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 multiples 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 en punto flotante y punto fijo

No se arranca con un simulador en python ya que las cuestiones del procesamiento de la imagen estn ms enfocada a la etapa de implementacin, sin embargo en la etapa de verificacin del sistema hubo que hacer un entendimiento del comportamiento de los distintos kernels(filtros).

Como primer paso tenemos que tomar una imagen y procesarla, pero hay que tener en cuenta que se cuenta con una resolucin finita, por ende necesitamos entender, cuantos bits de resolucin mnimos vamos a necesitar sin perder demasiada informacin.

2.1. Análisis de comportamiento

Trabajado con un análisis a nivel del comportamiento del kernel en python, se observa que el rango de la imagen deben ser modificadas, para trabajar en un rango de 8 bits, de ah surgen

2.1.1. Maximum norm. Toma como norma el mayor valor absoluto de una tupla con n elementos

$$\|\mathbf{x}\|_\infty := \max(|x_1|, ..., |x_n|)$$

2.1.2. División con respecto a la norma. Tomando el valor mximo de la tupla en valor absoluta y dividiendo cada uno de los elementos del kernel

$$\hat{k}_{ij} = \frac{k_{ij}}{\|\mathbf{k}\|}$$

la razn de todo esto es llevar los valores iniciales del kernel al rango [-1 ; 1). Para poder tener una representacin de 8 bits, para los mismos.

2.1.3. Expanción lineal dinámica de rango. Utilizados para el procesamiento de imgenes donde usualmente se traslada la imagen a un rango conveniente.

$$I_N = (I - Min)\frac{newMax - newMin}{Max - Min} + newMin$$

As como en el kernel, el objetivo aqui es llevar los valores de cada pixel de la imagen a [0 ; 1). Con el fin de tener una representacin en 8 bits.

Para reducir el nmero bits a manejar en la implementacin en hardware , ya sea en la etapa de clculo de convolucin como en la de transferencia de los datos por UART. se escoge como ancho de palabra 8 bits (para la etapa de envio de datos).

Al obtener un cierto cambio en el rango de la imagen que nos permiti a nosotros decir, que el producto de normalizar la seal, en principio, el rango de los datos nos entra en 8 bits y 7 bits, trabajando esto en python. Llegandose a lo siguiente.

Teniendo en cuenta que el kernel tiene componentes negativos Representacin utilizada en imagen S(8,7) Representacin utilizada en Kernel S(8,7)

En la operacin inicial de la convolucin se tendra como resultado en S(16,14) por cada producto , al tener un kernel de 3x3 se tiene un suma de 9 elementos. Teniendo una representacin S(20,14), lo que implica a la salida de 20 bits en el convolucionador.

Por cmo tenamos que trabajar con los mdulos UART la mxima unidad de informacin son 8 bits, por lo cual para enviar un dato , se requieren 3 envios.

Por ende se produce a realizar un post procesamiento que consiste en llevar el resultado de la convolucin a rango positivo y mapear los bits menos significativos a uno ms significativo. Para ello se realiza un truncado de los bits menos significativos.

2.1.4. Métrica. Para poder establecer una Mtrica y realizar una comparacin , para decidir la cantidad de bits de salida del procesamiento. todo esto con el fin de reducir, ya sea la parte fraccionaria, o la entera. Se empez a realizar una relacin entre la operacin a mxima resolucin y otra disminuyendo los bits.

Se genera una relacin seal ruido de la estimacin a 20 bits y el error generado de reducir la cantidad de bits. Con el objetivo poder determinar la cantidad de bits mnima de trabajo

[OJO ACA!!!! ????] ¡——

- Error:

$$e_r = f(x)_{20b} - f(x)_{pos}$$

- Energia:

$$E = \frac{1}{n}\sum_{i=0}^{n-1} x_i^2$$

- SNR:

$$SNR = \frac{E_{20b}}{E_{error}}$$

2.1.5. Representación en punto fijo. Haciendo la convolucin con un filtro unitario.

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Al realizar el anlisis reduciendo los bits de la parte fraccionaria, partiendo de 8 bits en totales se empez a observar que la imagen mejoraba , y que a partir de 13 bits totales 1 bit de signo, 5 bits parte entera y 7 en parte fraccionaria se tiene una SNR aprox. a 30 dB, la cual se considera suficiente.

