PROCESAMIENTO Y VISUALIZACIÓN EN ARQUITECTURAS EMBARCADAS DE CÓMPUTO DE ALTO RENDIMIENTO

MARIA ANDREA PIMIENTO OJEDA

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICOMECÁNICAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2017

PROCESAMIENTO Y VISUALIZACIÓN EN ARQUITECTURAS EMBARCADAS DE CÓMPUTO DE ALTO RENDIMIENTO

MARIA ANDREA PIMIENTO OJEDA

Trabajo de investigación para optar el título de Ingeniero de Sistemas

Director:
Ph.D CARLOS JAIME BARRIOS HERNÁNDEZ
Doctor en Informática
Codirector:
Ph.D ARTURO PLATA GÓMEZ
Doctor en Física

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICOMECÁNICAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2017

Dedicatoria

A mis padres Fanny y Ciro, por ser estar siempre conmigo, ser mi apoyo incondicional y mi modelo a seguir.

A Carmen, por siempre cuidarme y apoyarme como una madre.

A mis amigos y compañeros, por todos los buenos momentos compartidos.

A los profesores y a la UIS, por formarme profesional y personalmente.

Agradecimientos

A mis directores, los profesores Carlos Jaime Barrios y Arturo Plata Gómez, por la confianza que depositaron en mí, por brindarme su apoyo y conocimiento durante el desarrollo y satisfactoria culminación de este proyecto.

A los grupos de investigación Cómputo Avanzado y a Gran Escala (CAGE) y Grupo de Óptica y Tratamiento de Señales (GOTS), y al centro de Supercomputación y Cálculo Científico (SC3) de la Universidad Industrial de Santander por toda la colaboración, soporte y respaldo que me brindaron.

A todos mis compañeros del grupo de investigación CAGE, por el bonito ambiente de compañerismo que se vive allí.

TABLA DE CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

Resumen

TÍTULO: PROCESAMIENTO Y VISUALIZACIÓN EN ARQUITECTURAS EMBARCADAS DE CÓMPUTO DE ALTO RENDIMIENTO[*]

AUTOR: MARIA ANDREA PIMIENTO OJEDA[**]

PALABRAS CLAVE: Arquitectura embarcada o embebida, Nvidia Jetson, CUDA, Profundidad de Campo Extendida (EDF)

DESCRIPCIÓN:

El procesamiento de gran cantidad de imágenes y la visualización de sus resultados es indispensables para la investigación en microscopía, sin embargo, el aumento exponencial en la cantidad y tamaño de los datos hace estas tareas difíciles de manejar de manera local en computadores personales.

Tomando en consideración las alternativas para acelerar procesamiento y visualización, y teniendo en cuenta el tamaño, la eficiencia energética, los costos financieros y la portabilidad, se diseñó una Arquitectura de Cómputo de Alto Rendimiento Embarcada para el procesamiento y visualización de imágenes de microscopía de bajo costo, tanto financiero como energético. La plataforma embebida utilizada cuenta con un tamaño relativamente reducido, lo que le brinda portabilidad y cuenta con componentes programables, lo que le brinda flexibilidad en cuanto a los problemas que puede resolver.

También se implementó un piloto para resolver una situación que se presentaba en el Laboratorio de Óptica del grupo de investigación GOTS de la Universidad Industrial de Santander, allí los físicos requerían acelerar el tiempo de ejecución del algoritmo de procesamiento de imágenes llamado Profundidad de Campo Extendida (EDF), el cual renderiza una imagen 3D focalizada en todos los planos a partir de un stack de imágenes tomadas con diferentes planos de focalización, todo esto utilizando procesamiento masivamente paralelo sobre GPU con C CUDA en una plataforma Nvidia Jetson TK1 que se acopló al sistema de un Microscopio Axio Imager Z1m de Carl Zeiss.

[*] Trabajo de Grado en la Modalidad de Investigación

[**] Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingeniería de Sistemas e Informática. Director: Carlos Jaime Barrios. Codirector: Arturo Plata.

Abstract

TITLE: PROCESSING AND VISUALIZATION IN EMBEDDED ARCHITECTURES FOR HIGH PERFORMANCE COMPUTING[*]

AUTHOR: MARIA ANDREA PIMIENTO OJEDA[**]

KEYWORDS: Embedded architecture, Nvidia Jetson, CUDA, Extended Depth of Field (EDF)

DESCRIPTION:

Processing large numbers of images and visualizing their results is indispensable for microscopy research, however, the exponential increase in the amount and size of the data makes these tasks difficult to handle locally on personal computers or PCs.

Considering alternatives to accelerate processing and visualization, and keeping in mind the size, energy efficiency, financial costs and portability, a High Performance Computing Embedded Architecture was designed for the processing and visualization of microscopy images with low cost, both in terms of energy efficiency and financial costs. The embedded platform used has a relatively small size, which gives it portability and has programmable components, which gives it flexibility to the problems it can solve.

A pilot was also implemented to solve a situation that occurred in the Optics Laboratory of the research group GOTS at Universidad Industrial de Santander, there the physicists needed to speed up the execution time of the image processing algorithm called Extended Depth of Field (EDF), which renders a 3D image focused at all planes from an image stack captured with different focalization planes; all using massively parallel processing over GPUs with C CUDA on an Nvidia Jetson TK1 platform that was coupled to the Carl Zeiss Axio Imager Z1m Microscope system.

[*] Undergraduate final project, research modality

[**] Department of Physical-mechanical Engineering. School of Systems Engineering and Computer Science. Advisor: Carlos Jaime Barrios. Co-advisor: Arturo Plata.

INTRODUCCIÓN

El presente trabajo ofrece una alternativa para apoyar a la investigación científica que requiera acelerar el procesamiento y visualización de imágenes, ya que, como es bien sabido, el procesamiento de gran cantidad de datos y la visualización detallada de los resultados son requeridos en aplicaciones científicas, específicamente en microscopía, sin embargo, la cantidad de datos utilizados en estas aplicaciones está aumentando exponencialmente, haciendo estas tareas lentas y difíciles de manejar a nivel local en computadores personales.

Actualmente, hay una gran demanda por sistemas que ofrezcan capacidades de cómputo de alto desempeño porque brindan excelentes prestaciones en cuanto a rendimiento y eficiencia en tiempo; siendo uno de sus mayores exponentes los supercomputadores, sin embargo no todos los problemas requieren una infraestructura tan robusta y por ello es muy importante estudiar las características de cada caso que se va a implementar, porque estas suelen ser costosas de utilizar, tanto en términos de eficiencia energética como de costos financieros; además, la transferencia de los datos puede ser un proceso crítico en algunos lugares de difícil acceso.

Para los casos que son difíciles de manejar un computador personal pero no son tan grandes en términos computacionales como para trabajarlo en un supercomputador, o que por algún motivo no tiene la posibilidad de ser enviados a procesar a otro lugar, existe la alternativa de las plataformas embarcadas o embebidas, que hoy en día cuentan con excelentes prestaciones para acelerar procesos, pero con la ventaja de poder incorporarlas o incluirlas al sistema de trabajo; además de tener usualmente un tamaño reducido, un costo relativamente bajo, y especialmente, un eficiente consumo energético.

En este trabajo se presenta como objetivo principal el diseño de una Arquitectura Embarcada que cuenta con capacidades de Cómputo de Alto Rendimiento para realizar procesamiento y visualización de imágenes de un microscopio, y la implementación de un piloto para una situación específica. Con este piloto se busca resolver una problemática que se presentaba en el Laboratorio de Óptica del Grupo de Investigación en Óptica y Tratamiento de Señales (GOTS) de la UIS, en el cual requerían acelerar el proceso de obtener una imagen 3D focalizada en todos los planos, para el cual utilizan el algoritmo de procesamiento de imágenes llamado Extended Depth of Field (EDF) que trabaja con un stack de imágenes obtenido por un microscopio de contraste por interferencia diferencial.

Para ello, se encuentra plasmado en este documento el diseño de la arquitectura, sus componentes y especificaciones, junto con el piloto propuesto para el Laboratorio de Óptica, donde se presenta el análisis, tanto del sistema general del microscopio como del algoritmo, y se explica cuál fue la manera de proceder que se utilizó, en cuanto a las características de hardware, software, y programación.

Posteriormente, se realiza la validación de la arquitectura y del piloto implementado mediante una serie de pruebas, que permite evaluar su eficiencia para acelerar el algoritmo EDF y algoritmos que puedan ser implementados de forma masivamente paralelo sobre GPUs, además de conocer las limitaciones a las que está sujeto. Así, las conclusiones y trabajo futuro encontrados se presentan al final de este trabajo.

1. OBJETIVOS

1.1. OBJETIVO GENERAL

Proponer una Arquitectura de Cómputo de Alto Rendimiento Embarcada para el procesamiento y visualización de imágenes.

1.2. OBJETIVOS ESPECÍFICOS

- Identificar los componentes y relaciones del sistema microscopio - unidad de cómputo, para proponer una arquitectura de procesamiento y visualización embarcada de altas prestaciones.

- Implantar un piloto para acelerar el cálculo y la visualización en el sistema acoplado al microscopio a partir de los componentes definidos.

- Implementar algoritmos de tratamiento de imágenes siguiendo el Método de Profundidad de Campo Extendida EDF en el sistema embebido para evaluar la utilidad y rendimiento.

- Evaluar el piloto propuesto y caracterizarlo, para conocer las ventajas y limitaciones.

- Confrontar las imágenes obtenidas por el microscopio y las imágenes procesadas por la arquitectura propuesta para evaluar su resolución.

