APMA E4302

Strip and Grid-Based Parallel Convolution of Images using MPI

Kiran M Sreenivasa

December 17, 2021

Abstract

Convolutional Neural Networks are commonly utilised in the field of computer vision and image processing. The convolution operations involve a significantly large number of calculations and would therefore benefit from parallelization techniques. In the present work, the convolution operation has been parallelized via distributed memory parallelism using MPI. The data was divided among the processes by two methods, strip and grid-based domain decomposition. The effect of process-count on the speed-up was analysed for different image sizes and kernel sizes. The anomaly in speed-up trends has also been discussed in this work.

1 Introduction

In the field of AI, Convolutional Neural Networks or CNNs have proven to be the most promising network architectures for computer vision and image processing. These network architectures perform the mathematical operation of convolution between the image and a kernel to extract distinct features from an image as well as sharpen, blur, emboss images. The kernel is a matrix of whose size and values determine the transformation effect on the image. The process of convolution is performed as follows: the kernel is placed over a part of the image; a weighted sum of the kernel values and corresponding image-matrix values is calculated and stored as the value of the central pixel in a new image matrix. The kernel traverses over the entire image matrix and all image channels while always remaining within the confines of the matrix dimension. This implies that the dimensions of the final matrix are reduced in comparison with the original image. For example, a convolution performed on a 300x300 image using a 3x3 kernel results in a 298x298 image. To account for this, the input matrix is generally padded with 0's or 1's along the periphery.

In mathematics, convolution is a mathematical operation on two functions (f and g) that produces a third function $(f * g)$ that expresses how the shape of one is modified by the other. It is defined as the integral of the product of the two functions after one is reversed and shifted. It can be represented as:

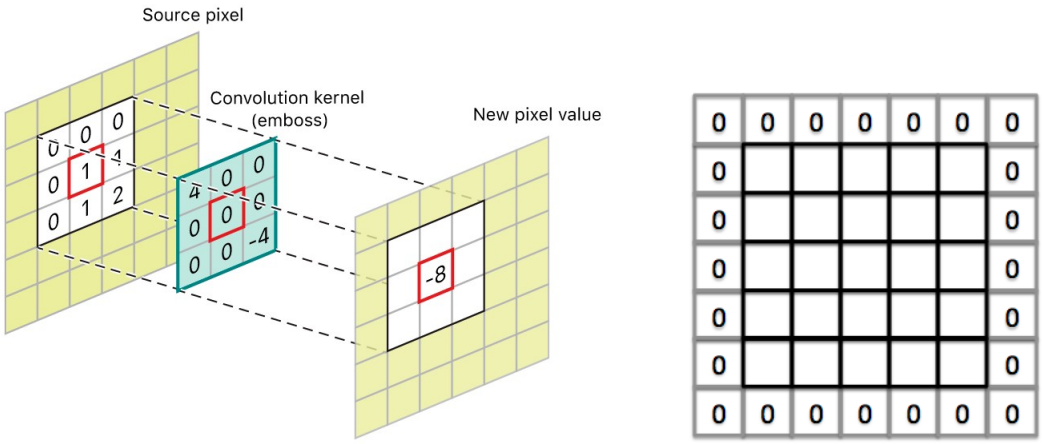
$$(f * w)(t) = \int_{-\infty}^{+\infty} f(\tau) w(t - \tau)\, d\tau \tag{1}$$

With respect to image convolution, the above expression can be simplified as follows:

$$g(x, y) = w * f(x, y) = \sum_{dx=-a}^{a} \sum_{dy=-b}^{b} w(dx, dy) f(x + dx, y + dy) \tag{2}$$

where $g(x, y)$ is the filtered image, $f(x, y)$ is the original image, w is the kernel.

CNNs use lesser parameters than conventional feed-forward neural networks because they utilise the same kernel over the entire image. However, the process of performing convolutions over the entire image involves thousands of calculations and can therefore be sped up using distributed-memory parallelism techniques. The work of Park, J., Kwon, D., Choi, B.W. et al. [1] focuses on dividing an input image into four overlapping sub-images and feeding each segment into into a fully convolutional neural network (FCN), following which the segmented results were combined into a single image. The study by Xu, Jing et. al. [2] focuses on the domain decomposition method for de-blurring of high resolution images using the blur operator. Observations revealed that the efficiency of the proposed method in reducing runtime increased as the image size increased. Yu, Shuling et al.[3] implemented a parallel convolution algorithm wherein each process performs convolutions on a strip obtained by splitting the input image. Therefore, parallelizing image convolution has proven to be one of the most effective strategies in speeding up the operation. In the present work, the image convolution algorithm was parallelized and the influence of core-count on speed-up was analysed. To gain a broader perspective, scalability tests were performed using different image sizes and kernel sizes. A comparative analysis of the two decomposition methods has also been discussed.



