DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico

Diseño de Sistemas Digitales con FPGA

Primer Cuatrimestre de 2023

Integrante	LU	Correo electrónico
Mateo Cantagallo	143/21	mateocantagallo@gmail.com
Francisco José Letterio	883/21	fj.letterio@gmail.com
Juan Pablo Ramos Ruiz	139/18	ramosjruiz@gmail.com

Facultad de Ciencias Exactas y Naturales

Universidad de Buenos Aires

Ciudad Universitaria - (Pabellón I/Planta Baja)

Intendente Güiraldes 2160 - C1428EGA

Ciudad Autónoma de Buenos Aires - Rep. Argentina

Tel/Fax: (54 11) 4576-3359

http://www.fcen.uba.ar

Índice

1. Introducción

Este informe aborda el problema de procesar imágenes de microscopio aprovechando las ventajas de la paralelización y rapidez de cómputo inherentes al hardware de las FPGAs, en comparación con metodologías de software como el uso de GPUs. El objetivo es obtener una imagen que represente de manera más precisa el objeto observado a través de un microscopio, mediante la deconvolución de la imagen empírica basada en una función de dispersión de punto modelada con una distribución gaussiana.

2. Explicación del diseño

Cuando se manejan imágenes de gran tamaño, se divide la imagen en bloques cuadrados, en el mejor de los casos se contempla la imagen entera en un bloque.

Para aprovechar el potencial que ofrece el hardware de la FPGA, se propuso un sistema que permite paralelizar el procesamiento de la imagen basado en los píxeles. Cada hilo se encarga de calcular el brillo del escaneo del microscopio para un píxel. De esta manera, se establece un pipeline que procesa todas las *point sources* necesarias para lograr el brillo deseado en cada píxel de la imagen final.

El modelo implementado paraleliza la imagen a procesar por píxeles, de esta manera podemos hacer un *pipeline* en el que se va ingresando la luminosidad que aporta cada *point source* a un píxel, así todos los hilos del sistema acabarán la mismo tiempo maximizando el *throughput*.

El enfoque de pipeline en los hilos individuales nos permite aumentar el *throughput* y coordinar los diferentes módulos en función de la señal de reloj.

Aprovechando la libertad que provee el diseno de FPGA se propusieron varias maneras de paralelizacion, como hacerlo por interpoladores lineales en las matrices de PSF o con un evaluador por *point source*. Al final se decidio por la paralelización por píxeles con el fin de intentar conseguir el mayor *throughput* posible y también para dar flexibilidad a la hora de sintetizar el código. Mediante el uso de generics se pueden ajustar el tamaño del bloque de píxeles corriendo en paralelo según las capacidades del hardware utilizado, siendo un bloque de todos los píxeles a la vez la mayor paralelización posible.

3. Módulos del sistema

Cada hilo del acelerador se compone de los siguientes módulos, en orden de procesamiento del flujo de bits de entrada:

3.1. Offset

Este módulo toma una *point source* y calcula el desplazamiento (*offset*) con respecto al píxel del hilo mediante una operación de resta coordenada a coordenada. También cuenta con un contador que lleva el registro de las *point sources* recorridas. Cuando se alcanza el límite, el módulo pasa al siguiente bloque, si existe, y reinicia el contador.

Señales

- count: *signed* - cuenta de las *point sources* recorridas en el estado actual
- block_count: *signed* - cuenta de los bloques recorridos en el estado actual
- xs: *unsigned* - desplazamiento a aplicar a la coordenada x en el estado actual
- ys: *unsigned* - desplazamiento a aplicar a la coordenada y en el estado actual
- count_n: *signed* - cuenta de las *point sources* recorridas en el siguiente estado
- block_count_n: *signed* - cuenta de los bloques recorridos en el siguiente estado
- xs_n: *signed* - desplazamiento a aplicar a la coordenada x en el siguiente estado
- ys_n: *signed* - desplazamiento a aplicar a la coordenada y en el siguiente estado

Puertos

- psx_i : *entrada* std_logic_vector - coordenada x del *point source* a correr
- psy_i : *entrada* std_logic_vector - coordenada y del *point source* a correr
- offx_o : *salida* std_logic_vector - coordenada x del *offset* resultado
- offy_o : *salida* std_logic_vector - coordenada y del *offset* resultado

3.2. Evaluador

Cada hilo cuenta un evaluador, que calcula la distribución gaussiana por separado para cada coordenada. Recibe las coordenadas correspondiente de la *point source* con el desplazamiento aplicado y calcula su brillo. Para calcular eficiente y rápidamente la componente exponencial en la distribución gaussiana, se utiliza un algoritmo de cordic expandido[1].