2. ESTADO DEL ARTE

Antes de entrar en materia es importante comprender el estado actual de la investigación en sistemas embarcados o embebidos para procesamiento y visualización, por eso a continuación, se hace referencia a algunos trabajos, investigaciones e información de proyectos que se han desarrollado y se relacionan con la temática de este proyecto.

2.1. TRATAMIENTO DE IMÁGENES USANDO EL MÉTODO DE PROFUNDIDAD DE CAMPO EXTENDIDA (EDF) EN ARQUITECTURAS PARALELAS

Proyecto de pregrado realizado por Mónica Liliana Hernández en la Universidad Industrial de Santander, en 2011 [1].

Este proyecto tenía como objetivo principal analizar y desarrollar un algoritmo a partir del Método de Profundidad de Campo Extendida (EDF) para el tratamiento de imágenes microscópicas usando procesamiento en paralelo, es decir, pretendía resolver casi el mismo problema del piloto del presente proyecto para Laboratorio de Óptica del grupo GOTS, pero utilizando herramientas diferentes.

La adquisición de las imágenes se realizó con un Microscopio de Contraste por Interferencia Diferencial (DIC), y las imágenes obtenidas eran enviadas para ser procesados externamente por un supercomputador, de manera que se aprovecharon los recursos del supercomputador Guane del Laboratorio de Supercomputación y Cálculo Científico (SC3) de la Universidad Industrial de Santander en dos tipos de arquitecturas: una máquina multicomputadora Cluster que permite el análisis de multiprocesamiento usando procesadores normales y un arreglo de GPUs que permite el procesamiento usando Procesadores Gráficos.

2.2. ZED STEREO CAMERA COMBINED WITH JETSON TX1 BRINGS ADVANCED 3D MAPPING TO DRONES

Proyecto desarrollado por la compañía Stereolabs, para procesamiento de video en tiempo real desde un drone utilizando la plataforma Nvidia Jetson TX1 [2].

La cámara estéreo ZED de Stereolabs se combina con unidades de procesamiento gráfico (GPU) integradas, la tarjeta NVIDIA Jetson TX1, en un drone para potenciar las aplicaciones de realidad virtual. De manera que este sistema es capaz de hacer Localización y Mapeo Simultáneo (SLAM) en 3D.

Un drone u otro vehículo, con capacidad 3D SLAM puede capturar fácilmente una comprensión detallada del mundo que la rodea, lo que apunta a nuevas oportunidades en la navegación autónoma. La mayor barrera a la navegación autónoma es la capacidad de ver y entender, en las simulaciones las computadoras han sido capaces de navegar por los obstáculos durante mucho tiempo, y con este sensor, una máquina puede ver el mundo “casi tan bien como un ser humano ", dicen sus autores.

2.3. FACE RECOGNITION BASED ON EMBEDDED SYSTEMS

Tesis de maestría, realizada por Hanna Björgvinsdottir, Robin Seibold, en Lund University en Junio de 2016 [3].

El objetivo fue implementar un algoritmo para el reconocimiento de rostros en un sistema embebido, con el propósito de obtener un rendimiento en tiempo real y una buena precisión en el reconocimiento facial. La autora investigó la posibilidad de desplegar una red neuronal para el reconocimiento facial en un sistema embebido, logrando una buena precisión y al mismo tiempo manteniendo una velocidad de inferencia aceptable. El objetivo no fue diseñar una nueva arquitectura de red neuronal, sino formar una existente sobre nuevos conjuntos de datos, y encontrar formas de compresión del modelo resultante. Otro objetivo también fue desarrollar un prototipo, que utilizará el modelo para clasificar las caras detectadas en las fotografías. Para este proyecto se utilizaron dos sistemas embebidos, el NVIDIA Tegra K1 y el NVIDIA Tegra X1.

2.4. DESIGN AND IMPLEMENTATION OF AN EMBEDDED H.264 COLOR VIDEO ENCODING PIPELINE FOR A MOBILE PROCESSING PLATFORM

Tesis de Maestría presentada por Andrew David Thompson, para obtener el grado de Master of Science in Electrical Engineering, en University of Dayton en Mayo de 2016 [4].

Esta tesis, propone el desarrollo de un pipeline para codificación de vídeo a color, incorporado H.264 en tiempo real, para una plataforma de procesamiento móvil. El H.264 es un formato de codificación digital de vídeo de alta definición. El pipeline de procesamiento móvil aborda la creciente necesidad de ver video de alta definición en dispositivos móviles tales como tabletas, o teléfonos móviles en un entorno de bajo ancho de banda. También el proyecto hace un análisis de la distorsión de velocidad y de la obtención de colores óptimos en la compresión de vídeo. El sistema embebido utilizado fue una cámara para la captura del video, y una tarjeta Nvidia Jetson TK1.

Figura 1. Arquitectura de pipeline para codificación de vídeo a color, incorporado H.264 en tiempo real, para una plataforma de procesamiento móvil [4]

3. MARCO TEÓRICO

3.1. SISTEMA EMBARCADO

Un sistema embarcado, también conocido como sistema embebido, es un sistema informático diseñado para realizar un grupo de funciones dedicadas y específicas, empleando para ello una combinación de recursos de hardware y de software. Poseen características diferenciales, entre otras: procesamiento concurrente, paralelo y distribuido, robustez, fiabilidad, bajo consumo energético y bajo costo, que los hace altamente recomendables y eficientes. Estos sistemas suelen estar limitados por la necesidad de ser soportados por sistemas operativos, sistemas de comunicación y/o bases de datos.

3.2. COMPUTACIÓN DE ALTO RENDIMIENTO

También conocida como Computación de Alto Desempeño o HPC (High Performance Computing), se refiere a la práctica de agregar poder computacional de manera que ofrece un rendimiento mucho mayor al que podría lograr una típica computadora de escritorio o estación de trabajo para resolver grandes problemas en ciencia, ingeniería o negocios. La práctica más común para alcanzar esto es el procesamiento paralelo, el cual se basa en la idea de dividir un problema en un conjunto de partes resolubles de forma concurrente, de manera que el tiempo total de resolución de la porción que se puede paralelizar del problema queda dividido por el número de procesadores utilizados, mientras que el tiempo de la porción serial se mantiene constante.

3.2.1. GPU: Unidad de procesamiento gráfico

Unidad de procesamiento gráfico o GPU (Graphics Processor Unit) es un coprocesador dedicado al procesamiento de gráficos u operaciones de coma flotante, para aligerar la carga de trabajo del procesador central. Una CPU está formada por varios núcleos optimizados para el procesamiento en serie, mientras que una GPU consta de millares de núcleos más pequeños y eficientes diseñados para manejar múltiples tareas simultáneamente.

3.2.2. CUDA

CUDA (Compute Unified Device Architecture) es una plataforma de computación paralela y un modelo de programación desarrollado por NVIDIA que permite al desarrollador programar algoritmos sobre la GPU mediante extensiones a lenguajes de programación como C, C++ y Fortran; así aprovecha la gran potencia de la GPU para proporcionar un incremento del rendimiento del sistema.

3.3. PROFUNDIDAD DE CAMPO

La profundidad de campo es un concepto empleado en óptica y fotografía para hacer referencia al grado de nitidez que posee la imagen de un objeto, organismo o paisaje en

general. La profundidad de campo corresponde a la distancia que está por delante y por detrás del plano de focalización, donde se pueden observar detalles finos del objeto real. Cuando la borrosidad se hace mínima en la profundidad de campo, se considera que la imagen es nítida en esa zona. La distancia nítida por detrás del plano de focalización es el doble que por delante de este. La profundidad de campo se ve afectada por la apertura del diafragma, la distancia focal y por la distancia entre el plano de foco respecto al plano a focalizar.

3.3.1. Método de Profundidad de Campo Extendida (EDF)

El método de profundidad de campo extendida (*Extended Depth of Field*) es un método artificial que permite a partir de un apilado de imágenes obtener una imagen focalizada en todos los planos. Esto es posible, obteniendo la intensidad de color en la posición focalizada y la posición en la cual logró focalizarse para cada una de las imágenes, obteniendo una imagen focalizada y una imagen de topografía respectivamente. Para lograr esto, el método EDF utiliza diferentes operadores como Varianzas, Sobel, Wavelets y Complex Wavelets.

A partir de la imagen de relieve o topográfica y la imagen focalizada, se reconstruye una imagen tridimensional.

3.4. MICROSCOPIO DE LUZ REFLEJADA

Los minerales absorbentes se caracterizan porque su estudio óptico no puede llevarse a cabo mediante el análisis de la luz transmitida por una lámina delgada y, por tanto, hay que estudiarlos mediante la luz que se refleja en superficies pulidas.

Por tanto, la microscopía de reflexión se basa en la incidencia normal, o al menos en la región de ángulos de incidencia muy pequeños, donde la óptica paraxial pueda ser aplicada. El microscopio de reflexión también es de polarización y su diseño está basado en los microscopios metalográficos, con la incorporación de luz polarizada incidente, y un polarizador ubicado en el tubo del microscopio (analizador) que permite el análisis de la luz reflejada por la muestra.

3.5. MICROSCOPIO DE CONTRASTE POR INTERFERENCIA DIFERENCIAL

El Microscopio de contraste por interferencia diferencial (DIC) utiliza la técnica DIC de contraste de fases que produce imágenes con un efecto óptico característico de relieve 3D, gracias a que está diseñado para observar relieves en superficies.