(a) (b)

Figure 1: (a) The convolution operation (b) Zero-padding of the input image matrix

2 Methods

2.1 Algorithm

In the present work, distributed-memory parallelism was implemented using MPI (from mpi4py) on Python platform. For the domain decomposition, the image matrix data was split among the processes using two different methods:

a) Strip-based decomposition: the image matrix is divided into horizontal strips among the processes and each process performs convolutions on its particular segment. This method works with any number of specified processes and requires more message passing.

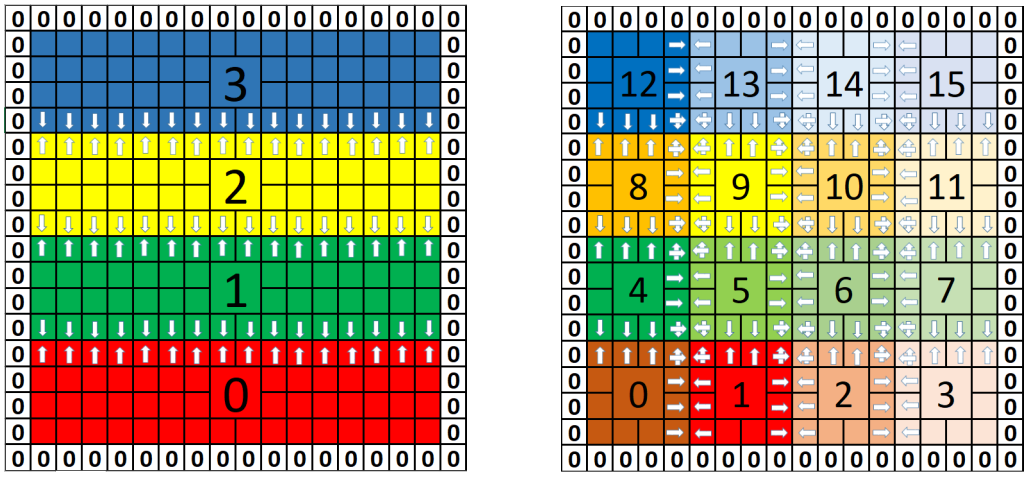
b) Grid-based decomposition: the image matrix is divided into rectangular grids. In the present work, this method works only if the specified number of processes is a perfect square. This method involves relatively less message passing.

The splitting was done after the image-matrix was zero-padded. The size of the pad depends on the kernel size, given by

$$pad = \frac{size - 1}{2} \tag{3}$$

In all cases, the kernel traverses by only one step in any direction. In other words, the kernel stride is 1.

In both methods, performing convolutions at the edges of these segments requires additional information from the surrounding segments belonging to other processes. In strip-based decomposition, every rank other than rank-0 sends a matrix containing the last 'pad' number of rows in its segment to (rank+1), every rank other than the last rank sends a matrix containing the first 'pad' number of rows in its segment to (rank-1). In addition to this, the grid-based parallelism requires processes to send additional information left and right. Once every process has completed its convolution operation, all convoluted segments are gathered using *comm.gather*() command and accordingly placed at appropriate locations in a new matrix to obtain the final processed image.



(a) (b)

Figure 2: Representation of message passing in (a) Strip-decomposition (b) Grid-decomposition (with process rank included)

2.2 Setup

The code for both decompositions was implemented on Python3 platform using mpi4py. The .jpg images were imported onto the code using Python Pillow (PIL) Library. The scalability tests were performed on Google Cloud VM Instance running Ubuntu 18.04 LTS. The specifications of the hardware have been specified in Table 1.

