

# Dynamic Reuse of Memory in 2D Convolution Applied to Image Processing

Martin Casabella, Sergio Sulca, Ivan Vignolles, Ariel L. Pola

*Escuela de Ingeniería en Computación*

*Facultad de Ciencias Exactas Físicas y Naturales*

*Universidad Nacional de Córdoba*

Email: martin.casabella@gmail.com, ser.0090@gmail.com, ivignolles@alumnos.unc.edu.ar, arielpola@gmail.com

**Abstract**—En este artículo se presenta una arquitectura de hardware para realizar una convolución 2D en una FPGA cuando no se puede instanciar suficiente memoria RAM para poder alojar la imagen completa. Se priorizó la velocidad de procesamiento, el uso eficiente de los recursos y un diseño escalable donde se pudieran agregar tantas operaciones de convolución en paralelo como se desee sin deber hacer grandes modificaciones en el diseño.

## 1. Introduction

The origin of digital image processing, due to the high level of processing they require, is directly related to the development and evolution of computers. Most filters for images that focus, blur, enhance edges and detect edges, among others, use convolution as a mathematical operation. Image processing is a very extensive research area with a large number of applications in multiple fields such as medicine, engineering, navigation, aeronautics, among others. Applications recently, use convolutional neural networks (CNN) [1] where a fast and efficient convolution implementation is important.

The implementation of high speed image processing systems in FPGA has been a very active field. This is mainly due to the ability to take advantage of bit level parallelism, pixel level, neighborhood level and task level to accelerate the calculation. In addition, FPGAs are reconfigurable, allowing the flexibility that is often desired in neural networks. This coupling of processing, parallelism and flexibility at very high speed is what makes FPGAs a platform of choice for the development of these areas [2].

Convolutional FPGA architectures can store the entire image [3] and optimize the pixel processing speed [4] and reduce complexity of implementation [5]. Another approach is the filtering by blocks of image data, optimizing the parallelization of convolvers [6] and memory management [7]. The difficulty presented by these schemes is the non-modularization of processing, making it difficult to increase the work rate by increasing parallelism.

In this document, we propose a high-performance, parallelizable hardware architecture and dynamic memory reuse with a linear increase in BRAM resources.

## 2. Convolution Operation

Due to the data and convolution operation nature, a parallel approach it is much more efficient than a sequential one in terms of speed. FPGAs were used for system implementation, owing to the fact that these devices not only solve parallelism requirement but also allow different hardware prototypes testing.

2D discrete convolution is defined by the following equation:

$$S(x, y) = \sum_i \sum_j I(x - i, y - j)K(i, j) \quad (1)$$

Following nowadays deep learning community tendency of no filter flipping, since filter must be learnt anyway [8], cross correlation function is obtained:

$$S(x, y) = \sum_i \sum_j I(x + i, y + j)K(i, j) \quad (2)$$

which will be implemented, so in the following sections every time convolution operation is mentioned indicates a reference to eq. 2

For simplicity, grey scale images were used, since only one channel is needed.

## 3. Previous Analysis

The analysis purpose is to work with a finite minimum representation for image pixels and kernel values.

Pixels values  $I_{ij}$  range is  $[0, 255]$ . Dynamic range expansion [9] was used in order to rearrange values to  $[0, 1]$ . Kernel values  $K_{ij}$  can be negative or positive. Thus maximum norm [10] was also used to displace this range to  $[-1, 1]$ , so that kernel effects are maintained.

Signed fixed point representation [11]  $S(8,7)$  for previous range mentioned above.

Convolution between image and a  $3 \times 3$  kernel results in  $S(20,14)$ . Hence, post processing is made and the outcoming result value is shifted to positive range and truncated. Then, doing post-processing.

Signal to Noise Ratio (SNR) measure between  $S(20,14)$  output and error produced by reducing bits number  $e_r = f(x)_{20b} - f(x)_{pos}$  [12] is analyzed.  $S(13,8)$  representation

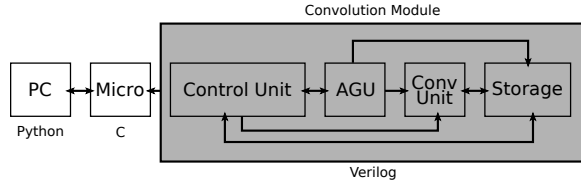


Fig. 1. System's workflow and languages used in every instance

leads to 30 [dB] SNR that shows filter effects with minimum data loss. This SNR value is acceptable because the objective is achieved despite not getting a perfect image.