El contraste en las imágenes DIC es producido exclusivamente a través de un efecto óptico. Esta técnica, es usada para analizar especímenes vivos sin teñir, o examinar polímeros y otros materiales, según sea su configuración de luz, transmitida o reflejada.

3.6. HERRAMIENTAS

3.6.1. Plataformas Nvidia Jetson

Las NVIDIA Jetson [5] son plataformas de computación visual para el procesamiento paralelo acelerado por GPUs en sistemas móviles embebidos de NVIDIA. Estas cuentan con alto rendimiento computacional y bajo consumo energético para aplicaciones de deep learning y visión computarizada.

Tabla 1. Especificaciones de las plataformas Jetson TK1 y TX1 [6][7]

Plataforma Embebida		**Jetson TK1**	**Jetson TX1**
Procesador		Tegra K1	Tegra X1
CPU	Nombre	ARM Cortex-A15	ARM Cortex-A57
	Arquitectura	32 bits	64 bits
	Núcleos	4	4
GPU	Arquitectura	Kepler	Maxwell
	Núcleos	192	256
DRAM		2 GB DDR3L	4 GB LPDDR4
Memoria		16 GB eMMC	16 GB eMMC
Consumo energético		1-5 Watts	5-20 Watts
Precio		192 dólares	599 dólares

3.6.2. Microscopio Axio Imager Z1m

El microscopio Axio Imager Z1m [8] de Carl Zeiss es un microscopio de contraste por interferencia diferencial (DIC) de luz reflejada para investigación, controlado por computador y completamente motorizado.

El microscopio tiene una cámara AxioCam HRc, una unidad Z motorizada y puede utilizarse para adquirir imágenes en 3D.

Tabla 2. Características cámara AxioCam HRc [9]

Resolucón básica de CCD	1388 x 1040 = 1.4 megapixels
Tamaño del pixel	6.45 µm x 6.45 µm
Tamaño del sensor	8.9 mm x 6.7 mm
Sensibilidad espectral	Aprox. 400 - 700 nm con filtro de bloqueo de IR BG 40
Rango dinámico	Típico> 2200: 1 con <8 e lectura de ruido
Full Well	Típico 17.000 e

3.6.3. Software AxioVision

AxioVision [10] es un software desarrollado por Carl Zeiss para el control e interacción con sus microscopios. Este es un sistema modular de procesamiento y análisis de imágenes para microscopía moderna. La funcionalidad básica para la adquisición de imágenes y el control del microscopio, el procesamiento de imágenes y las anotaciones, el análisis de imágenes, la documentación y la configuración también se pueden ampliar integrando módulos adicionales en el sistema para tareas específicas.

3.6.4. Software ImageJ

ImageJ [11] es un programa de procesamiento de imágenes de código abierto diseñado para imágenes multidimensionales científicas, programado en Java y desarrollado en el National Institutes of Health. Se caracteriza por ser muy extensible, con gran cantidad de plugins y scripts para realizar una amplia variedad de tareas, además de una gran comunidad de usuarios.

4. METODOLOGÍA

El proyecto se realizó en 5 fases principales, de las cuales las fases 2, 3 y 4, enfocadas en el desarrollo, se trabajaron en 2 ciclos iterativos permitiendo la corrección de defectos y realización de mejoras.

4.1. FASE 1: INVESTIGACIÓN Y DOCUMENTACIÓN DE LA PROBLEMÁTICA A SOLUCIONAR

El desarrollo del proyecto se inició con la investigación en arquitecturas embarcadas y sus posibilidades en el campo a tratar; esto con el objetivo de definir las ventajas y dificultades o limitaciones que presentaba; también para descubrir los componentes y características necesarios para la implementación y así obtener una vía adecuada de dar solución la problemática.

4.2. FASE 2: DISEÑO DE LA ARQUITECTURA EMBARCADA

Una vez estudiadas y definidas las posibilidades, se inició el diseño de la Arquitectura Embarcada enfocada en procesamiento y visualización acelerados por medio de tecnologías de cómputo de alto rendimiento embarcadas o embebidas; para la cual se definió cada uno de los componentes y sus respectivas características.

4.3. FASE 3: IMPLEMENTACIÓN DEL PILOTO

Se estudió a detalle el caso problema a solucionar, que involucra el microscopio y la infraestructura del Laboratorio de Óptica del grupo de investigación GOTS, teniendo en cuenta la arquitectura previamente diseñada, se procedió a definir los componentes de hardware y software necesarios, y finalmente realizar la implementación del piloto propuesto, utilizando la tarjeta Jetson TK1 de Nvidia que se acopló al sistema del Microscopio Axio Imager Z1m para procesar las imágenes.

4.4. FASE 3: IMPLEMENTACIÓN EN PARALELO DE EDF EN CUDA

Se analizó la estructura del método de procesamiento de imágenes que se quería acelerar, Método de Profundidad de Campo Extendida (EDF), y la manera de paralizarlo sobre GPUs, para así hacer su implementación en C CUDA, la cual es ejecutada en la tarjeta Nvidia Jetson TK1.

4.5. FASE 4: VALIDACIÓN Y MEDICIÓN DE RESULTADOS.

En la última fase, se desarrolló el proceso de validación de los resultados, realizando pruebas para medir la eficiencia del piloto implementado en paralelo con CUDA respecto a una implementación con la que ya contaban en el laboratorio, permitiendo observar la ganancia obtenida y también las limitaciones que posee.

5. MÉTODO DE PROFUNDIDAD DE CAMPO EXTENDIDA (EDF)

Antes de entrar en materia, vamos a revisar cómo funciona el método que se desea acelerar, su procedimiento y resultados.

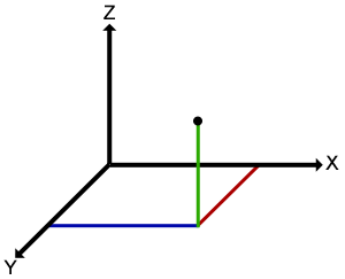
El método EDF emplea diferentes operadores que permiten tratar una imagen con base en las necesidades que se tienen y al tipo de información que se desea extraer de ella. Para el desarrollo de la implementación de este trabajo se escogió el operador de Varianza, el cual calcula las variaciones de intensidad de un píxel teniendo en cuenta los píxeles vecinos

PROCEDIMIENTO

1. Stack de imágenes

Es necesario obtener un stack de imágenes del objeto, ya que una imagen sólo puede estar focalizada en un área pequeña y no en todo el plano; así cada imagen estará focalizada en diferentes áreas. La forma como se capturan las imágenes del stack en un microscopio óptico, es manteniendo constante los planos x e y, haciendo un desplazamiento en el plano z del objeto en orden de micrones, de manera que en cada captura se obtiene un plano diferente focalizado. La cantidad de imágenes que contiene el stack depende del tamaño de objeto y del tamaño del paso al que se realiza el desplazamiento en el plano z.

Figura 2. Plano x, y, z



2. Descomposición en Matrices RGB

Los colores de un pixel son el resultado de combinar en diferentes porcentajes los colores rojo, verde y azul (RGB, por las siglas en inglés: Red, Green, Blue). Estos porcentajes tienen un equivalente numérico entre 0 y 255, de manera que el color de un pixel, puede expresarse como una terna de números (R, G, B).

Así, cada una de las imágenes se descompone pixel a pixel en 3 Matrices, una para cada color, del mismo tamaño en píxeles que la imagen.

3. Cálculo de las Matrices de Varianza

Se desea encontrar donde están los valores máximos de varianza, ya que es allí donde la imagen está mejor focalizada, y para ello los cálculos de variaciones se realizan sobre la Matriz G (Verde), porque para las condiciones de iluminación utilizadas en la toma de datos

y la reflectividad de las muestras, se observa que esta componente refleja de manera adecuada las variaciones de intensidad.

Para calcular las variaciones de intensidad se calcula la varianza en cada elemento de la matriz teniendo en cuenta sus vecinos más próximos, por lo que la Matriz de Varianza tiene el mismo tamaño que la Matriz G. Para ello se define una máscara de tamaño *pxp,* siendo p un número impar, la cual es una matriz donde su elemento central es el elemento al cual se le va a calcular la varianza.

Figura 3. Matriz G de 8x8 y máscara de 3x3 para el cálculo de la varianza

0,0	0,1	0,2	0,3	0,4	0,5	0,6	0,7
1,0	1,1	1,2	1,3	1,4	1,5	1,6	1,7
2,0	2,1	2,2	2,3	2,4	2,5	2,6	2,7
3,0	3,1	3,2	3,3	3,4	3,5	3,6	3,7
4,0	4,1	4,2	4,3	4,4	4,5	4,6	4,7
5,0	5,1	5,2	5,3	5,4	5,5	5,6	5,7
6,0	6,1	6,2	6,3	6,4	6,5	6,6	6,7
7,0	7,1	7,2	7,3	7,4	7,5	7,6	7,7

En la **Figura 3** se observa un matriz de tamaño 8x8, representando la Matriz G de una imagen con resolución 8x8 pixeles. Para realizar el cálculo de las varianzas se define una máscara de tamaño 3x3, la cual se desplaza por todas las posiciones ij de la matriz.
A continuación, se muestran las fórmulas:

$$Varianza = \frac{1}{T} * \sum_{i=1}^{T} (Xprom - Xi)^2$$

$$Xprom = \frac{1}{T} * \sum_{i=1}^{T} Xi$$

Dónde, T es el tamaño de la máscara, es decir pxp, Xi es el valor numérico en la posición i, Xprom es el promedio de los elementos contenidos en la máscara.

