Image Processing Algorithm Test Bench

BYU CAPSTONE

BACKGROUND

Computers perform image classificiation differently than humans. Humans notice difference in large scale features that help us differentiate between images of a cat and a dog. In contrast, computers perceive images as a set of individual pixels of various color values. To help extract meaningful information from the images, "texture features" are computed using the pixel values.

Since existing code is not fast enough, Sandia National Laboratories tasked our team with creating with three versions of standard Haralick algorithms: a single-threaded version for a generic CPU, a multi-threaded version for a generic CPU, and a multi-threaded version written in CUDA for execution in a graphics processing unit (GPU).

PROJECT OBJECTIVE

Design, build, and test the performance of a single-threaded, multi-threaded, and CUDA implementation of fourteen Haralick texture feature algorithms. Each implementation should achieve measurable speedup relative to publicly accessible versions of the algorithms.

TEXTURE FEATURES

Computed Haralick texture features describe aspects of images like contrast,correlation, and entropy as numeric values. These values are used by researchers for image classification,pattern recognition, and machine learning.

KEY SUCCESS MEASURES

- Single-Threaded C++ implementation at least as fast as Python reference code. The reference code is unoptimized publicly accessible code.
- Multi-Threaded Implementation at least 5x faster than reference.
- CUDA GPU Implementation at least 20x faster than reference.

HARALICK TEXTURE FEATURES

1. Angular Second Moment
2. Contrast
3. Correlation
4. Sum of Squares (Variance)
5. Inverse Difference Moment
6. Sum Average
7. Sum Variance
8. Sum Entropy
9. Entropy
10. Difference Variance
11. Difference Entropy
12. Information Measures of Correlation 1
13. Information Measures of Correlation 2
14. Maximal Correlation Coefficient

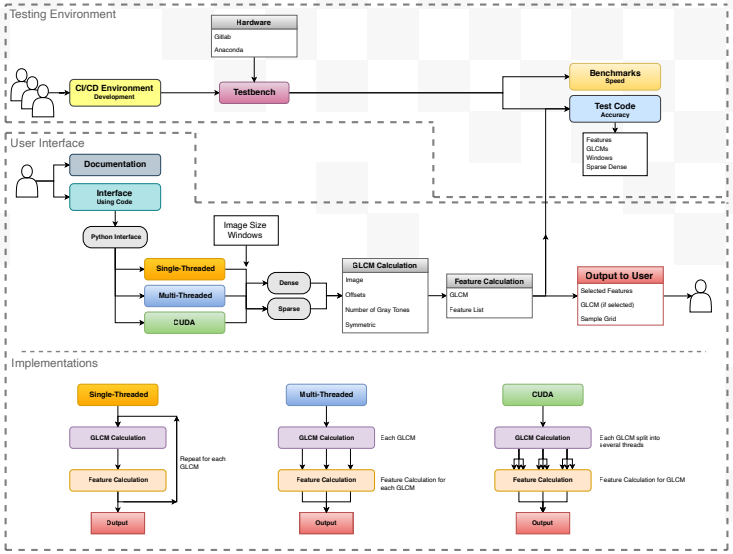
CUDA

CUDA is a C++ like parallel computing platform programming designed to utilize fast GPUs (Graphics Processing Units), like those used in high-end graphics cards. GPUs perform a large number of similar or relatedcalculations simultaneously. This is useful for large batches of images.

WINDOWING

Texture features vary across an image. Our code can calculate features in user-specified window sizes across an image, which reveals how texture varies across an image. Windowing enables speedup capabilities in multi-threaded and CUDA implementations since each window can be computed simultaneously.

PROJECT ARCHITECTURE



TESTING ENVIRONMENT

Our code is stored on Gitlab, an online platform which runs of updated code against a testbench. Our testbench includes small and large images with a variety of texture differences.

USER INTERFACE

Sandia wanted our final product to be easy to use and understand by a first-time user so we included very thorough documentation of our code. A (slower) Python version is also included for ease of access.

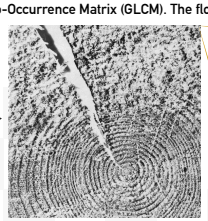
IMPLEMENTATIONS

- Single-threaded version suitable for a simple microprocessor.
- Multi-threaded version can operate on a variety of computers or servers, and is significantly faster than the single-threaded version.
- The CUDA version runs on a Graphics Processing Unit (GPU).

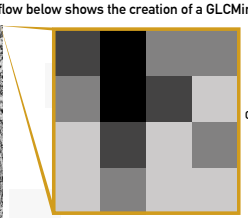
GLCM CALCULATION

The first step is creating a Gray-LevelCo-Occurrence Matrix (GLCM). The flow below shows the creation of a GLCMin a 4 x 4 window.

The Haralick texture features can be calculated onany. This is a color wood grain image.



The color image is first computed to grayscale by averaging color values.



A "window" of 4 x4 pixels is moved over the grayscale image. Gray values are converted(quantized) to 4 distinct values gray to black, yielding a 4x4 GLCM matrix.

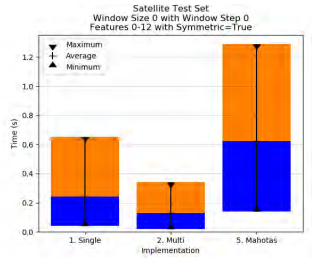
GLCM Calculation

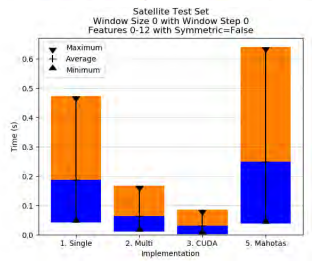
$$\begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 2 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 2 & 1 \end{bmatrix}$$

The GLCM is related looking in the rightward direction. Each row is a pixel value, the first row being black and the last being light gray. Each column shows how many rightward neighboring pixels the row has. In this example, there are two black pixels with right neighbors, so row one totals to two. One has a dark gray neighbor to the right, so the second column has a one. The other has a gray neighbor to the right, so the third column has a one, etc.

GRAPHS

The graphs on the right report test results. Mahotas is the reference open-source implementation. These graphs show our speed without windowing. Windowing speeds our implementation. Our versions are significantly faster than open-source code versions.





RESULTS

Our single-threaded implementation (Single) is 2.5x faster than the reference version (Mahotas). Our multi-threaded (Multi) achieved a speedup of about 4x, and our CUDA achieved a speedup of 8x. As the amount of windows per image increases, the relative speedup of the multi-threaded and CUDA versions also increases,

Sandia National Laboratories

2019-2020 Sponsor

IMATRIX

Team members from left to right: Stephen Summers, Gregg Cowley (coach), Peter Sawyer, Chandler Johns, Ben Ogles, Bryan Redd, Alex Bowcut.