Arquitectura de Hardware para Convolución Bidimensional con Memoria Limitada Aplicada al Procesamiento de Imágenes

Martin Casabella, Sergio Sulca, Ivan Vignolles
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

***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. Introducción

La convolución bidimensional discreta es ampliamente usada en mltiples campos de la ingenieria, siendo uno de ellos la visión artificial que ha tomado fuerza en la ultima década con los resultados obtenidos con las redes neuronales convolucionales. El creciente tamaño en los modelos, como tambien en la informacion disponible para el entrenamiento hacen de la velocidad de procesamiento un factor de gran importancia.

Por la naturaleza de los datos y de la operación de la convolución, un enfoque paralelo resulta mucho más eficiente que uno secuencial en lo que a velocidad de procesamiento respecta. Se utilizaron FPGAs para la implementacion, pues resuelven la necesidad de paralelismo como también la de poder probar distintos prototipos de arquitecturas de hardware.

La convolución discreta en 2D esta definida por la siguiente ecuación:

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

Al no ser necesaria la propiedad de conmutatividad en este caso, es posible eliminar el proceso de espejar una de las matrices [1] , obteniendo la función de correlacion cruzada

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

la cual será en realidad implementada, por lo que en lo que resta del artículo, cada vez que se nombre a la operación de convolución, se estará haciendo referencia a la ecuación 2.

Uno de los mayores desafíos fue trabajar con memoria RAM lo suficientemente escaza que no fuera capaz de alojar todos los pixels de la imagen que se desea procesar. Por lo que la imágen debe ser fraccionada y procesada en lotes, con todo el procesamiento adicional que esto implica.

Por una cuestión de simplicidad, todo el trabajo se desarrolló utilizando imagenes en escala de grises, al tener estas un único canal.

2. Análisis Previo

The analysis purpose is work whit a finite minimum representation for image pixels and kernel values. To that is necessary make a study about bits number for a god image representation.

Pixels values I_{ij} have a range from 0 to 255. We use normalization dynamic range expansion [2] in ordet to get a new range from 0 to 1. Kernel values K_{ij} can to be negative or positive. Thus We use Maximum norm [3] to get maximum absolute value then it divide to every K_{ij} to achieve a new range from -1 to 1. So kernel effect is maintain.

We use Signed Fixed Point representation [4] $S(8,7)$ in order to use previos ranges.

Convolution of image whit a kernel $\mathbb{K}^{3\times3}$ results in 20-bits output $S(20,14)$. Hence we make post-processing. It come result value to positive range and truncate value. We compare both output post-processing and 20-bits so that find a minimum bits amount and do not lose important data.

We use SNR measure between ouput 20-bits $S(20,14)$ and error that results from reduce bits number $e_r = f(x)_{20b} - f(x)_{pos}$ [5]. Representation $S(13,8)$ has a approximate SNR $30[dB]$ that show filter effect whit minimum data loss. It is acceptable becouse the objet in no show a perfect image.

The values can come to a 8-bits representatio using dynamic range expansion [2] but it se need to get maximum and minimum number pixel value after convolution. So image should be in memory totally.

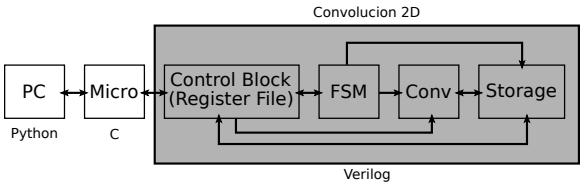

Fig. 1. Flujo de trabajo del sistema y lenguajes en los que se trabajó en cada instancia.

3. Arquitectura del sistema

3.1. Flujo de trabajo

El flujo de trabajo consta de tres instancias como se observa en la figura 1. En la primer instancia, se desarrolló un cliente en python donde se ingresa la imagen a procesar junto con los coeficientes del filtro que se desea aplicar, el filtro o kernel es una matriz que pertence al conjunto $\mathbb{R}^{3\times3}$. Toda la arquitectura del sistema ha sido diseñada para trabajar con filtros de esa dimensión, aunque no debería ser dificil generalizarla a filtros de otras dimensiones. La información es entonces separada en lotes, el cliente envía un lote, espera a recibir el mismo procesado y luego envía el lote siguiente.

Un microprocesador instanciado en la FPGA es el encargado de recibir los lotes, hace un desempaquetado de los datos y además le añade a los mismos una cabecera necesaria para la comunicación con el módulo de convolución. Asimismo una vez que el módulo finalizó el procesamiento, el microprocesador toma esa información procesada, la empaqueta y envía nuevamente a la computadora.

El módulo de convolución procesa el lote, internamente es el Bloque de Control (Control Block) quien hace de interfaz entre el resto del módulo y el exterior. La porción de la imagen es almacenada, los bloques de convolución (Conv) realizan la suma de los productos de los coeficientes del filtro con una sección de la imagen igual al tamaño del filtro, una vez que toda la porción de la imagen fue procesada, se envía el resultado y se espera a recibir una nueva porción. Una maquina de estados finitos (FSM) lleva cuenta del ciclo de entrada→procesamiento→salida que ocurre en el módulo.

3.2. Módulo de convolución

En esta sección se describe en detalle la arquitectura del módulo de convolución.