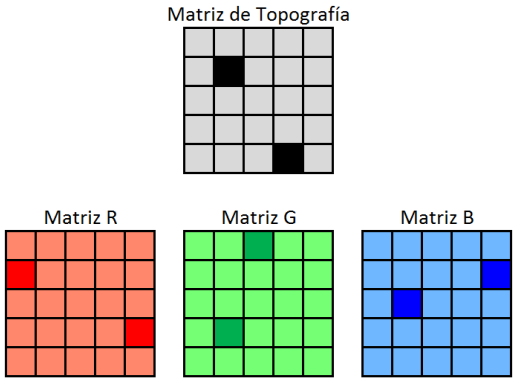
4. Cálculo de la Matriz de Topografía

Con el objetivo de encontrar las zonas focalizadas se realiza una comparación elemento a elemento de entre todas las Matrices de Varianza, donde una mayor variación para dicho elemento implica un área focalizada, por lo que obtenemos el máximo de varianza para cada elemento. Como resultado se obtiene una matriz de posiciones o de topografía, donde cada elemento ij de la matriz de topografía contiene la posición de la imagen en el stack que tiene el mayor valor de varianza en ese elemento ij, siendo la posición del stack correspondiente a la altura en el eje z.

5. Cálculo de las Matrices RGB finales

Tomando los valores de Matriz de Topografía, que equivalen a la posición en el stack, se obtiene para cada elemento su respectivo color RGB, y así se generan las matrices R, G y B finales, para poder reconstruir la Matriz de Textura o de Color de la imagen focalizada.

Figura 4. Resultado del método EDF: Matriz de Topografía y matrices RGB finales

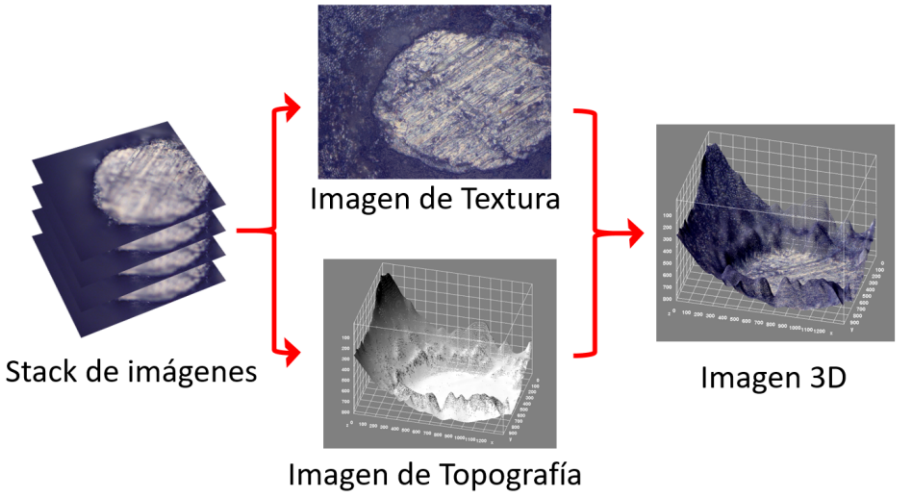


Así, se tiene como resultado una Matriz de Topografía y las 3 matrices de color R, G, y B, que componen la Imagen de textura.

6. Visualización

Finalmente se realiza la visualización, como se observa en la **Figura 5**, a partir del stack de imágenes y con los pasos anteriores se obtiene la imagen de Textura o imagen focalizada y la imagen de Topografía; con ellas se obtiene y visualiza la Imagen 3D final, focalizada en todos sus puntos.

Figura 5. Reconstrucción de imagen 3D obtenida con el método EDF, a partir de imágenes de Textura y Topografía



6. DESARROLLO DEL PROYECTO

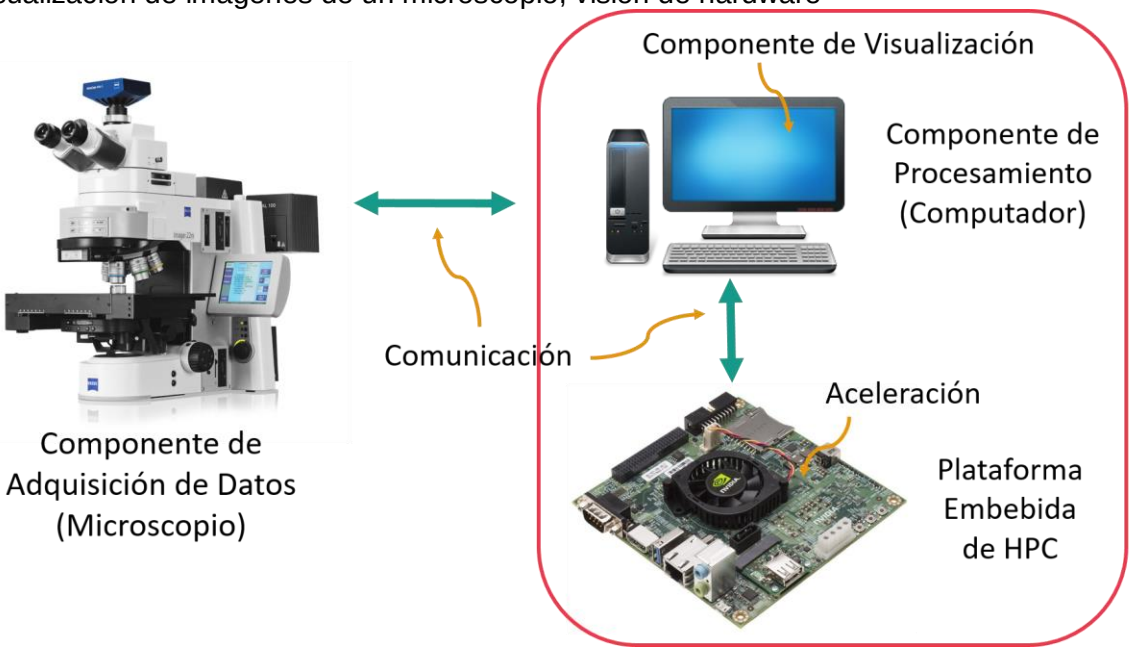
6.1. DISEÑO DE LA ARQUITECTURA

El desarrollo del proyecto inicia con el diseño de la arquitectura y la descripción de sus componentes y relaciones, lo cual será la base para los pasos a seguir.

La arquitectura embarcada, o embebida, de procesamiento y visualización de imágenes propuesta se diseñó con base en 5 componentes principales, los cuales describen las necesidades básicas para el correcto funcionamiento y son flexibles a ciertos escenarios.

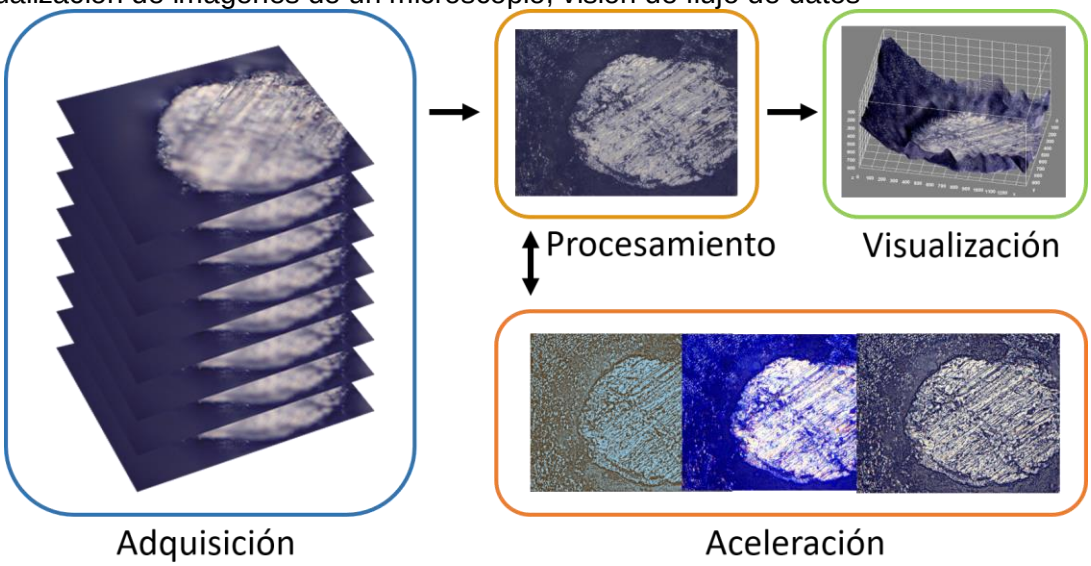
En la Figura 6 se presenta la arquitectura propuesta, desde el punto de vista de hardware, con sus componentes: Adquisición de datos, Plataforma embebida, Procesamiento, Visualización y Comunicación; y más adelante en el apartado de "Componentes" se detallan las características y la manera en que se relacionan.

Figura 6. Arquitectura embarcada de cómputo de alto rendimiento para procesamiento y visualización de imágenes de un microscopio, visión de hardware



En la **Figura 7** se presenta la arquitectura propuesta, desde el punto de vista del flujo de datos, el cual inicia con la adquisición de las imágenes por parte del microscopio y que se envían para ser procesadas al computador, en este punto es donde la plataforma apoya al sistema con la aceleración de este procesamiento y regresa los resultados para su visualización.

