# Optimization of Image Convolution using Parallel Programming

P Gagan Kumar Chowdary
Roll No : 191ME160
National Institute of Technology Karnataka

Setti Naveen Naidu
Roll No : 191ME177
National Institute of Technology Karnataka

G Tejesh
Roll No : 191ME288
National Institute of Technology Karnataka

## I. Introduction

In image processing, convolution is the process of transforming an image by applying a kernel over each pixel and its local neighbors across the entire image. The kernel is a matrix of values whose size and values determine the transformation effect of the convolution process. Convolution has a multitude of application including image filtering and enhancement, image restoration, feature detection, etc.. The number of computations involved in image processing will be large if high resolution images comes into picture. So parallel implementation of convolution will be a better option. Also Parallel implementation of image convolution reducesthe time taken and increases its efficiency. The result of image convolution computed at a pixel point is depend only on the nearby points. So the convolution operation can be efficiently mapped to parallel computers. During convolution, the image

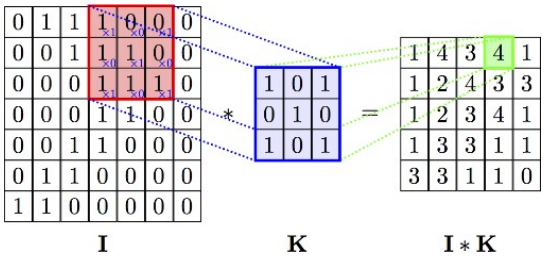


Fig. 1

is partitioned into square blocks and each process is ”mapped” to a specific block. At first, each process convolutes only the inner pixels with a given kernel, and then, each process exchanges with its neighbours its outer pixels to perform a convolution to the remaining pixels. This process is repeated n times.

## II. Related Work

[1] A parallel processing based on CUDA for CT image convolution filter FDK reconstruction. The technique of implementation of CUDA has taken from here and used for the implementation in our program.[2] Knowing how to implement parallel algorithms with multi-core.[3] knowing how to do image convolution and process it.[4] knowing how to convert serial programming into parallel programming with MPI.

## III. Sequential Programming

The sequential programming allows to code the technique called convolution .The execution of the program is done sequentially, thus it is called sequential programming.

### A. Implementation Logic

In the paragraphs below, we use word IMG _MAT , which is also the matrix representation of image in terms of pixel values as in the cells of matrix.

```
image = pixel_values_of_image

for (int i=1;i<=image.rows;i++) {
  for (int j=1;j<=image.columns;j++) {
    image[i][j] = image[i-1][j-1]*kernel[0][0] +
      image[i-1][j]kernel[0][1] + image[i-1][j+1]kernel[0][2] +

      image[i][j]*kernel[1][0] +
    image[i][j]kernel[1][1] + image[i][j+1]kernel[1][2] +

      image[i+1][j-1]*kernel[2][0] +
    image[i+1][j]kernel[2][1] + image[i+1][j+1]kernel[2][2]

    }
}
```

Fig. 2: Logic

In the previous figure (fig 2) you can see, image is a matrix of pixels' values, kernel is the small matrix of dimensions 3x3, which may contain any values, the values in the kernel differs according to the desired output.for example, if the output is to get a blur image, the values in the kernel matrix is as in figure 3.

The center of the kernel matrix correctly overlap with

```
kernel = [
 [1,2,1],
 [2,4,2],
 [1,2,1]
]
```

Fig. 3: kernel

each cell in the IMG MAT, lets say the current cell is IMG_MAT(i,j) ( or image(i,j) ) , then multiplies the values of kernel's cells to respective cells mapped to the IMG_ MAT, after multiplication , the sum of their products are added and stored in another matrix's cell of same position conv_image(i,j).

In this way we have to do the convolution with the kernel matrix.The kernel is moved towards rightwards by one step after the calculation of conv image(i,j).

Preprocessing the images We are preprocessing the image, since we are doing calculations with for each pixel in the image, for edge pixels the calculation becomes complex as they don't have neighbouring pixels, so we used padding here to solve the issue.Padding is adding pixels of value 0 around the image.

