Convolutional Image Processing

Yousif Khaireddin
William Martin
Zhuofa Chen
Ihor Leshchyshyn

Electrical and Computer Engineering
Boston University

Abstract

With developments in computer vision, image processing has become more in demand everywhere. At its root, image processing can be thought of as a convolution between an image and a kernel. Convolutions are extremely powerful and are also extremely prevalent in machine learning. They have given rise to a new architecture called convolutional neural networks. Due to this, there is a huge demand for more efficient and faster convolution implementations. Throughout this project, we implemented a serial 2D convolution. We then study the effects of different commonly used filters such as blurring and edge detection. Next, we rewrite our convolution using 3 different parallel implementations openACC, openMP, as well as C++ threading and compare their performance. The best performance is attained using openMP.

Introduction

With the development of deep learning and other computer technologies, computer vision is playing an important role in our life [1]. Among the techniques used in computer vision, image processing is the most commonly used technique for manipulating and modifying images [2]. There are two types of image processing, analogue image processing and digital image processing. Analogue imaging processing mainly focuses on processing analog signals (electrical signal) in two dimensional images like photographs and medical images. Digital image processing, on the other hand, is concerned with manipulating the digital pixels in images using computers. Nowadays, most deep learning models and proposing algorithms focus on processing digital images. Therefore, in this study, we explore digital image processing techniques [3].

Since AlexNet got noticed in 2012, convolutional neural networks and convolutional image processing have gathered more and more attention [4]. These networks are currently used in countess applications due to their efficient and powerful image processing abilities [4]. Convolutional processing is achieved by moving a kernel (filter) across an image and calculating the elementwise product between filter and overlapped region. Each pixel in the resulting image is calculated by this sum of the product. The filter is then shifted over by some stride, and the next element is calculated. This process, seen in Figure 1, can be easily scaled up and boosted using parallel computing techniques [5].

In order to manipulate digital images, filtering is an important imaging processing technique. Many commonly used filters in image processing include, denosing, sharpening, blurring, edge detection etc. Figure 2 shows an example of a blurring filter. The left is the original picture, the right shows the effect of blurring. One important thing to notice is that these filters can be thought of as convolution kernels.

In this work, we mainly apply several parallel computing methods to convolutional imaging processing and compare the performance of these methods. A serial computing method is used as an upper edge of the time consumed during this computational process.

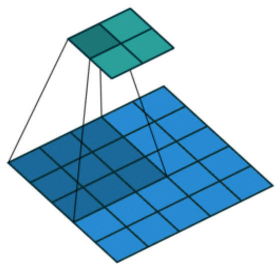


Figure 1: Image Convolution [6]

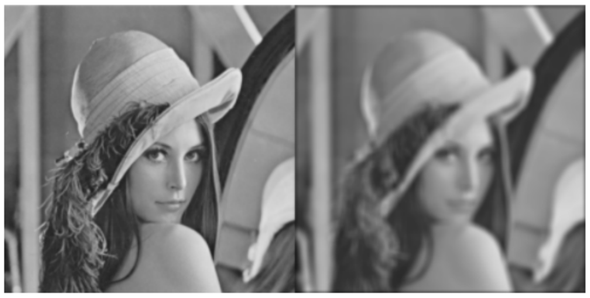


Figure 2: Blurring [7]

Workflow and Results

To properly understand convolutions, we initially start by implementing a simple 2D serial algorithm and applying it on an image using several different commonly used filters [7-8]. The resultant images can be seen in Figure 3. Here we place the original image on the left and apply four different filters onto it. Each filter is shown underneath its resulting image. The effects of the filters are mostly evident in box blurring, unsharp masking, and edge detection where the filter's output is visually different from the original image and performing the expected function. As for gaussian blurring, the resultant image does not look too different from the original, we have verified that the pixel values are different. This could be fixed easily by increasing the standard deviation of the gaussian matrix to apply the filter more aggressively [8].