[IMAGEN DE CONV FIX POINT Y POSTPROC]

Se considera suficiente debido a observacin de las imgenes, en donde se aprecia que si nos quedamos con 8 bits de salida (lo que intuye al principio por las resoluciones el filtro y de la imagen) se tiene una calidad muy baja. pero a medida que se aumenta la cantidad de bits, la imagen se va mejorando.

[IMAGEN LA ESTRUCTURA DE LOS BITS (SIGNO/ENTERO/FRACCIONARIO)]

[IMAGEN DE LENA CON DISTINTAS RESOLUCIONES]

Al nivel de 13 bits se tiene una mejor apreciacin de la imagen. Aunque si se realiza un anlisis ms finos se nota una degradacin respecto de la original. Pero como el objetivo no es no es representar una imagen en su totalidad sino, hacer deteccin de borde o detectar elementos, etc. para la cual la precisin de la imagen queda en un segundo plano.

Para avanzar un paso ms en la reduccin de bits se realiza el mismo anlisis reduciendo el bit ms significativo de la parte entera. lo cual permite quedarse con una resolucin de 12 bits.

[IMAGEN LA ESTRUCTURA DE LOS BITS NUEVA(SIGNO/ENTERO/FRACCIONARIO)]

[IMAGEN DE LENA CON DISTINTAS RESOLUCIONES NUEVA]

Se puede llevar el rango final a 8 bits, si se hace un cambio de rango igual al que se hizo para la imagen antes de procesarla . pero para ello se requiere conocer el maximo y el mnimo valor de pixel de la imagen luego de filtrar, esto requiere que toda la imagen se encuentre en la memoria. Debido a las limitacin esto no se poda llevar a cabo.

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

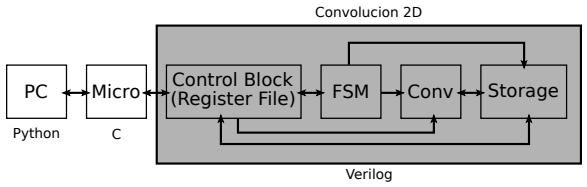

Fig. 1. Flujo de trabajo del sistema y lenguajes en los que se trabajó en cada instancia.

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 finita (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. Almacenamiento de la información. El kernel o filtro que se quiere aplicar a la imagen se configura durante la etapa de carga y es almacenado en registros de los bloques de convolución. Con respecto a la imagen, se optó por utilizar columnas de memoria, 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.

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 las 3 columnas 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 de cada columna, como en una estructura FIFO (first in, first out). Como muestra la figura 2, esto es equivalente a ubicar el filtro sobre tres columnas de la imagen y desplazarlo verticalemente.

Luego de analizar distintas alternativas, se decidió que cada bloque de convolución que se tiene

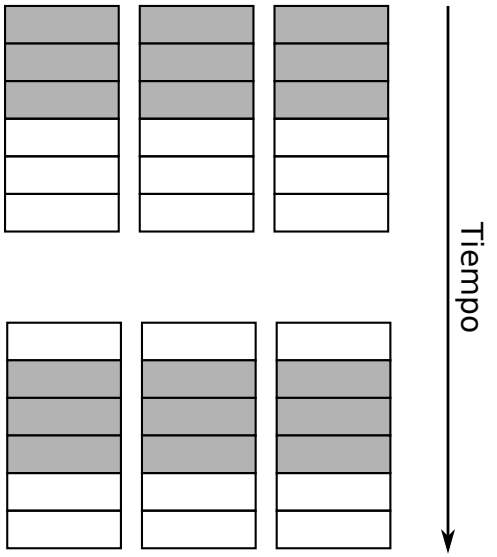

Fig. 2. En gris se observan los datos que han sido cargados a los registros del bloque de convolución, en el ciclo siguiente es posible ver como los valores de cada columna que ingresaron primero han sido decartados, y nuevos valores han sido cargados, produciendo un efecto de desplazamiento hacia abajo.

3.2.2. Estados. Durante todo el ciclo de trabajo, el módulo atraviesa distintos estados, en primera instancia, está en un estado inicial de **reset** esperando a ser configurado

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.