### B. Execution

The coding of programs is done in three steps.

1. Reading image
2. Convolution
3. Writing image

**Reading image** The image is taken as input into the program, library called OpenCv is used to read the image.We are taking a color image, it usually have 3 channels with differing height and width (h x w x 3).Currently the program reads an image in BGR format(Blue-Green-Red).

As already mentioned about the logic, for each cell in the matrix computation is done, but the problem raises when the computation is done any cell on the boundary of the matrix, as they don't have cells surrounding them in all directions.This problem is dealt with padding, where an extra layer of cells is added around the matrix with value in each cell as 0.Now, for each cell which is read from image has a pixel around in all directions.The program stores the values of pixel value of Blue channel, Green Channel, Red channel in their respective txt files and save them.

**Convolution** The data from txt files are taken by this program which does convolution and write back pixel values into the txt files.

**Writing Image** The pixel data stored by convolution step is written as image named "conv.jpg" using the same library, Opencv using function "imwrite" .

### C. Concurrent Region

When we calculate the time complexity here, it would be $O(n\hat{2})$ ( i.e., Big-O , rate at which the time for calculation changes ), this is a theoritical calculation, since calculation of conv image(i,j) is done for each cell in matrix image (IMG MAT), and in a nxn matrix there are $n\hat{2}$ cells,

In the figure 2, there are two for loops which is executed sequentially, we can make it run parallely as one cell's computation in IMG_MAT doesn't effect other cell's computation since we are storing it in new matrix.If we can use the hardware present in our local machine effectively, the computation of conv_image matrix can be calculated in lesser time than the sequential part.

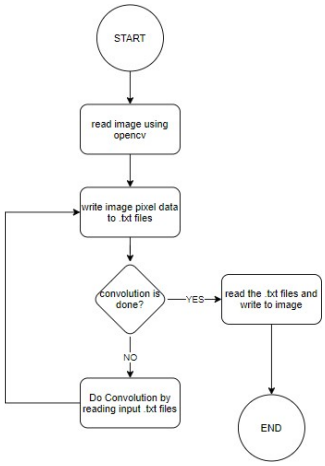


Fig. 4: Flow diagram

## IV. Parallel Programming

Description and methodology about what all the techniques we used, with their names as headings.The sequential part which is executed by three parallel techniques which are OpenMP, MPI, CUDA.

### A. OpenMP

OpenMP is a set of compiler directives or pragmas as well as an API for programs, that provides support for parallel programming in shared-memory environments.OpenMP is a parallel programming approach and is supported by C, C++ and Fortan. Parallel regions are identified as blocks of code by OpenMP that may run in parallel.

OpenMp is used for parallel construction in image convolution as we can observe in fig. 3. Here 'pragma omp parallel' directive is used for creating many threads which are used to divide the program into different branches and executing the code in parallelly. 'numthreads' clause is used to initialize the number of threads. 'pragma omp for' directive divides loop iterations between the spawned threads. 'collapse' clause allows you to parallelize multiple loops in a nest without introducing nested parallelism. 'schedule' clause is used to assign the loop iterations to threads.The static schedule assigns such that all threads get approximately the same

number of iterations as any, and the respective threads can independently determine the iterations assigned to it. In this parallel convolution method we are parallelly multiplying the part of the image matrix with the kernel matrix and and the values are assigned to the convoluted matrix. In this image convolution let us consider the image matrix of the image. The kernel matrix is a 3x3 matrix as shown in fig. 3. Let the size of image matrix be (n x m) where n is the rows and m is the columns. Here the final convoluted matrix size will be (n-2) x (m-2). The values of the convoluted matrix are the sum of the 3x3 part of the image matrix and the kernel matrix. Here there are many 3x3 sub matrices of the image matrix that are to be computed. Hence this is done parallel as of to reduce the time taken.

```
image = pixel_values_of_image
n = total_number_threads
x = (columns-2)(rows-2)/n
#pragma omp parallel num_threads(10)
    {
    #pragma omp for collapse(2) schedule(static, x)
for (int i=1;i<=image.rows;i++) {
  for (int j=1;j<=image.columns;j++) {
    image[i][j] = image[i-1][j-1]*kernel[0][0] +
      image[i-1][j]kernel[0][1] + image[i-1][j+1]kernel[0][2] +

      image[i][j]*kernel[1][0] +
    image[i][j]kernel[1][1] + image[i][j+1]kernel[1][2] +

      image[i+1][j-1]*kernel[2][0] +
    image[i+1][j]kernel[2][1] + image[i+1][j+1]kernel[2][2]

    }
}
}
```