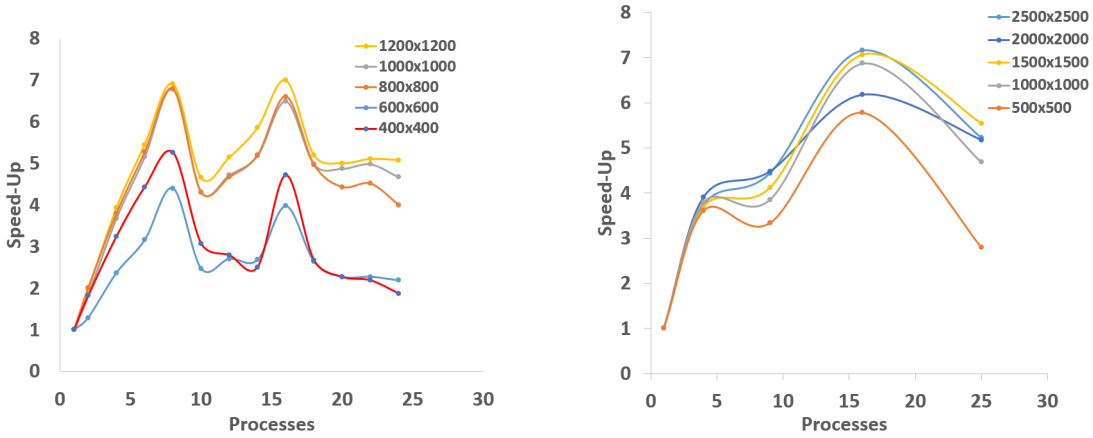
For the initial scalability tests, a 1000x1000 image and 3x3 kernel were used in both decompositions and the effect of process-count on speed-up was observed. The number of CPUs used for strip-decomposition were 1,2,4,6,8,. . . .,24 and 1,4,9,16 and 25 for the grid-decomposition. The

Table 1: System Specifications

Architecture	x86_64
CPU(s)	32
Thread(s) per core	2
Cores per socket	8
Socket(s)	2
Model Name	Intel(R) Xeon(R) CPU @ 2.30GHz
L1d cache	32K
L1i cache	32K
L2 cache	256K
L3 cache	46080K

next perfect square is 36, which is greater than the available processes. In the 2nd set of tests, 2 sets of 5 different images were used, with sizes varying from 400x400 to 1200x1200 for strip-method and 500x500 to 2500x2500 for grid-method. In the 3rd scalability test, a grid-based convolution was performed on a 1000x1000 image with filter sizes varying from 3x3 to 15x15. Each case was executed 5 times and the average of these execution-times was taken as the final run-time for that particular case.

3 Results and Analysis

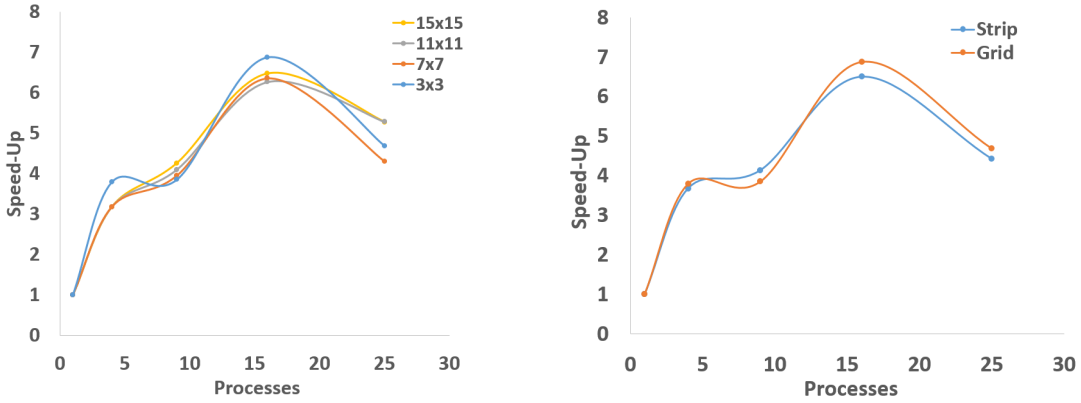


(a) (b)

Figure 3: Scalability tests for (a) Strip-Decomposition and (b) Grid decomposition

Figure 3 (a) illustrates the speed-up charts for 5 different image sizes. There exists a similar trend among all 5 cases, but the magnitude of speed-up increases with image size. With 8 processes, the speed-up is 5 for 400x400 image convolution and 7 for 1200x1200 image convolutions. Figure 3 (b) exhibits a similar pattern with bigger images reaping more benefits of parallelization. With smaller images, the latency from inter-process message passing is of comparable magnitude to the computational time per process. This margin, however, widens with increasing image size. After a certain image size, the speed-up magnitude is similar as seen in cases 800, 1000 and 1200 for strip-method and 1000,2000 and 2500 for grid-method.

In all cases, the speed-up charts exhibit an anomaly from 8 processes to 16 processes. The speed-up decreases after 8 processes and peaks again at 16 processes before permanently decreasing again. Since the scalability tests for strip-method was performed for 13 different process-counts in comparison with 5 process-counts for grid-method, this anomaly is more pronounced in the former. Since 8 is not a perfect square, the grid method was not run using 8 processes. However, there is a local maxima at 4 processes. Since peaks occur at multiples of 4 or 8, a possible explanation for this anomaly lies in either the CPU architecture or the inter-process communication. This can be further investigated by controlling the aforementioned communication via explicit commands in the program code.



