Parallelizing a convolution operation

Abstract:

First we understand what convolution operation is and what uses does it have. We then understand why is parallelizing the convolution operation important and how it is done so. Next we study the parallelized convolution algorithm in detail which is optimized for an NVIDIA GPU and performed on CUDA. This algorithm is compared to the sequential one mainly in terms of time complexity using a test case. Further applications of this are discussed.

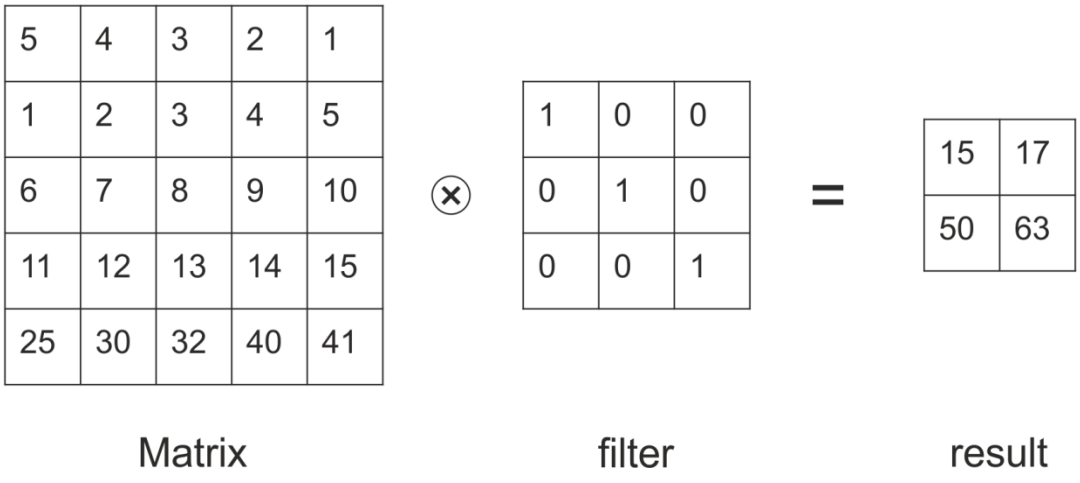
1. Introduction:

What is convolution?

We use convolution to extract common features from an image, using a filter.

A filter is a smaller matrix, which we use to perform this convolution operation on the image matrix to extract these features from it. The same matrix is used on multiple images to extract common features from it like finding edges or blurring the image.

An example of convolution with stride 2:



Formulaic example of convolution with stride 1:

```
|1   2   3|
|4   5   6|  *  |a   b|  =  |1*a+2*b+4*c+5*d  2*a+3*b+5*c+6*d|
|7   8   9|     |c   d|     |4*a+5*b+7*c+8*d  5*a+6*b+8*c+9*d|
```

Why do we need to parallelize convolution?

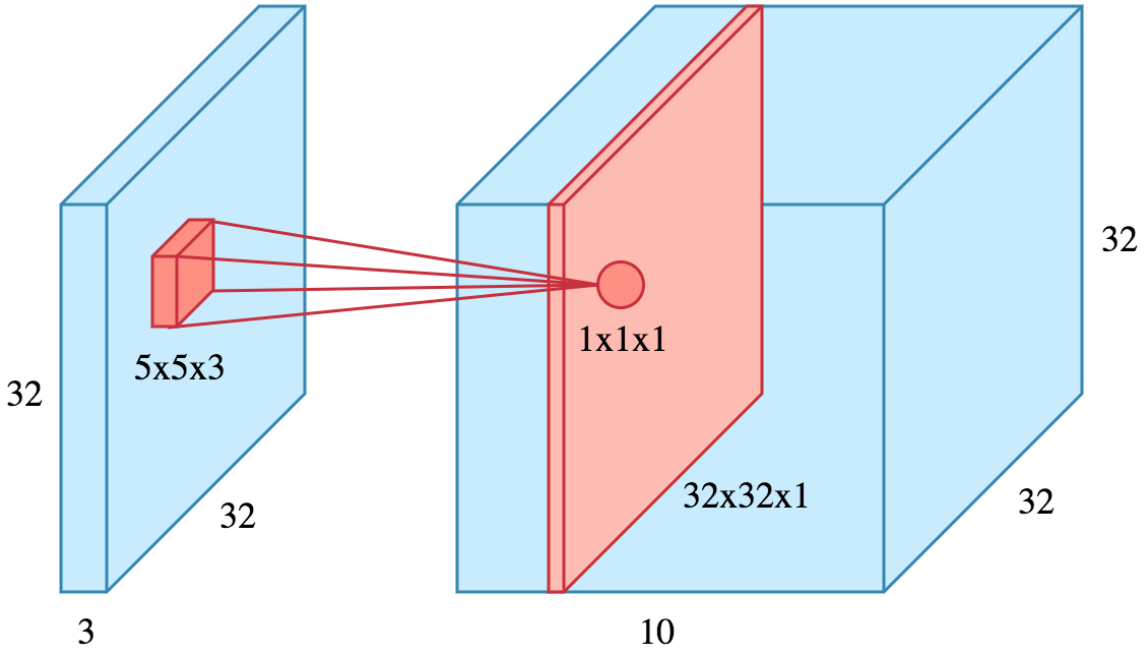
Why do we need to parallelize anything? To take less time.