Fig. 5: OpenMP

### B. MPI - Message Passing Interface

MPI is a library specification for message-passing,proposed as a standard by a broadly based committee of vendors, implementors, and users. In sequential execution, the image is read through BGR format sequentially that is first Blue colour of the image is convoluted followed by Green colour then Red colour. Same process is executed for all the 3 colours. So, if we assume execution of each process takes 1 ms then total time taken to execute the 3 colours will be around 1+1+1 = 3 ms which is quite time consuming. So if we parallelize their execution then their execution time gets reduced. In this case, MPI is used.

MPI can be initiated by using MPI-Init (argc,argv) which acts as a message passing interface by passing arguments on command line. They are 0 by default. Then, MPI-Comm-size(MPI-COMM-WORLD,totalnodes) - This function gives total number of processes. First parameter is communicator. Second parameter contains the total number of process. MPI-Comm-rank(MPI-COMM-WORLD mynode) - This function gives the rank of the process, which is 1 less than total nubmer of process. Inorder to parallize those three formats we need 3 process to execute in parallel.

So first we can declare 3 process in MPI execution. Then

```
char c[3] = {'b','g','r'};

vector<vector<int>> pixel_data;

int size,myrank;
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD,&size);
MPI_Comm_rank(MPI_COMM_WORLD,&myrank);
MPI_Status status;

if(myrank==0)
{
        in_filename[11] = c[0];
        out_filename[7] = c[0];
}
else if(myrank==1)
{
        in_filename[11] = c[1];
        out_filename[7] = c[1];
}
else if(myrank==2)
{
        in_filename[11] = c[2];
        out_filename[7] = c[2];
}

MPI_Barrier(MPI_COMM_WORLD);
```

Fig. 6: MPI Process Allocation

alloting each process to each colours that is process0 executes convolution for blue colour, process1 executes for green colour, process2 executes for red colour. Then MPI-Barrier(MPI-COMM-WORLD) is declared since to ensure that all process running above are completed. After allocation, each formats run in separate parallel process and gets convoluted accordingly. After convolution the convoluted data is stored in 3 different files. Finally, MPI-Finalize() is declared to stop the parallel execution. In this MPI process, the total time may take only around 1 ms as assumed above which is a average of time taken by 3 formats. So it is quite less than the sequential execution.

### C. CUDA - Compute Unified Device Architecture

CUDA allows to use Nvidia GPU using programming language C/C++.The developing of programs in CUDA is done in the following way,Host program - there would be Serial code which supplies data for huge/large computations,Kernel Program - there would be parallel code which does computations using GPU. The computed data if required is written back to Serial Code for synchronization.

An execution model in CUDA consists of threads, block, grid where thread performs computation run on Scalar processor.A group of threads form a block with maximum number of threads as 1024.A group of block form grid.The threads, blocks in grid are visualised programmed thinking they can be structured in 3-dimensional way.Each thread, block can be positioned as co-ordinate in space with x,y,z directionsfor each dimension.GPU/CUDA follows SIMD.

As mentioned earlier in the section III (Sequential Programming), Image convolution involves 3 steps, reading image, convolution, writing image.As for part of the code which can be executed in parallel is in second step i.e., convolution.This part is executed in Google colaboratory using CUDA.The pixel data stored in the text file is extracted

as a matrix for each channel(Blue, Green, Red).