## 4. Architecture Design

### 4.1. Workflow

As shown in Fig. 1, the workflow consists of three instances: first, in the PC a python script does the preprocessing described in section 3, separates the image in batches and then sends the kernel's coefficients and the first batch through an UART connection to the FPGA. In the FPGA a microprocessor receives the information and communicates it to the convolution module using a 32 bits GPIO port. The batch is then convolved with the kernel inside the module. When the operation is finished a notification is sent to the microprocessor, which gives the order to retrieve the processed batch from the module and sends it to the PC and then waits for the next batch to arrive.

### 4.2. Convolution Module

A simplified architecture of the module is shown in Fig. 2, the **control unit** is the one in charge of handling the communication with the processor, in the **convolution unit** is where the sum of the products of the kernel's coefficients with the pixels happen, the **address generation unit (AGU)** manages the memory addresses, the **memory management unit (MMU)** manages how the values in memory must be read and written, and finally the **storage**, which is implemented with a set of columns of the FPGA's block RAM.

**4.2.1. States.** It is possible to see that during the work cycle, the convolution module goes through several states, as shown in Fig.3.

At the beginning the module must be in a **reset** state, waiting to be configured, to do so it must receive a reset signal.

State transition are controlled by coded instructions embedded in the 32 bits input frame.

The module will stay in the **reset** state until it receives the instruction to go to the **KernelLoad** state, in which state the kernel coefficients are loaded into the module. Then it goes to the **SizeLoad** state where the image's height is loaded, the need for this is explained in section 4.2.2. Once the module has gone through this setup states it will not come back to them until the entire image is processed.

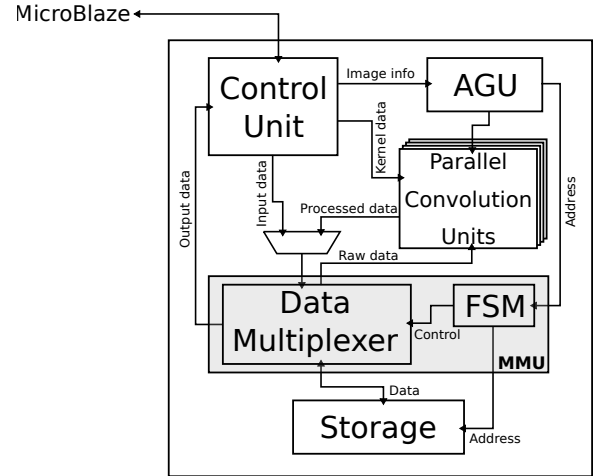


Fig. 2. Simplified block diagram of the convolution module.

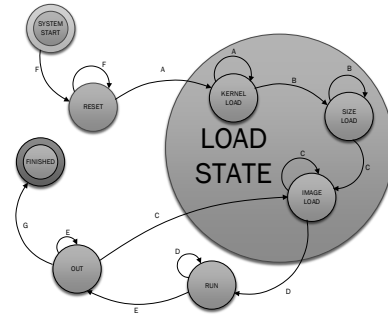


Fig. 3. Module's state diagram. A - Kernel load, B - Size load, C - Image load, D - Processing, E - Output, F - Reset

Now the processing loop start, first the module goes to the **ImageLoad** state and saves in the memory the batch, then is goes to the **Run** state where the processing occurs, while in processing the module can not be interrupted, i.e. it will ignore all instructions till the batch is processed, as soon as the processing is ended a notification will be emitted through Control Unit's *EoP* pin and the module will wait the instruction to got to the **Out** state where the already processed batch is send back to the microprocessor.

All this states logic is managed by the control unit, which depending on the state will communicate the corresponding control signal to the other components of the module.

**4.2.2. Data storage.** The kernel coefficients are stored in registers inside each convolution unit. In order to store a batch, the proposed approach is to arrange the block RAM memory in columns, where each of them will store a image's pixel column. Therefore a batch size is given by the height of the image and the number of memory columns, also the maximum height of the image will be restricted by the number of entries that each memory columns has.

During a batch processing, every pixel is read only once, so in order to do a more efficient use of the memory, the

pixels that were already used are overwritten with processed pixels. This approach reduces drastically the amount of memory needed because it reuses the same memory to store the input batch and the processed batch. In section 4.2.3 additional aspects of the design that reduces even more the memory use are explained.

**4.2.3. Data processing.** Given a  $k \times k$  kernel, each convolution unit takes  $k$  adjacent memory columns as input. To produce the first processed pixel, a convolution unit first needs to load  $k$  pixels from each columns in such a way as to have  $k \times k$  pixels loaded into it, then proceeds to multiply the pixels with the kernel's coefficients and sum everything. Finally, it truncates the result to reduce the bit length and the result is saved into the first position of the memory.