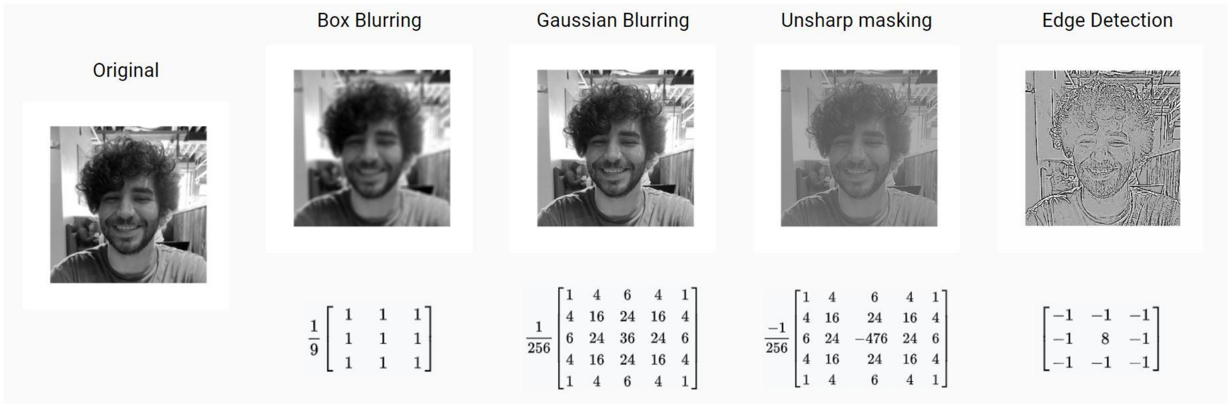
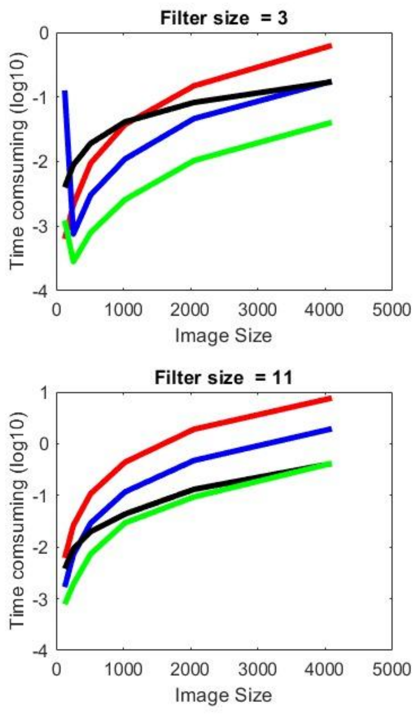

Box Blurring
Gaussian Blurring
Unsharp masking
Edge Detection
Original
$\frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$
$\frac{1}{256}\begin{bmatrix}1&4&6&4&1\\4&16&24&16&4\\6&24&36&24&6\\4&16&24&16&4\\1&4&6&4&1\end{bmatrix}$
$\frac{-1}{256}\begin{bmatrix}1&4&6&4&1\\4&16&24&16&4\\6&24&-476&24&6\\4&16&24&16&4\\1&4&6&4&1\end{bmatrix}$
$\begin{bmatrix}-1&-1&-1\\-1&8&-1\\-1&-1&-1\end{bmatrix}$