3.2.1. Estados. Durante todo el ciclo de trabajo, el módulo atraviesa distintos estados como se aprecia en la figura 2. En primera instancia, está en un estado inicial de **reset** esperando a ser configurado. Permanecerá en ese estado hasta recibir la intruccion para pasar al estado de **kernel load** (carga de kernel) donde se carga al módulo el filtro que se desea aplicar. Una vez que se termina de cargar los coeficientes del filtro, se espera la instrucción para pasar a **size load** (carga de tamaño), en este estado se debe informar

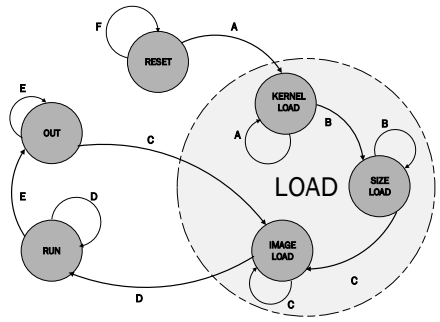

Fig. 2. Diagrama de estados del módulo de convolución. A - Carga de kernel, B - Carga de tamaño, C - Carga de imagen, D - Procesamiento, E - Out, F - Reset

al módulo la altura de la imagen. Habiendo ocurrido la transición al estado siguiente, no se retornará a los estados previos hasta finalizar el procesamiento completo de la imagen. Al salir del estado de **size load** se pasa al de **image load** donde se carga una sección de la imagen. Pasados estos tres estados se finaliza la etapa de carga. En la instacia siguiente el módulo se encuentra en el estado de **run**. En este estado se realiza el procesamiento del lote de forma ininterrumpida, es decir, toda instrucción será ignorada hasta finalizar el filtrado donde una señal de aviso es emitida. Se espera entonces la instrucción para pasar al estado **out**. Con el lote ya procesado y la intrucción recibida se da inicio al envío de los datos filtrados. La sucesión de los ultimos tres estados mensionados **image load** → **run** → **out** se repetirá por cada nuevo lote hasta obtener la imagen procesada en su totalidad.

3.2.2. Almacenamiento de la información. El kernel o filtro que se quiere aplicar a la imagen es almacenado en registros de los bloques de convolución. Con respecto a la imagen, se optó por ordenar la memoria en columnas, donde cada columna almacena una columna de pixels. Queda entonces como una limitación la altura máxima de la imagen dada por el número máximo de pixels que es posible almacenar en una columna de memoria.

El lote ya procesado es almacenado sobreescribiendo los datos que se utilizaron para producirlo, es decir, se utiliza la misma memoria tanto para los datos de entrada como los de salida.

3.2.3. Procesamiento de la información. Cada bloque de convolución toma los valores de tres (o k, para un filtro que pertence a $\mathbb{R}^{k\times k}$) columnas adyacentes. Comienza cargando los tres primeros valores de cada columna en sus registros, realiza la suma del producto con los coeficientes del filtro, y hace un desplazamiento de los registros con los valores de la imagen, de forma que se descarten los valores mas antiguos de cada columna, y se cargue un nuevo valor por columna, como en una estructura FIFO (first in, first out). Como muestra la figura 3, esto es equivalente a ubicar el filtro sobre tres columnas de la imagen y desplazarlo verticalemente.

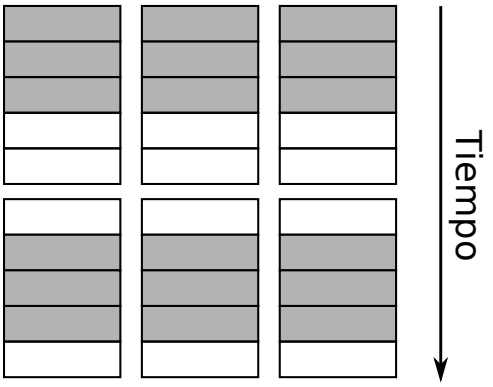

Fig. 3. En gris se observan los datos que han sido cargados a los registros del bloque de convolución, en el ciclo siguiente es posible ver como los valores de cada columna que ingresaron primero han sido decartados, y nuevos valores han sido cargados, produciendo un efecto de desplazamiento hacia abajo.

Luego de analizar distintas alternativas, se decidió que los bloques de convolución que se tienen trabajando en paralelo produzcan columnas procesadas contiguas, es decir, que el bloque de convolución 1 produzca la primer columna del lote, el bloque 2 la segunda y asi sucesivamente. Una de las razones por la que se tomo esta decisión es la naturaleza de la operación de la convolución: se necesitan k columnas de imagen para generar una columna procesada, para generar la columna contigua es necesario desplazarse un lugar en la imagen de entrada, por lo que existe unicamente una columna de diferencia en la entrada de dos bloques que generan dos columnas adyacentes. Esta superposición en la entrada implica una reduccion del número de columnas necesarias de $k \cdot N$ a $N+k-1$ para N bloques de convolución si estos generan columnas adyacentes.

4. Conclusion

The conclusion goes here.

Acknowledgments

The authors would like to thank...

References

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[2] R. E. W. Rafael C. González, *Digital Image Processing*. Prentice Hall, 2007.

[3] W. Rudin, *Principles of Mathematical Analysis*. McGraw-Hill, 1964.

[4] 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.

[5] 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.