Once a pixel is processed, it is saved into the memory while the convolution unit shifts its image register, discarding the oldest  $k$  pixels and loading new  $k$  pixels, i.e. one from each input column, like a FIFO structure. This procedure is equivalent to shift the kernel vertically on the image. All this steps are synchronized in such a way that one processed pixel is generated in every clock cycle.

The AGU manages the memory's read and write address, it takes into account the latency between the clock cycle where the first pixel in the memory is read and the clock cycle where the first processed pixel is written in memory. This latency is equal to  $k$  plus some more clock cycles needed to latch values during the convolution, and the difference between the read address and the write address must be equal to it.

Until this point the analysis was over a single convolution unit, now the approach to work with multiple convolution units working in parallel is described.

As mentioned above,  $k$  columns are needed as input to produce one processed column in a convolution unit, that means that  $2k$  columns are needed for two convolution units, however given the nature of the convolution operation, to produce a contiguous column it is necessary to shift the input columns by one. Hence, there is an overlap between the two inputs and this produces that even that  $k$  input columns are still needed per convolution unit, only  $k + 1$  different input columns are needed for all the units. For that reason, it was decided that multiple convolution units in parallel will produce contiguous processed columns, sharing the common input columns. With this approach, for  $N$  convolution units the number of memory columns needed gets reduced from  $N \cdot k$  to  $N + k - 1$ .

The same overlap explained above produces also that repeated data will be needed between one batch and the next one. In order to eliminate the need to send data that already is in the memory, some data is reused between iterations. Given  $N$  convolution units and a  $k \times k$  kernel, a  $N + k - 1$  batch width is needed, but the processed batch has a width of  $N$  columns, i.e. one for each convolution unit, therefore the last  $k - 1$  memory columns are not overwritten and maintain the input data. This  $k - 1$  columns will be reused as the first columns of the next batch, thus the batch width

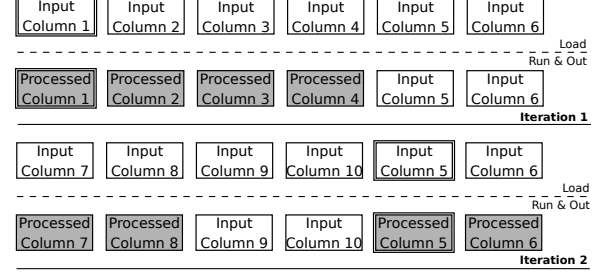


Fig. 4. Memory writing process using circular shift for  $N = 4$  and  $k = 3$ .

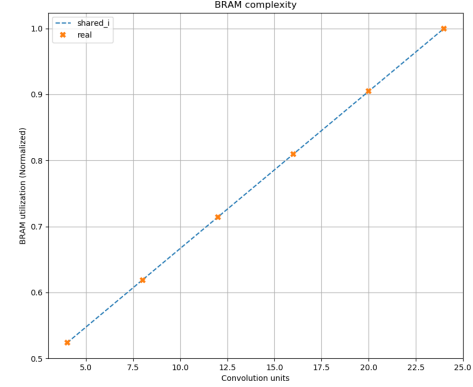


Fig. 7. Comparison between theoretical and synthesis.

is reduced to  $N$ , with the exception of the first batch that will still have a  $N + k - 1$  width.

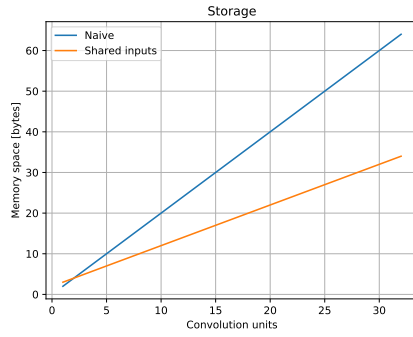
A consequence of reusing the memory columns written by the previous batch is that in every iteration the position of the memory columns associated with each convolution unit describes a circular shift by  $N$  places. From the above it follows that a periodicity in the relation between the memory columns and the convolution units' inputs must exist, where the period  $It$  is the number of iterations necessary to get to the original memory columns - convolution units inputs relation, i.e. when  $It(k-1)$  is a multiple of  $N + k - 1$ . That is, there must be an integer  $m$  such that

$$\frac{It}{m} = \frac{N}{k-1} + 1 \quad (3)$$

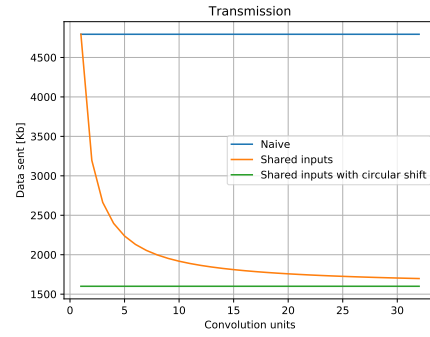
Figure 4 shows how the memories are written using this logic.