(a) (b)

Figure 4: (a) Scalability Tests for Grid-Decomposition using different kernel sizes (b) Strip Vs Grid comparison

Figure 4 (a) illustrates the speed-up charts for different kernel sizes. Apart from minor variations, the graphs are almost identical to each other. As the kernel size increases, the total number of calculations proportionally increase for both the serial and parallel cases while the message-passing latency remains the same. Therefore, the $t(message - passing) + (\frac{t(serial)}{t(parallel)})$ remains constant throughout the 4 cases.

Figure 4 (b) depicts a comparative analysis of the two decomposition methods using a 1000x1000 image and 3x3 kernel. This scalability test was performed using 1,4,9,16,25 processes for the purpose of homogeneity. Both methods exhibit very similar performances. The hypothesis was that the strip-method would initially perform better due to lesser message-passing latency and as the number of processes/image size increase, the grid method would perform better since the size of the rows shared among the processes is smaller, thereby imposing lesser burden on cache memory. This hypothesis is a reasonable explanation for the minor differences in speed-up between the two methods, particularly when the grid-method starts performing better after 12 processes.

Let N be the width of an NxN image, P be the number of processes and K be the width of a KxK kernel. Since each method performs the same number of convolutions, the total complexity is $O(N^2K^2)$. For the strip-method, the number of message passes by each process is NK, the total complexity of message passing is $O(PNK)$. For the grid-method, each process sends $2NK\sqrt{p}$ messages and the total time complexity is $O(2NK\sqrt{p})$. On analysing the two complexities, it is seen that $t(grid) \propto 2\sqrt{p}$ while $t(strip) \propto P$. As the number of processes increases, the influence of message-passing on overall complexity is more pronounced in the strip-method. This can be

confirmed by the speed-up chart in Figure 4 (b), where the grid speed-up is marginally better than the strip speed-up for larger processor-counts.

It can be noticed that a different set of images were used for the two decomposition methods. Beyond a size of 1200x1200, the strip-method experienced a deadlock, wherein the python3 code runs indefinitely with no termination. The grid-method experienced no such deadlocks and performed successfully for larger sizes. This issue could be related to cache memory, since the processes in the strip-method send and receive bigger rows, thereby requiring more memory. Further investigations are required for identifying the root cause.

4 Conclusion

This study involved the implementation of distributed memory parallelism to speed up the convolution of images using two methods of domain decomposition, namely strip and gridded. Numerous scalability tests were carried out to analyse the performance of the two decomposition methods, influence of image/kernel size on the speed-up. Image size influenced speed-up to a certain threshold while kernel size had minimal impact. Certain anomalies in speed-up trend were noticed at process-counts 4,8 and 16, which was attributed to inefficient inter-process communication. The strip-method experienced deadlocks with larger images while the grid-method experienced no setbacks. Further work includes investigating the speed-up anomaly, strip-method deadlock. It would be also beneficial to implement a grid-based algorithm that works for an arbitrary number of processes. Another possible experiment is the implementation of random-grid parallel convolution wherein each process is allotted a random set of pixels to convolute. Such an implementation would eliminate the need for any message-passing and could potentially perform better than strip and grid-method.

The code can be found at:
https://github.com/kiranms98/Parallel-Convolution-of-Images-using-MPI

References

[1] Park, Jinhee; Kwon, Dokyeong; Choi, Bo Won; Kim, Ga Young; Kim, Kwang Yong; Kwon, Junseok (2019). Small object segmentation with fully convolutional network based on overlapping domain decomposition. Machine Vision and Applications, (2019), 707–716. doi:10.1007/s00138-019-01023-x

[2] Xu, Jing; Chang, Hui Bin; Qin, Jing (2014). Domain decomposition method for image deblurring. Journal of Computational and Applied Mathematics, 271(), 401–414. doi:10.1016/j.cam.2014.03.030

[3] Yu, Shuling Snell, Quinn Morse, Bryan. (1998). Parallel Algorithms For Image Convolution.

[4] CS4670/5670 - Introduction to Computer Vision, www.cs.cornell.edu/courses/cs4670/2018sp/.

[5] https://en.wikipedia.org/wiki/Kernel_(image_processing)

[6] https://mpi4py.readthedocs.io/en/stable/

[7] https://github.com/0xnirmal/Parallel-Convolution-MPI