*1) Implementation:* The pixel data as a matrix of each channel is of size NxM where N is number of rows, M is number of columns.The 2D array or the matrix which contained pixel data is flattened to a linear array.As for the structure of usage of GPU in CUDA, we used a grid which contains B blocks where B = (NxM/512) and each block has contains 512 threads.This structure is used so that there would be a thread for each cell.Each thread executes computation for each cell in the matrix of image.In parallel part of the program, each thread's ID is computed and then mapped to each cell of the matrix.So now each thread computes for each cell.
The mapping of thread's id to its respective cell in the matrix and vice-versa is listed below.

1)ID = blockIdx.x * blockDim.x + threadIdx.x
2)i = id/columns
3)j = id%columns
4)pos = i*cols + j

In the above list columns is number of columns in 2D array, [1] is calculating the unique ID of each thread according to the block's dimension and thread_id in the block.[2],[3] is row, column of the cell in the 2D matrix. [4] is position of cell in the flattened array of 2D matrix. The number of threads for each block is set as 512 because it is common for images to have one of their dimension around 512 pixels. And also made sure that number of threads formed would be greater and sufficient for all the cells in the matrix.
The linear array of pixel data stored in the Sequential part is copied to GPU global shared memory, this same array is passed to function in which is executed in parallel, where each thread's executed computation for each cell, then stored into the 2D array or matrix in the sequential part of the program with help of mapping linear array to 2D array mentioned as [2],[3] in the above list.With this the parallel part of the program's task is completed.Now,completion of convolution the pixel data after convolution is written back to the text files which is sequential part of the program's task.

## V. Results and Analysis

With convolution we made the program to blur an image , we used the kernel matrix as showed in figure 3.

We also made an execution-time analysis of convolution in each technique.We took the same image of elephant mentioned in figure 7, but of different resolutions.We used images of size nxm , which means n rows, m columns of pixels.We used 50x50, 100x100, 300x200, 500x500, 700x700, 1024x1024, 2048x2048.We noted the execution-time of convolution of each image in each technique.In figure 9,we plotted the graph of execution-time of every technique, with x-axis as resolution and y-axis as execution time in milli-seconds (ms).

Here in figure 10,we plotted the graph of resolution vs execution-time of all techniques.We can cleary say that CUDA takes lesser execution-time than others.Sequential programming is taking lesser time than MPI,OpenMP upto some point, then after that OpenMP, MPI seems to take lesser time than