Este modulo no se termino de implementar en este momento. La idea es implementar el evaluador propuesto por el paper referenciado.

3.3. Acumulador

Este módulo recibe el brillo calculado por cada *point source* para un píxel y los suma. Cuenta con un contador interno que indica cuándo se ha terminado de procesar el píxel y activa la señal de salida.

Señales

- count: *signed* - cuenta de las *point sources* sumadas en el estado actual
- acc: *signed* - suma de los brillos en el estado actual
- res: *signed* - señal de salida en el estado actual
- count_n: *signed* - cuenta de las *point sources* sumadas en el siguiente estado
- acc_n: *signed* - suma de los brillos en el siguiente estado
- res_n: *signed* - señal de salida en el siguiente estado

Puertos

- off_i : *entrada* std_logic_vector - luminosidad calculada para un *point source* a acumular
- out_o : *salida* std_logic_vector - luminosidad del píxel resultante

[1] A Novel Method for Computing Exponential Function Using CORDIC Algorithm - J. Sudha a, M. C Hanumantharaju b, V. Venkateswarulu a, Jayalaxmi Hc

3.4. Output de valores finales

La logica del acelerador que se encarga de coordinar la salida de valores de los hilos es el siguiente. Se espera a que los acumuladores terminen de sumar hasta obtener su valor final todos al mismo tiempo, una vez ocurre esto se comienza a asignar uno por uno los valores de los acumuladores a la salida final.

Señales

- matrix_o: *array of array of signed* - matriz de señales de salida de los acumuladores
- countx: *signed* - indice de la salida actual en x
- county: *signed* - indice de la salida actual en y
- countx_n: *signed* - siguiente estado del indice en x
- county_n: *signed* - siguiente estado de la salida en y
- ready: *signed* - cantidad de valores sin despachar de los acumuladores
- ready_n: *signed* - siguiente estado de la senal ready
- dispatching: *std_logic* - indica que el modulo es en proceso de ciclar el valor de cada acumulador en el output
- dispatching_n: *std_logic* - siguiente estado de dispatching, se mantiene solo si todavia ready no es igual a 0

4. Próximos pasos

Los próximos pasos para este proyecto incluyen la implementación de testbenches unitarios e integrales del sistema propuesto en VHDL. Además, se planea realizar análisis comparativos con otras metodologías de procesamiento de imágenes en términos de velocidad y calidad de los resultados obtenidos.

El otro principal aspecto que debe considerar, diseñar e implementar es la conexión con la maquina host y el programa de procesamiento de imágenes para que este modulo puede realizar su tarea como acelerador. Para esto se debe primero se debe establecer la interfaz de comunicación, por ahora se considera FPGA link, y en base a esto ajustar los mecanismos de input y output del modulo para permitir el intercambio de información.

Una vez esto se resuelva el solo faltaría realizar experimentos con diversas configuraciones de cantidad de Point sources, tamaños de bloque e imagen y paralelización con el fin de comparar el acelerador de FPGA contra el usado en el momento en GPUs.

5. Conclusiones

En este informe, se ha presentado un diseño de sistema para procesar imágenes de microscopio utilizando FPGAs. El enfoque de paralelización y el uso de pipelines han demostrado ser eficientes para acelerar el procesamiento de imágenes y obtener resultados de mayor calidad. El objetivo final de esta implementación fue aprovechar la libertad que ofrecen las FPGAs para encontrar un formato de paralelización que mejor se adapte al problema en cuestión que lo ofrecido en una GPU.