Consider an application of Image Processing using Convolutional Neural Networks, for which we need to perform the same convolution operation hundreds of times.

In the image given below, It shows the visualization of a convolution layer as shown in this line of code:

```
Input=(32,32,3)
X=Convolution2D(10,filter=(5,5),padding='same') //filter size is (5,5,3) but size is
                                                automatically the shape of the depth of
                                                the input to the layer.
```

The output is a convolution operation of the image 10 times. If this function needs more time, the entire learning takes much longer.



If we parallelize it we can reduce the complexity from n^3 to n

NVIDIA GPUs are ubiquitous and best in the market for Machine Learning.
CUDA is a parallel computing platform and programming model invented by NVIDIA. It enables dramatic increases in computing performance by harnessing the power of the graphics processing unit (GPU).

2. Convolution algorithm

In this CUDA implementation, we have used a 1D array but we are using it as a 2D array stored in row major form.

Serial Pseudo Code:

```
For (i from 0 to size_result)

        For(j from 0 to size_result)

                For(k from 0 to size_filter)
                        For (l=0 to size_filter

                result[i*size_result+j] += filter[k*size_filter+l]*a[(2*i+k)*size_a+2*j+l];
                //here 2 is the stride which is constant for this op.

                        End for
                End for
        End for
End for
```

Parallel Pseudo Code:

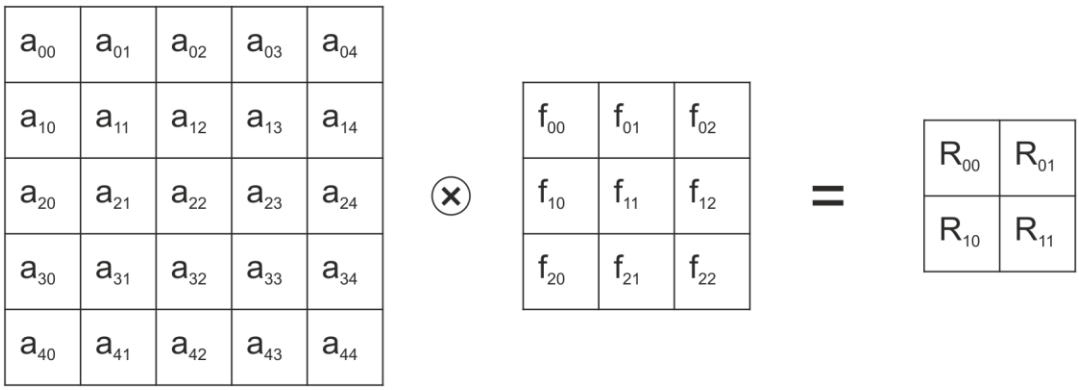
```
//(size_result,size_result) blocks made while calling the function.
//(size_filter,size_filter) threads made for each block.

begin
        i=blockIdx.x
        j=blockIdx.y
        k=threadIdx.x
        l=threadIdx.y

        if(i<size_result||j<size_result||k<size_filter||l<size_filter)

           atomicAdd(&result[i*size_result+j],filter[k*size_filter+l]*a[(2*i+k)*size_a+2*j+l]);
        end if
end
```

We are parallelizing by performing the computation by making a grid of the size of result matrix and in each block of the grid, we are making size_filter*size_filter threads.



$$size_{result} = \frac{(size_{img} - size_{filter} + 2 * padding)}{stride} + 1$$

$$R_{ij} = \sum_{k,l=0}^{size_{filter}} filter_{kl} * A_{(stride*i+k)(stride*j+l)} \quad \dots \; i,j = 0 \; to \; size_{result}$$

A *stride* is the number of elements that you skip while you move the filter across the image matrix.

3. Test Results

Size of Image Array: 10001 x 10001

Size of Filter: 3 x 3

Size of Matrix after Convolution with stride = (2) will be: 5000

Time for Convolution using serial: 3.104000 seconds

Time for Convolution with 25000000 x 9 threads: 0.131000 seconds

These results are on an NVIDIA GTX 1050, Intel i7 7700 processor.
Using CUDA 9.0 for parallelization and C as language.

4. Possible applications.

This parallelizing can be used in Convolutional Neural Networks (CNNs) which can drastically improve efficiency by reducing time taken to perform operation.
It can be used in medical field to help doctors interpret scans like MRIs, CT Scans, X-ray scans. This can be used in MRIs to detect edges and features like tumors.
It can be used in X-rays for hairline fracture detection.
In CNNs, convolution operation is performed at each node for a large input in each hidden layer, this task is gargantuan and with the help of this algorithm you can make your CNNs more efficient and less time consuming with a very popular GPU.