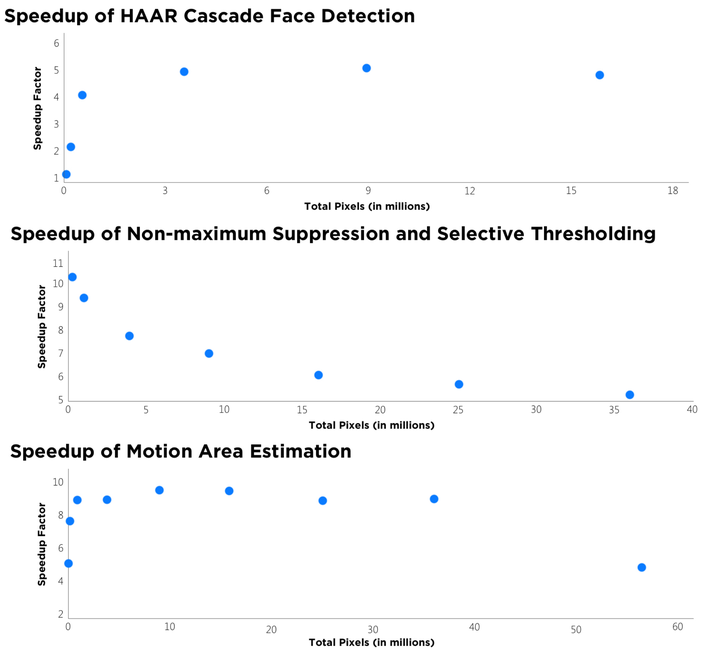

Fig 9. CUDA facial recognition (top) consistently achieved 1.1 to 5.1x speedup. CUDA edge detection (middle) achieved 5.2 – 10.3x speedup. CUDA motion tracking tracking (bottom) achieved 4.2– 9.5x speedup.

VI. CONCLUSIONS

-In edge detection, as the computation size becomes larger, the speedup decreases, an unexpected trend. However this could be due to the memory overhead of running multiple parallel tasks in sequence.

-Overall, the results demonstrated that GPUs offer a powerful solution to the rising data requirements of modern computational and signal processing problems. Despite any trends or outliers, the GPU implementations consistently were faster than the CPU counterparts.

-Potential limitations include: CUDA memory optimization and usage of 3rd party libraries (especially with facial recognition).

VII. ACKNOWLEDGEMENTS

-We would like to thank CJ Barberan and Richard Baraniuk for their guidance and support in this project.