The logic is implemented in the MMU, which serves as an interface between the memories and the rest of the components, keeping them independent from the parallelism degree and the iteration number.

The MMU keeps track of the memory positions where it must save the input batch, the information which must feed to every convolution unit, the memory position where the processed data must be stored and the order in which the processed data must be returned, for each iteration. To do so, it has a finite state machine where each state corresponds to a set of positions named above. The number of states is given by eq. 3.

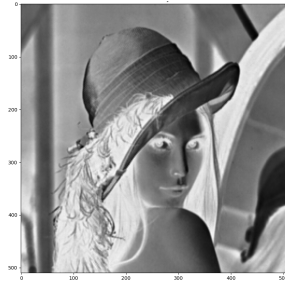


(a) Amount of memory required.

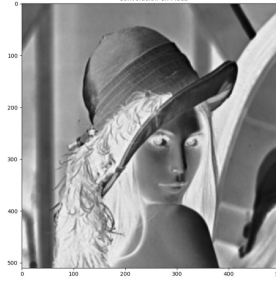


(b) Amount of data transmitted for an image of  $1600 \times 1024$ px and a  $3 \times 3$  kernel.

Fig. 5. Comparison between different approaches.



(a) filtration in Python.



(b) filtration in Board.

Fig. 6. Comparison between different processing.

To route the incoming and outgoing data, the MMU has a set of multiplexers whose select lines are managed by the finite state machine (Data multiplexer and FSM in Fig. 2).

A comparison between the naive (which would be assign  $k - 1$  independent memory columns to each convolution unit), the shared inputs and the shared inputs with circular shift approaches is shown in Fig. 5. Figure 6a shows the amount of memory required in function of the parallelism degree while Fig. 6b shows the amount of transferred bytes (considering that every byte transmitted is stored in only one memory register) for the processing of a  $1600 \times 1024$  image and a  $3 \times 3$  kernel.

## 5. Implementation and Results

Figure 7 shows the BRAM usage.

## 6. Conclusion

Being an engineer is pain

## Acknowledgments

The authors would like to thank... mucha gracia' ariel

## References

- [1] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov 1998.
- [2] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '15. New York, NY, USA: ACM, 2015, pp. 161–170. [Online]. Available: <http://doi.acm.org/10.1145/2684746.2689060>
- [3] A. Gupta and V. K. Prasanna, "Energy efficient image convolution on fpga," University of Southern California Los Angeles, USA, agrimgupta92@gmail.com, prasanna@usc.edu, Tech. Rep., 2011.
- [4] B. Cope, "Implementation of 2d convolution on fpga, gpu and cpu," UImperial College London, benjamin.cope@imperial.ac.uk, Tech. Rep., 2010.
- [5] A. D. Leila kabbai, Anissa Sghaier and M. Machhout, "Fpga implementation of filtered image using 2d gaussian filter," *International Journal of Advanced Computer Science and Applications*, vol. 7, no. 7, pp. 514–520, 2016.
- [6] H. Ström, "A parallel fpga implementation of image convolution," Master's thesis, Department of Electrical Engineering, Linköping University., 2016.
- [7] R. R. Diego Ramírez and F. Santa, "Parallel processing on fpga for image convolution using matlab," Master's thesis, Universidad Distrital Francisco José de Caldas, Bogotá, Colombia, 2015.
- [8] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, <http://www.deeplearningbook.org>.

- [9] R. E. W. Rafael C. González, *Digital Image Processing*. Prentice Hall, 2007.
- [10] W. Rudin, *Principles of Mathematical Analysis*, 3rd ed. McGraw-Hill, 1976.
- [11] G. C. S. Keith B. Cullen and N. J. Hurley, "Simulation tools for fixed point dsp algorithms and architectures," *International Journal of Signal Processing*, vol. 1, no. 4, pp. 199–203, 2008.
- [12] H. D. Ramón, "Análisis del error en algoritmos de transmisión de imágenes comprimidas con pérdida," Master's thesis, Facultad de Informática, U.N.L.P, hramon@lidi.info.unlp.edu.ar, 2002.