GPU Acceleration of Edge-Based Motion Detection and Facial Recognition with NVIDIA CUDA

Emilio Del Vecchio, Kevin Lin, Senthil S. Natarajan
Department of Electrical and Computer Engineering, Rice University, Houston, TX 77005

I. INTRODUCTION

-Graphics Processing Units (GPU's) are powerful tools for parallelized computations on large data inputs, achieving massive speedups over a serial CPU

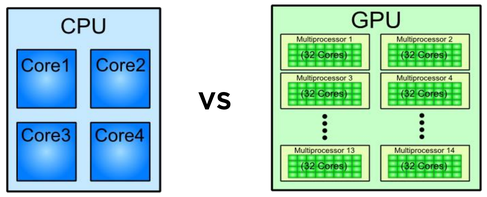

Fig 1. Massively parallel GPU architecture compared to a multicore CPU

-NVIDIA CUDA (Compute Unified Device Architecture) exposes simple C-based APIs for GPU computation

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

-Existing computer vision algorithms, when parallelized with CUDA, run several times faster than on a CPU

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.

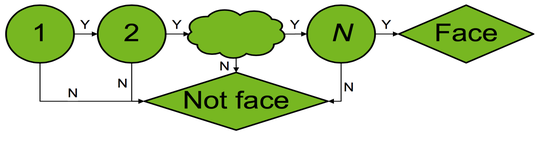

Fig 3. Cascade Classifier training rejects most non-face regions early.

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

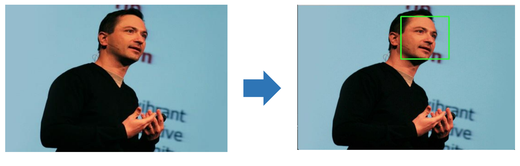

Fig 4. Original sample image (left), detected faces (right)

III. EDGE DETECTION

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

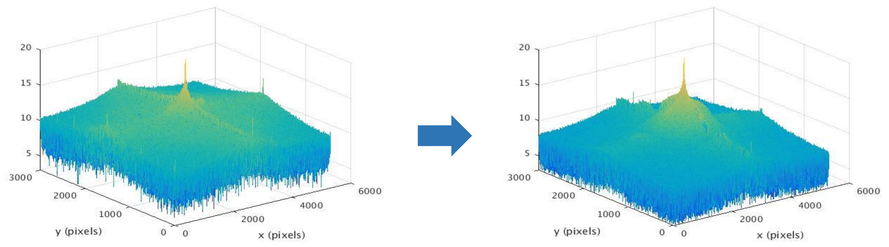

Fig 5. FFT of grayscale noisy image, unfiltered (left) versus filtered (right)

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

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

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

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

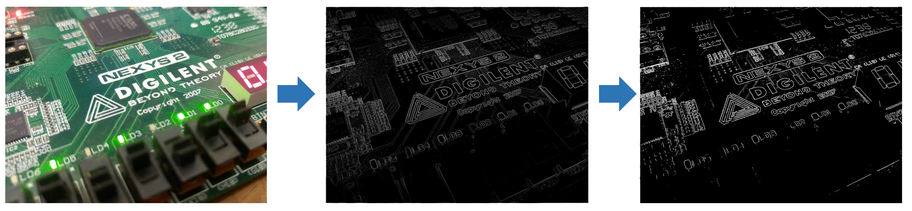

Fig 6. Original image (left), Sobel filter output (center), final edges (right)

IV. MOTION DETECTION

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

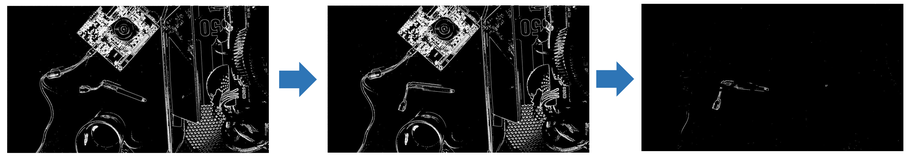

Fig 7. Two sequential frames in a video stream and the detected difference between them

2. Estimate the motion region from a difference density map

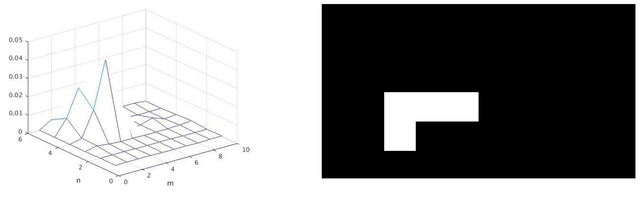

Fig 8. Spatial density map (left), motion area estimation (right)

V. GPU SPEEDUP RESULTS

-We implemented our algorithms in C using the CUDA framework. The results below were obtained on a **Intel Core i5 4200U** CPU and an **NVIDIA GeForce GTX 860M** GPU.

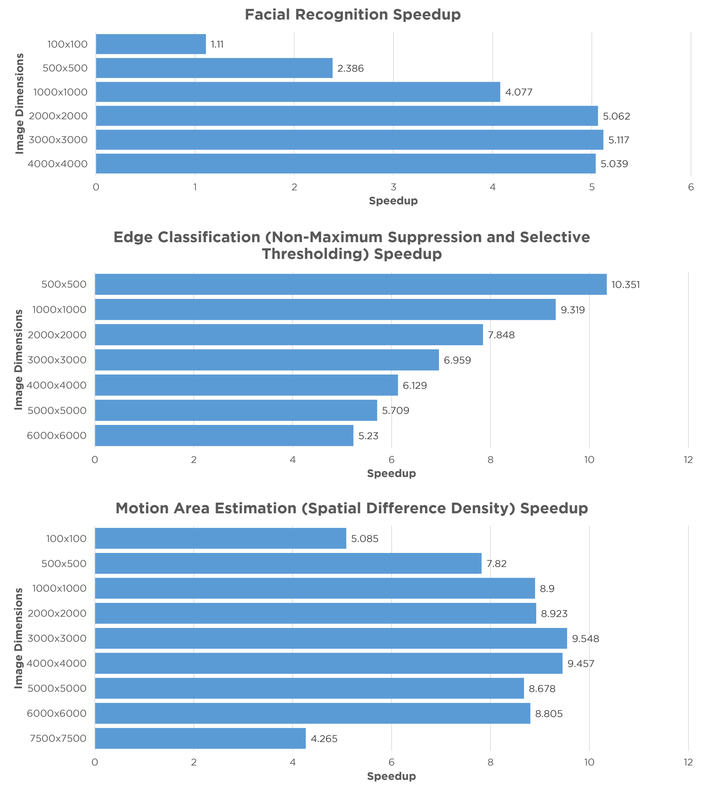

*Fig 9. CUDA facial recognition (top) consistently achieved **1.1 – 5.1x** speedup. CUDA edge detection (middle) achieved **5.2 – 10.3x** speedup. CUDA motion detection (bottom) achieved **4.2 – 9.5x** speedup.*

VI. CONCLUSIONS

-Generally, our algorithms demonstrated increasing speedup with pixel count because of the overhead associated with launching kernels on small input sizes.

-Deviations from this trend result from large memory footprints and lack of consideration for idle thread times.

-In all cases, GPU parallelization resulted in increased throughput, demonstrating that GPU computation is a viable and fast solution for signal processing on big data.

VII. ACKNOWLEDGEMENTS

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