Figura 7. Arquitectura embarcada de cómputo de alto rendimiento para procesamiento y visualización de imágenes de un microscopio, visión de flujo de datos



COMPONENTES

1. Componente de adquisición de datos: Microscopio

El microscopio es el encargado de proporcionar al sistema las imágenes que serán procesadas y analizadas, las cuales pueden incluir o no pre-procesamiento. El microscopio debe ser digital y motorizado, para permitir la transmisión de las imágenes a los componentes de procesamiento.

2. Componente de procesamiento: Computador

Este componente por lo general ya se encuentra directamente acoplado al microscopio, porque la gran mayoría de microscopios digitales cuentan con un computador que entrega directamente el fabricante y trae todo lo necesario para su manejo, este incluye software y los DLLs propios para el control que suelen ser privativos. También puede apoyar a la plataforma embebida con algunas funciones.

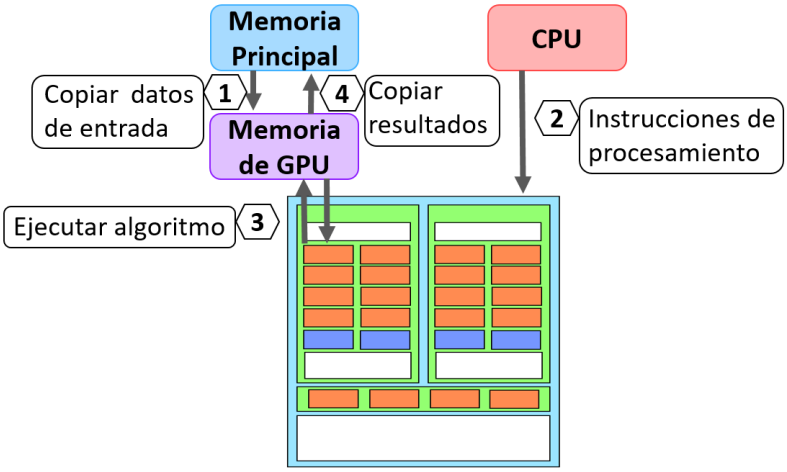
3. Plataforma Embebida de HPC

La plataforma embebida es la encargada de acelerar y debe ofrecer características de cómputo de alto rendimiento enfocado en el procesamiento sobre GPU, porque su objetivo es acelerar algoritmos de tratamiento de imágenes. Además, para este tipo de arquitecturas es esencial que cuente con bajo consumo energético y posea un tamaño pequeño, para que sea portable.

La plataforma se encuentra conectada con el computador principal que está encargado de recibir las imágenes del microscopio y enviarlas a la plataforma, donde realiza el procesamiento sobre la GPU, como se observa en la **Figura 8**, esta presenta un flujo de datos en el cual la memoria principal (CPU) envía los datos de entrada a la Memoria de la

GPU, ya que la GPU no tiene acceso a la memoria principal, y también envía las instrucciones de procesamiento para poder ejecutar el algoritmo; finalmente el resultado enviado a la memoria principal, porque la CPU tampoco tiene acceso a la memoria de la GPU. Estando en la memoria principal de la CPU los resultados son enviados al componente de visualización.

Figura 8. Flujo de procesamiento en la GPU



4. Componente de visualización

El componente de visualización está compuesto por un software que permita visualizar y analizar de una manera correcta y amigable los resultados obtenidos del procesamiento, y por el hardware para proyectar, que puede ser alguna pantalla o pantallas que ya se encuentren disponibles o que se agreguen al sistema general.

5. Comunicación

En este tipo de arquitectura mantener una comunicación estable entre todos los componentes es esencial y el tipo de comunicación utilizado depende de los canales de comunicación con que cuente el componente de procesamiento y la plataforma embebida, siendo la comunicación por Red la más tradicional y fiable.

Al final de esta etapa del desarrollo se tiene el diseño definitivo de la arquitectura de procesamiento y visualización embarcada de altas prestaciones, para el sistema de un microscopio.

6.2. IMPLEMENTACIÓN DEL PILOTO

Una vez diseñada y definida la arquitectura, se desea validarla mediante su utilización para un caso real, es decir, una situación que permita observar cómo funciona, y lo más importante, que resuelva un problema existente y apoye a los científicos en sus investigaciones.

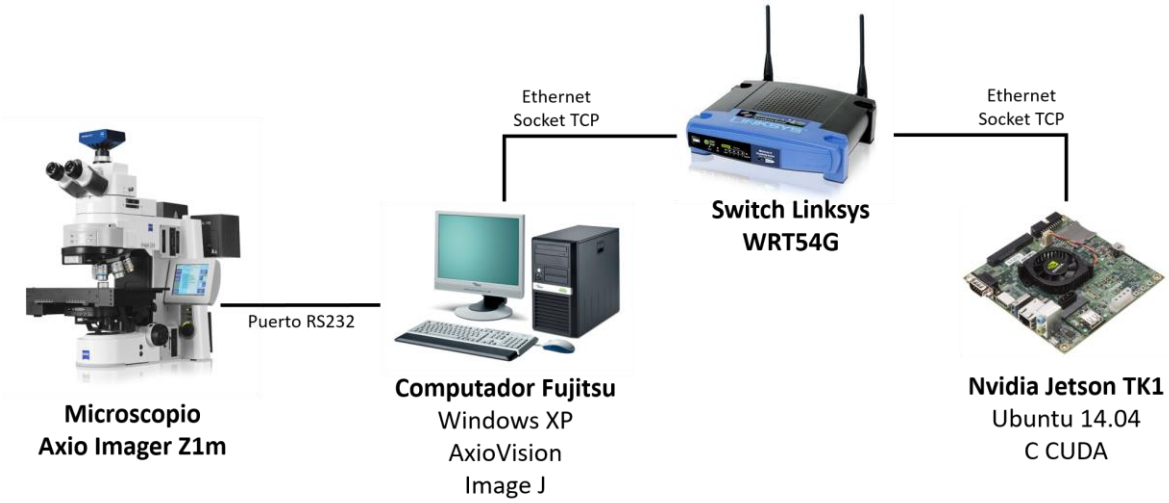
Así, el piloto propuesto se desarrolló para resolver la problemática existente en el Laboratorio de Óptica del grupo GOTS, donde requerían acelerar el Método de Profundidad de Campo Extendida EDF de las imágenes tomadas con un microscopio DIC.

Una vez comprendido el problema, se procede analizar las opciones de implementación, teniendo en cuenta los recursos e infraestructura disponibles. En el laboratorio ubicado en el Parque Tecnológico Guatiguará se encuentra el Microscopio Axio Imager Z1m que se encuentra acoplado con un computador de escritorio Fujitsu Siemens.

También se contaba con la plataforma embebida NVIDIA Jetson TK1 para acoplar con el sistema existente y realizar el procesamiento de las imágenes.

Teniendo en cuenta todas estas características se implementó el piloto de la arquitectura basado en los componentes descritos en la etapa de diseño, como se puede observar en la **Figura 9**, y sus descripciones se detallan en el apartado de “Componentes”.

Figura 9. Piloto propuesto para el Laboratorio de Óptica



COMPONENTES

1. Componente de adquisición de datos: Microscopio

El microscopio Axio Imager Z1m, de Carl Zeiss, es un microscopio de Contraste de Interferencia Diferencial (DIC) de luz reflejada y posee una cámara AxioCam HRc, éste es

el encargado de adquirir el stack de imágenes a diferentes planos de focalización y enviarlo hacia el computador de escritorio (3.6.2).

2. Componente de procesamiento: Computador

El computador de escritorio Fujitsu Siemens, como se había comentado que suele ocurrir con este tipo de microscopios, es el entregado por el fabricante, se encuentra acoplado al microscopio y posee el software AxioVision y todos los DLLs requeridos, de manera que soporta todas las funciones del microscopio y a través de este se realiza el proceso de toma de imágenes. Este computador tiene como sistema operativo Windows XP Service Pack 2. Luego de obtener el stack de imágenes, lo envía a la tarjeta Jetson TK1.

3. Plataforma Embebida de HPC

La plataforma embebida es la tarjeta Jetson TK1, que cuenta con una GPU NVIDIA Kepler GK20A con 192 CUDA cores, lo que la hace muy adecuada para el procesamiento masivamente paralelo de las imágenes sobre la GPU, y tiene un sistema operativo Linux4Tegra OS, que es básicamente un Ubuntu 14.04 con drivers preinstalados; además, cuenta con dimensiones de 12.7cm x 12.7cm y un consumo de energía bajo, entre 1 y 5 Watts.

Ella se encarga de acelerar mediante el procesamiento paralelo, y regresa el resultado al computador de escritorio para su visualización.

4. Componente de visualización

El componente hardware de visualización seleccionado fue la pantalla del computador de escritorio con la que ya se contaba, por tanto, para el componente software se buscaron herramientas para hacer uso de ella, siendo finalmente ImageJ la seleccionada, ya que es una herramienta amigable y cuenta con una amplia variedad de plugins, en especial el plugin “Interactive 3D Surface Plot” que permite visualizar la imagen 3D de manera interactiva.

5. Comunicación

La comunicación entre el computador de escritorio y la tarjeta Jetson TK1 se realizó por medio de socket TCP programado en lenguaje C++, utilizando la red local disponible en el laboratorio dada por un switch Linksys WRT54G, ya que ambos componentes de procesamiento cuentan con puertos de red. Para la tarjeta Jetson TK1 con Ubuntu 14.04 se utilizó un servidor TCP, mientras que para el computador con Windows XP se utilizó un cliente TCP desde el cual se realiza el envío del stack para ser procesado en la platafroma y recibe el resultado que ésta retorna. La comunicación entre el microscopio y el computador ya estaba establecida.