Figure 3: Image Processing Examples

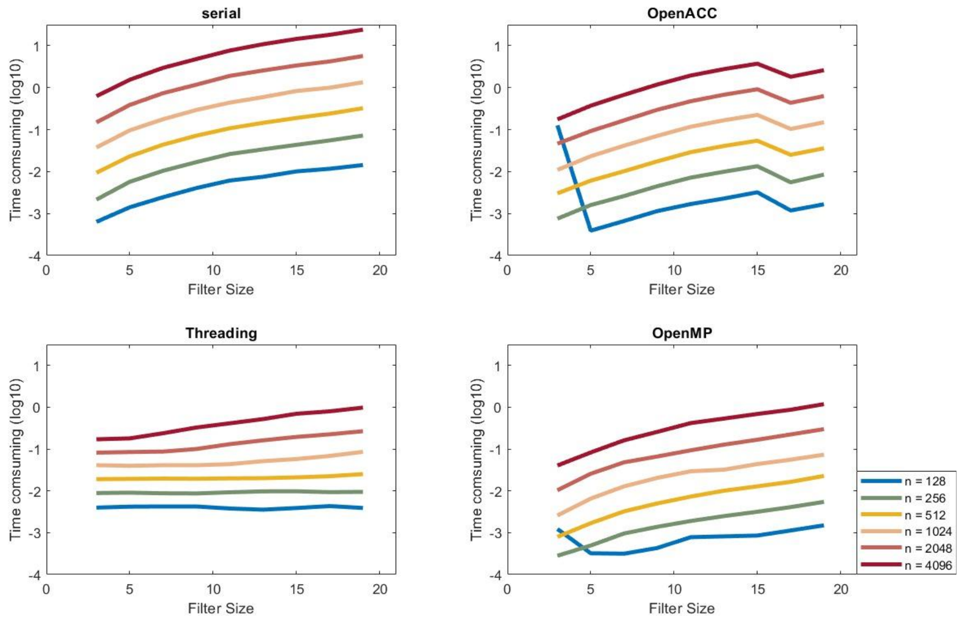
Performance Results and Challenges

We explored three additional implementations of our convolution program beyond the serial version, of which included OpenACC for parallelization on the GPU, as well as the C++ standard threading library and OpenMP for CPU-based concurrency. We can see, based on the results in Figure 4, that the OpenMP implementation appears to be the most performant, performing better than or on par with the C++ threading implementation, at least up until an Image size of 4000 with a filter size of 11 and 15. Interestingly, the GPU implementation with OpenACC performs significantly worse than either CPU threading implementations in all cases, even at large image sizes, suggesting that further tweaking of the OpenACC code is required to achieve optimal performance.

The development process for our image convolution program was without any real challenges in terms of the actual application of image filters. We did, however, run into problems with the ability to profile our program when using the C++ standard threading library on the BU SCC. It appears that due to the way the SCC allocates hardware resources when combined with our threading implementation, we weren't able to profile kernel sizes larger than 32. No such errors were encountered when profiling on our personal machines.



(a)



(b)

Figure 4: Performance Comparisons

Conclusion

We have introduced simple and computationally efficient methods for 2D convolutions. Our methods are aimed towards problems with different image and kernel sizes. The methods combine the advantages of simplicity and efficiency. These methods are straightforward to implement and use. The tests confirm our intuition about the performance of different methods. Overall, we found these implementations robust and well-suited to a wide range of real-world problems in computer vision and machine learning.

We empirically showed that our methods could be used in many real-time systems which require fast image processing (e.g. self-driving cars). We also propose to implement hybrid models (e.g. OpenMP + MPI). It would be interesting to integrate our methods into real-world problems. While we focused our implementations on simplicity and robustness, we believe that better results can be achieved using the hardware's low-level features.

References

[1] Forsyth, D.A. and Ponce, J., 2012. *Computer vision: a modern approach*. Pearson,.
[2] Ekstrom, M.P., 2012. *Digital image processing techniques* (Vol. 2). Academic Press.
[3] Razzak, M.I., Naz, S. and Zaib, A., 2018. Deep learning for medical image processing: Overview, challenges and the future. *Classification in BioApps*, pp.323-350.
[4] Krizhevsky, A., Sutskever, I. and Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, *25*, pp.1097-1105.
[5] LeCun, Y., Bengio, Y. and Hinton, G., 2015. Deep learning. *nature*, *521*(7553), pp.436-444.
[6] Dumoulin, V. and Visin, F., 2016. A guide to convolution arithmetic for deep learning. arXiv preprint arXiv:1603.07285.
[7] Ai.stanford.edu. 2021. Available at: Stanford - Image Filtering
[8] En.wikipedia.org. 2021. Kernel.Available at: Kernel (image processing)