Fig. 7: Input image

Fig. 8: Output image

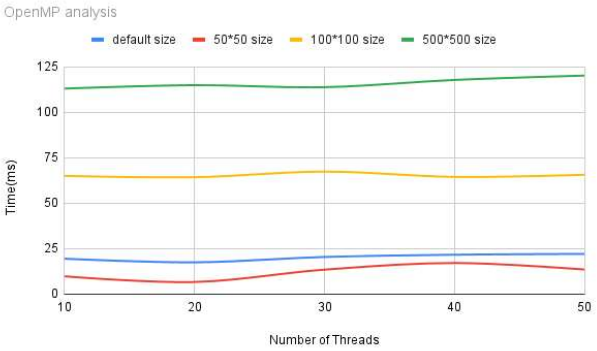


Fig 9.a Execution time in OpenMP implementation

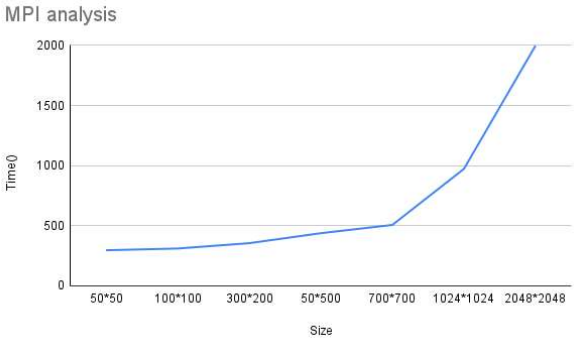


Fig 9.b Execution time in MPI-message passing interface implementation

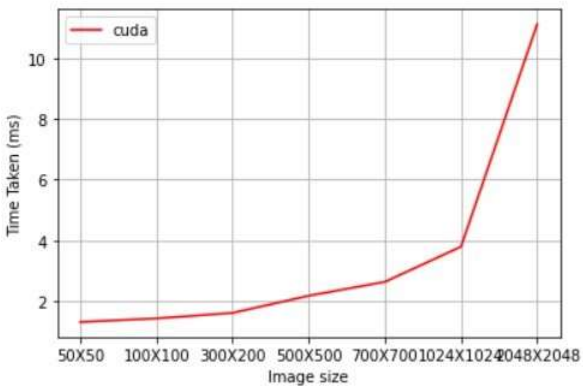


Fig 9.c Execution time in CUDA implementation

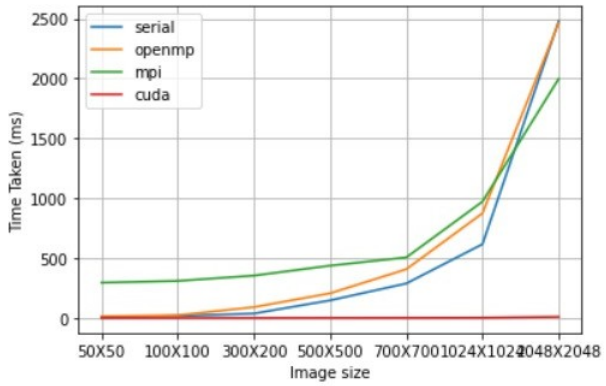


Fig. 10: Comparative analysis of all techniques

Sequential, we can say this as the slope is decreasing from image of resolution 1024x1024.Plot for CUDA in figure 10, seems like a horizontal line, but when plotted individually, you can see the execution-time in CUDA. The device configuration in which we executed the programs of sequential, OpenMP, MPI techniques , has Intel i7 7th gen processor, has 4 cores. We executed CUDA version of convolution in Google Colab-oratory, where cplaboratory makes Nvidia GPU available, 13 processors, 1024 maximum threads per block.

## VI. Conclusion

We implemented convolution in 3 different parallel programming techniques, also compared them by their execution-time.CUDA takes really lesser time than other parallel programming techniques as this follows Single-Instruction-Multiple-Data(SIMD), and in convolution we can pass the multiple data and do similar calculations at higher rate tahn other techniques, sp the convolution suites well with CUDA and has the lower execution time.When compared to CUDA, both techniques OpenMp and MPI takes higher execution time, since the number of execution units in OpenMP and MPI are lesser than CUDA since uses GPU rather than CPU.And OpenMP , MPI's slope is lesser than Sequential when we introduce images of more resolution, we can explain this trend by saying sometimes parallel execution may take same execution-time as sequential but as the calculations/parts need to be executed increases parallel programming works better than Sequential programming after certain resolution of images.

In CUDA and OpenMP we implemented parallel execution for the for loop, i.e., we parallelised the execution of nested loops , so calculation for each cell in the matrix can be executed parallely, whereas in MPI, we parallelised the convolution of matrices in channel level.So the improvement/future work can be like, we should execute parallely in both channel level of image (as in MPI) and cell/loop level in a particular channel (as in CUDA and OpenMP) then convolution becomes more optimized.

## VII. References

The Parallel Processing Based on CUDA for Convolution Filter FDK Reconstruction of CT *https://ieeexplore.ieee.org/abstract/document/5715077*.

Parallel Implementations of Im- age Processing Algorithms on Multi-Core *https://ieeexplore.ieee.org/document/5715373*.

Image convolution processing: A GPU versus FPGA comparison *https://ieeexplore.ieee.org/document/6211783*.

Parallel Programming with MPI *https://www.elsevier.com/books/parallel-programming-with-mpi/pacheco/978-0-08-051354-6*.