Al final de esta etapa del desarrollo se tiene el piloto para el Laboratorio de óptica definido, implementado y listo para ejecutar el algoritmo de procesamiento de imágenes.

6.3. IMPLEMENTACIÓN DEL EDF EN PARALELO SOBRE GPU

La etapa final del desarrollo está centrada en la programación del algoritmo EDF en paralelo sobre GPU para ser ejecutado en la Plataforma Nvidia Jetson TK1.

El modelo de programación utilizado es CUDA, específicamente creado para programar sobre las GPU de Nvidia, en su versión 6.5.45, y trabajando junto con el lenguaje de programación C.

Antes de programar, es necesario entender el proceso y funcionamiento con que trabaja el algoritmo de Profundidad de Campo Extendida EDF (Explicado en el numeral 5).

Así, de las 5 secciones del método presentadas, como se observa en la **Figura 10**, la primera, la adquisición del stack de imágenes (color azul), se obtiene en el software AxioVision en el computador de escritorio acoplado al microscopio y las otras 4 (color naranja), se envían a procesar sobre la tarjeta Jetson TK1, para finalmente regresar el resultado al computador de escritorio para visualizar.

Figura 10. Pasos del método de Profundidad de Campo Extendida EDF

Luego de esto, se definió implementar el método utilizando el operador de Varianzas y calculado sobre la Matriz G (Matriz de color verde) con una máscara de 3x3, por las razones expuestas en el numeral 5.

Una vez definidas las características que se usarían para el método se procede a implementar una versión inicial del algoritmo serial en lenguaje C para observar el comportamiento, el flujo de información, algunos aspectos de programación y los resultados esperados, en este caso, la imagen 3D. (Disponible en el repositorio: http://forge.sc3.uis.edu.co/redmine/projects/project-mpimiento/repository)

6.3.1. Antecedente de Implementación para un Supercomputador

Para la programación se hizo una revisión de la implementación del algoritmo realizada por la ingeniera Mónica Hernández presentada en el Estado del arte (Numeral 2.1).

Esta implementación se compone de 4 códigos o scripts, uno en Matlab y 3 en C con una versión más antigua de CUDA, pensados para ser ejecutados en un supercomputador, con altas capacidades de memoria RAM y almacenamiento. A continuación, se explicará brevemente cómo fueron implementados.

- **Código 1: Descomposición de Matrices en RGB:**

La descomposición de las Matrices no fue programada en lenguaje C, como el resto de la implementación, para esta se utilizó la herramienta Matlab, y con ella se generaban 3 archivos llamados R, G y B para cada imagen del stack.

- **Código 2: Cálculo de las Matrices de Varianza:**

La entrada para este código es el nuevo stack de componentes RGB creado con Matlab. Ahora, para realizar los cálculos de las varianzas la máscara generaba complicaciones en las esquinas y bordes, y para ello agregaron 2 columnas y 2 filas a la matriz de la componente verde (matriz G) en cada una en uno de los extremos de la misma, con valor cero.

Cada matriz G es procesada de manera serial, pero se calcula la varianza a cada elemento de dicha matriz en paralelo, donde el número de hilos es igual al número de elementos que tiene la matriz G original. Obteniendo un archivo con la matriz de varianzas por cada imagen procesada.

- **Código 3: Cálculo de la Matriz de Topografía:**

Lee cada uno de estos archivos y se invoca un kernel de CUDA con un número de hilos igual al número de elementos de la matriz de la matriz G, en él se hace la comparación, elemento a elemento, para encontrar el máximo de varianza y, por ende, la matriz de topografía, que almacena en un archivo.

- **Código 4: Cálculo de las Matrices RGB finales:**

Lee el archivo de matriz de topografía, crea un ciclo para leer los componentes RGB de cada imagen e invoca un kernel de CUDA con un número de hilos igual al número de elementos de la matriz de topografía, en él compara si la topografía coincide con el número de imagen, para actualizar las matrices RGB finales con estos colores, las cuales al final almacena en 3 archivos.

6.3.2. Implementación del algoritmo

La adquisición de imágenes se realizó con un Microscopio de Contraste por Interferencia Diferencial (DIC) diseñado para observar relieves en superficie de especímenes voluminosos, o difíciles de manejar o muy gruesos como para ser observados en el microscopio de fase.

Esta implementación se compone de un único código o script programado en C CUDA.

Análisis de paralelización:

En el análisis se observa que el algoritmo serial presenta gran cantidad de ciclos que operan sobre matrices, siendo las matrices la representación de las imágenes; especialmente en

el cálculo de la varianza, donde para cada elemento debe leer sus elementos vecinos, efectuar el cálculo de la varianza y almacenarlo en una nueva matriz.

Sin embargo, se observa dependencia en el cálculo de la topografía, lo que no es bueno cuando se desea paralelizar, allí se requiere hacer comparaciones con todas las matrices de varianza para obtener el mayor valor. Además, hay una limitación de hardware importante a considerar y es la memoria de la plataforma embebida, esta cuenta con 2GB en RAM y 16GB en ROM, por lo que se espera hacer la menor cantidad de copias a memoria.

Por las razones descritas, la propuesta implementada se centra en mantener en todo el tiempo de ejecución temporales de las 4 matrices finales (Matriz de topografía y Matrices RGB), de manera que no deba almacenar matrices de varianza para cada imagen, y en paralelizar el cálculo de la varianza.

Detalles de implementación:

La entrada para el algoritmo es el stack de imágenes en formato jpg.

El algoritmo propuesto cuenta con un gran ciclo for que recorre todas las imágenes del stack, así dentro de este ciclo:

1. Lee la imagen, obtiene sus componentes RGB utilizando la librería jpeglib y lo almacena en matrices llamadas R,G,B respectivamente (Estas matrices son reutilizadas cada vez que procesa una imagen).

2. Transfiere de la memoria de la CPU (Host) hacia la GPU (Device) las matrices de máximo de varianzas, topografía y RGB finales (Estas matrices se actualizan en cada ciclo y contienen el resultado del algoritmo), y las matrices RGB de la imagen actual.

3. Envía a procesar a la GPU, invocando un kernel CUDA con un número de hilos igual a la cantidad de elementos de la imagen, es decir, cada elemento de la matriz es procesado por separado en un hilo.

3.1. Calcula la varianza para dicho elemento con la matriz G, con una máscara de 3x3
3.2. Compara si la varianza actual es mayor que el máximo de varianza global, si es así actualiza el valor del máximo y al tiempo actualiza las matrices temporales de topografía y RGB finales

4. Transfiere de la memoria de la GPU (Device) hacia la CPU (Host) las matrices de máximo de varianzas, topografía y RGB finales.

5. Fin del ciclo

De esta manera al terminar todas las iteraciones ya tiene las 4 matrices de resultado, topografía, R, G y B; y sólo es necesario almacenarlas en archivos.

Ganancia:

Con la estrategia utilizada se logra el objetivo de disminuir al máximo las copias temporales a memoria, sin perder las ventajas de la paralelización en la GPU; lo que si ocurría en la implementación para el supercomputador, donde no sólo era costoso en términos de almacenamiento tener que guardar toda esa cantidad de archivos, 4 por cada imagen del stack para ser exactos, sino también en términos de tiempo, porque leer y escribir consume un tiempo considerable, y cada vez que cambiaba de código debía abrir estos archivos. Este código podía ser soportado por un supercomputador, pero no una plataforma embebida como la que se está trabajando para este proyecto.

Con un manejo cuidadoso de los índices fue posible superar la complicación en las esquinas, donde no se tenían vecinos para el cálculo de la varianza con la máscara, sin la necesidad de agregar a la Matriz G las 2 columnas y 2 filas, que aumentaba el tamaño de la matriz y, por ende, la complejidad de cálculo y tiempo de transferencia.

También, al tener unificados todos los códigos y utilizar una sola herramienta de programación para todos los pasos del algoritmo, hace que el procesamiento sea más fluido y que la portabilidad del código sea más sencilla, porque todo se encuentra programado en C con CUDA.

Código:

