Image Processing using parallelization techniques and their performance yields

Authors: Mohammad Khan, Tung Pham, Johann Zhang

Motivation:

Processing of images/videos is a data intensive task. Due to its enormous volume nature, to perform transformation on a single image and translate that to a series of images without any optimization or distribution would consume lots of resources. A single transformation of a single image would take a significant amount of time and when doing image processing as in sequential approach, we would have to loop through each element in a 3D matrix to do this. This degradation in performance would cascade into a significant amount of time required to finish processing a series of images and especially videos, where we treat this as a continuous series of images, on a single local machine. Therefore, to mitigate this performance issue, a more robust approach is needed. Given that we are working in a limited resource environment, it is crucial to be able to perform analysis and transformation of image/videos data on a single machine that can produce acceptable results in acceptable time. In this project, we're going to attempt to redevelop the parallelized image processing which primarily focuses on image transformation.

Background:

In computer vision, we have to work with multiple transformations of images and videos. This is to either preprocess the data for Machine Learning model for object detection, objects classification, or with the latest trends in generative AI, to generate images or videos. Another application of image transformation is that it is crucial for classification tasks where we have a limited number of labeled data and there's no way to automate this. With image transformation, we could therefore increase the number of labeled data that we have and therefore allow better prediction and better accuracy for tasks like Transfer Learning, Zero-shot learning, semi-supervised learning, etc.

There are already multiple frameworks and libraries that do matrix multiplication in parallel, notably the Numpy library of Python (which offload the vector calculation to C pre-compiled programs), or the library Eigen of C++. All of these libraries will optimize the performance of the matrix calculation by utilizing either OpenMP or other libraries that interface the parallelism of the program. There is currently a popular interface, CUDA, that allows offloading the parallelism to the GPU which allows even faster performance.

We also have a library specifically for Computer Vision tasks which is OpenCV. However, for matrix manipulations and calculations, it still uses Numpy under the hood which mentioned before that does tasks in SIMD fashion.

Related work

Parallel processing in image processing has been a significant area of research in the modern world due to the rise of multicore systems and increased need for image manipulation. This leads us to not only imagine manipulation techniques but also their optimization. One way of optimization is using parallelization techniques by leveraging the OpenMP API to run images processing methods in parallel. In a research paper written by Slabaugh et al. (2008), the papers show how OpenMP can be used to common image processing techniques like image warping, morphological transformation, and median filtering, resulting in a substantial speed up compared to performing them sequentially. This work highlights some of the challenges and simplicity of using the openMP API for scalable parallelization in image processing tasks on modern day multicore systems.

In another study by Huang et al. (2012), the paper examines the application of OpenMP for medical ultrasound image processing algorithms on multicore embedded DSP systems. The paper highlights the importance of efficient workload partitioning and workload balancing which is important in reducing computational overheads and working with parallelization techniques. It also emphasizes memory and data access overhead which is critical in embedded systems for real time processing in ultrasound imaging. The study shows how effective data placement and cache configurations in combination with OpenMP directives can improve significant performance gains on embedded systems which provide useful insights on CPU parallelization techniques for image processing.

These two papers focus on CPU based parallelization techniques and their knowledge serve as a foundation to base our project off of to employ multi core parallelization techniques on image processing.

Planned work

For the next few weeks, we'll first experiment with images to matrix conversion in C++. By doing this, we'll be able to represent images as a 2D array with 3 RGB channels. Then, we'll develop basic parallelized matrix operations like matrix multiplication or matrix additions or summation. This acts as the core of our image transformation. For our next task, we'll develop functionalities like rotate, bend, apply filters, Gaussian blur, sharpen, color transformations, or even do Fourier transform of the image. As we've already done Gaussian Blur, this can take us to a convolution of the image with a given kernel and therefore allows us to understand more of the convolution done in the Convolutional Neural Network. A potential addition that we can extend was to apply image stitching of multiple related images to create a panorama final result. However, this is

certainly a rather complex task and is not promised to be completed given the time constraints but rather an exploration on this topic.

Citations

1. Slabaugh, G., Boyes, R., & Yang, X. (2010). Multicore image processing with OpenMP [Applications Corner]. *IEEE Signal Processing Magazine, 27*(2), 134–138. https://doi.org/10.1109/MSP.2009.935452
2. Huang, L., Stotzer, E., Yi, H., Chapman, B., & Chandrasekaran, S. (2012). Parallelizing ultrasound image processing using OpenMP on multicore embedded systems. *2012 IEEE Global High Tech Congress on Electronics*, 131–138. https://doi.org/10.1109/GHTCE.2012.6490139