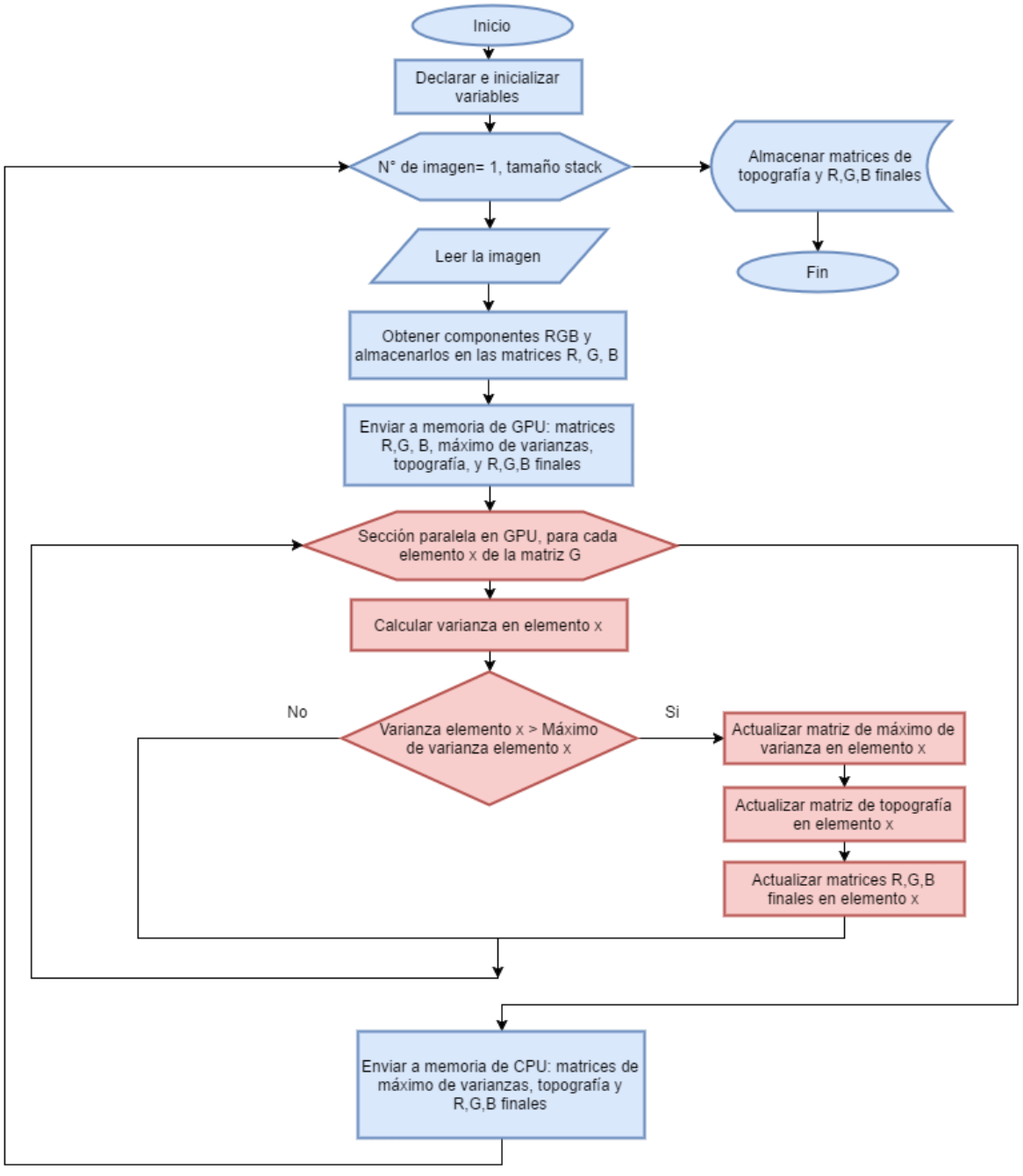
Los códigos implementados, en paralelo utilizando C CUDA y en serial utilizando C, están disponibles en el repositorio: http://forge.sc3.uis.edu.co/redmine/projects/project-mpimiento/repository

Ambos códigos manejan la misma lógica de programación, en cuanto al flujo de datos, manejo de las matrices temporales, y no almacenamiento de archivos de temporales, la diferencia radica en que los procesos que son paralelizados para la GPU, en serial son ejecutados con ciclos for, por lo que el tiempo es mayor.

A continuación, en la **Figura 11**, se presenta el diagrama de flujo diseñado para la implementación en paralelo con GPU del algoritmo EDF, las acciones en serial que ejecuta la CPU (Host) se muestran en color azul, y las acciones que se ejecutan en paralelo sobre los cores de GPU (Device) se muestran en rojo.

Diagrama de Flujo del Algoritmo EDF en Paralelo con GPU

Figura 11. Diagrama de flujo del algoritmo EDF en paralelo con GPU



7. PRUEBAS Y ANÁLISIS DE RESULTADOS

Para la captura de los stack de imágenes y la realización de las pruebas se utilizó el piloto de la arquitectura propuesta, la implementación del algoritmo EDF en C CUDA y los recursos del Laboratorio de Óptica del grupo GOTS.

Las pruebas de dividen en 2 secciones, en la primera el objetivo es evaluar la calidad de los resultados obtenidos con el algoritmo implementado, es decir, la calidad de la reconstrucción 3D de la imagen, y en la segunda el objetivo es evaluar la eficiencia en términos de tiempo de ejecución, tanto del algoritmo como de la implementación del piloto.

7.1. PRUEBAS DE CALIDAD DE RESULTADOS

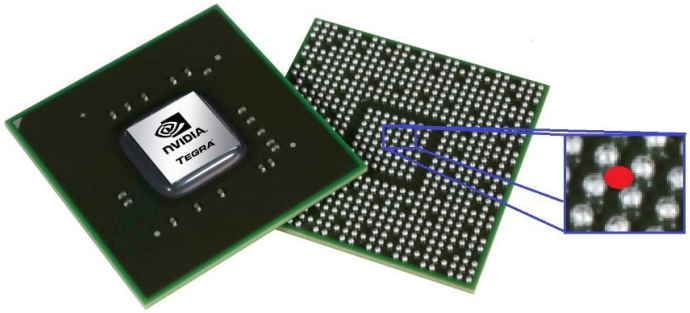
Para evaluar la calidad de las imágenes obtenidas, se realizó la visualización de las imágenes de resultado, topografía, R, G y B, para reconstruir la imagen 3D final, utilizando el software ImageJ con el plugin “Interactive 3D Surface Plot” que permite visualizar la imagen 3D de manera interactiva y modificar algunos de los parámetros.

7.1.1. Stack 1: Chip Nvidia

Las características del stack son las siguientes:

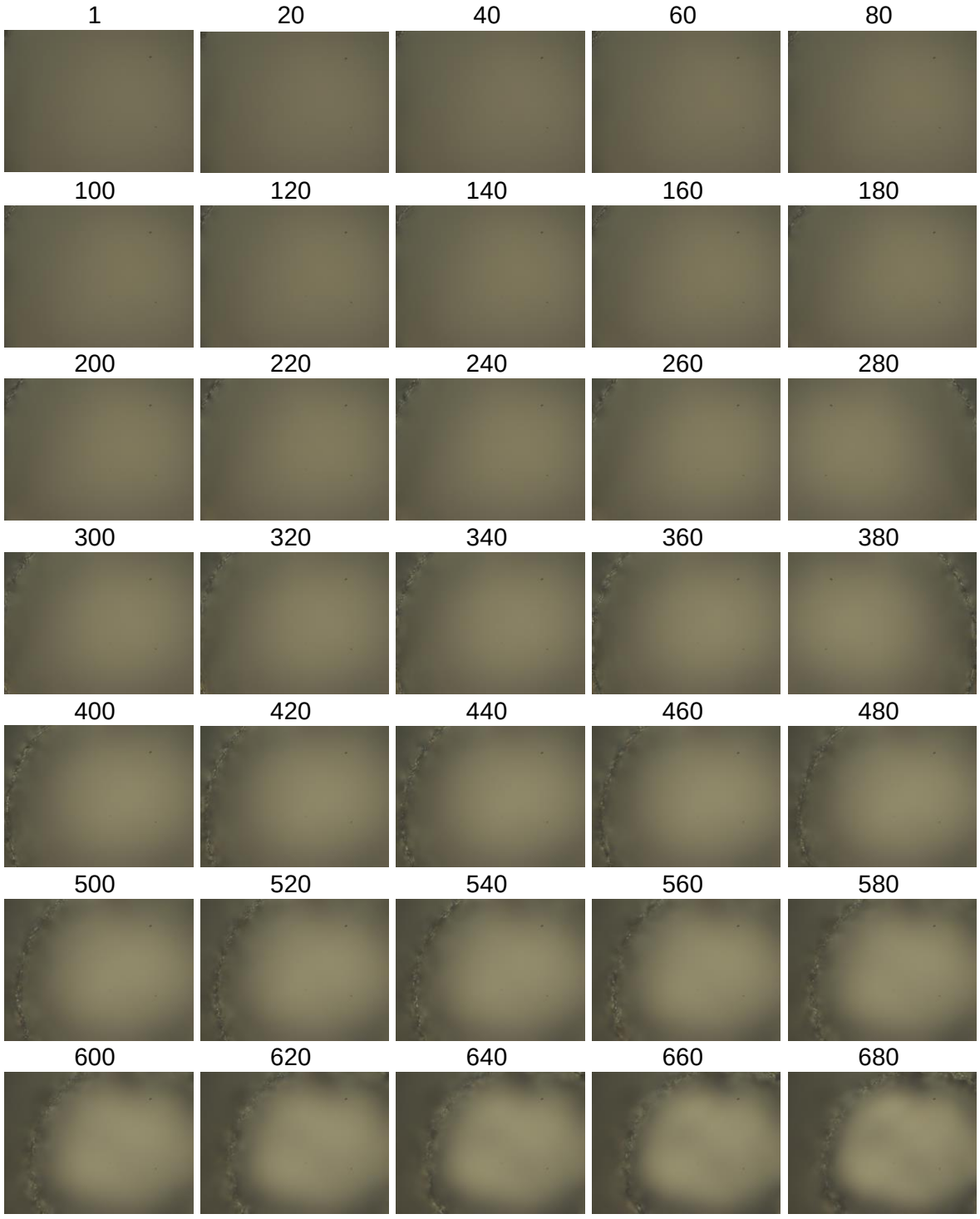
- Microscopio: Axio Imager Z1m
- Objetivo del microscopio: 50x
- Paso: 200nm
- Tamaño del stack: 843 imágenes
- Resolución de las imágenes: 1388 x 1040
- Objeto analizado: Chip NVIDIA, se tomó captura de uno de los espacios que quedan entre los pines (Punto rojo de la **Figura 12**)

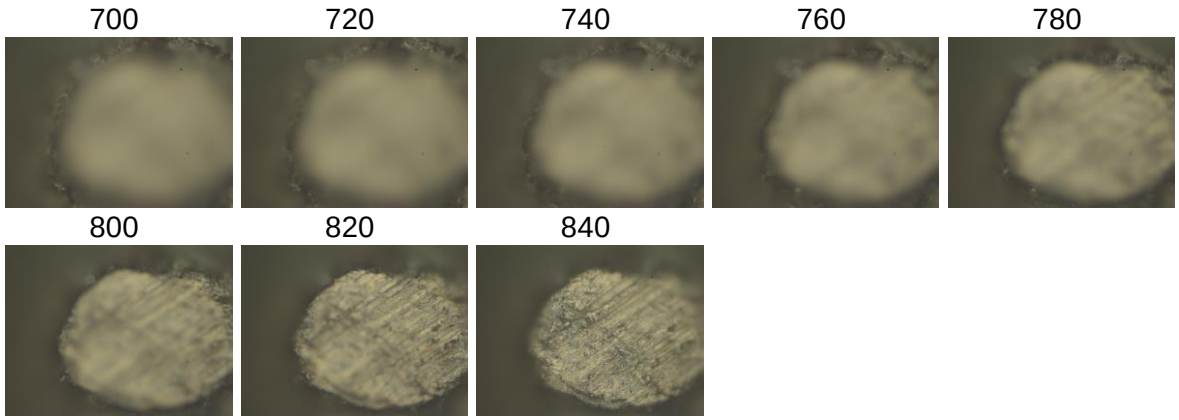
Figura 12. Objeto stack 1: Chip NVIDIA



Imágenes de muestra del stack:

Figura 13. Imágenes de muestra del stack 1





La **Figura 13** muestra la manera en que el microscopio va focalizando a diferentes alturas.

Reconstrucción para el Stack 1

Imagen de Textura:

Figura 14. Imagen R (Rojo) del stack 1

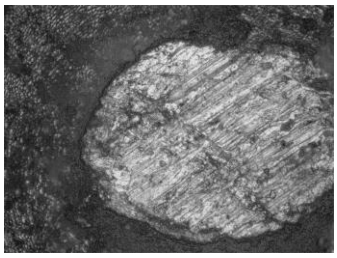


Figura 15. Imagen G (Verde) del stack 1

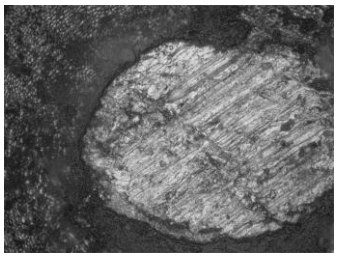


Figura 16. Imagen B (Azul) del stack 1

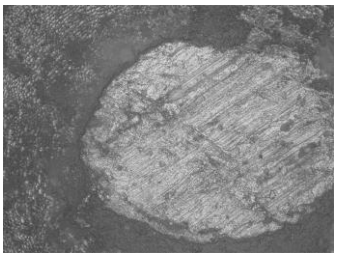


Figura 17. Imagen de Textura del stack 1

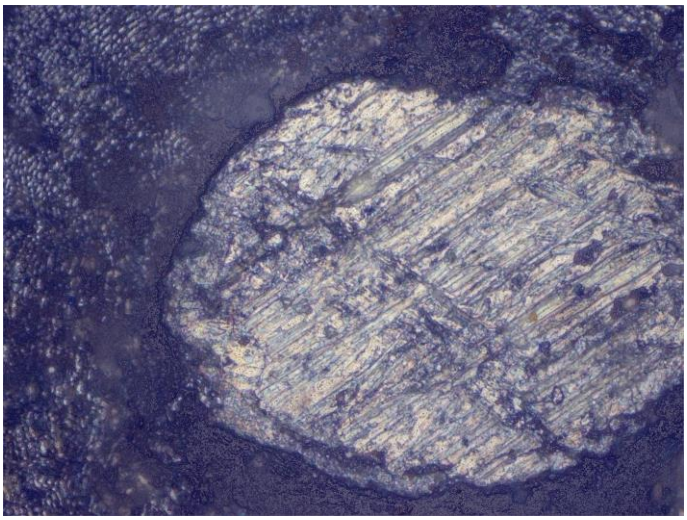


Imagen de topografía:

Figura 18. Imagen de topografía plana del stack 1

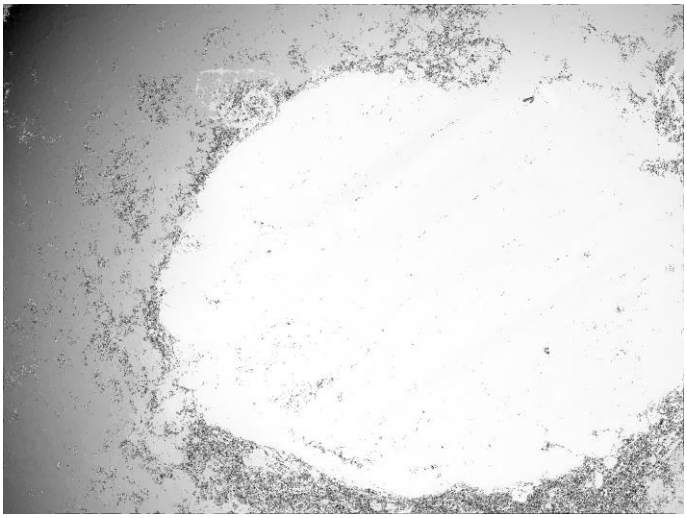


Figura 19. Imagen de topografía del stack 1, vista 1

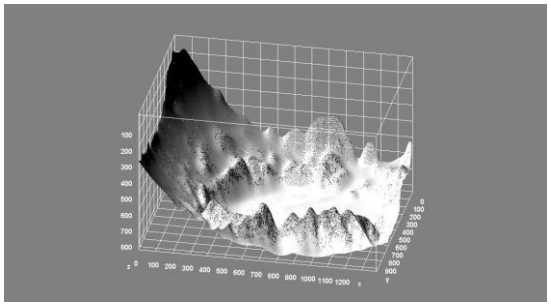


Figura 20. Imagen de topografía del stack 1, vista 2

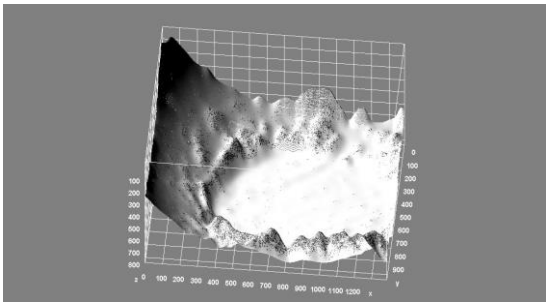


Imagen 3D final:

Figura 21. Imagen 3D del stack 1, vista 1

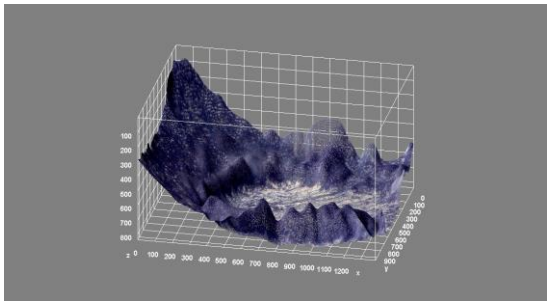
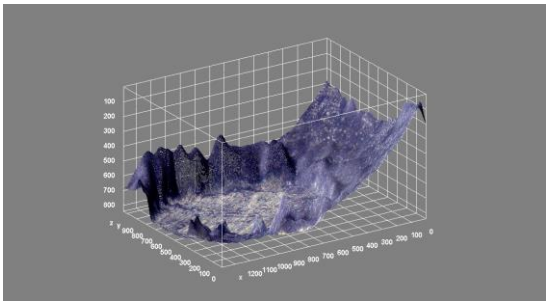


Figura 22. Imagen 3D del stack 1, vista 2



En las figuras presentadas se observa la reconstrucción del objeto, que corresponde a un agujero formado por el espacio entre los “pines”. Esta reconstrucción permite observar tanto la imagen de rango de topografía como la imagen de textura, gracias al material metálico del objeto que le brinda alta reflectividad. Allí se ve pequeñas áreas de desgaste que posee el metal.

También, se nota un cierto color azulado en la imagen, y esto se debe a que al aumentar de contraste en la imagen se pueden observar detalles en el color que no se veían antes.

7.1.2. Stack 2: Piedra de fantasía

Las características del stack son las siguientes:

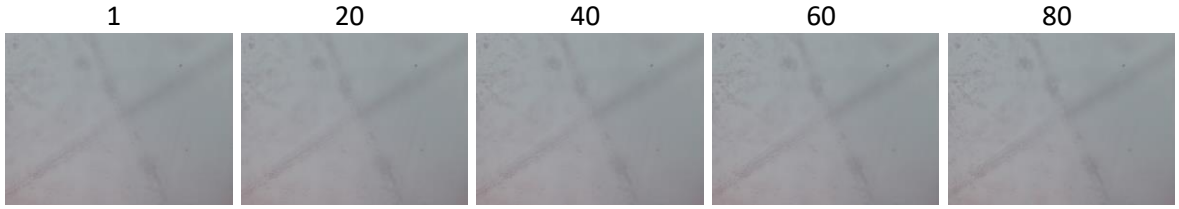
- Microscopio: Axio Imager Z1m
- Objetivo del microscopio: 20x
- Paso: 200nm
- Tamaño del stack: 536 imágenes
- Resolución de las imágenes: 1388 x 1040
- Objeto analizado: Piedra de fantasía para bisutería (La parte central de una de las piedras de la **Figura 23**)

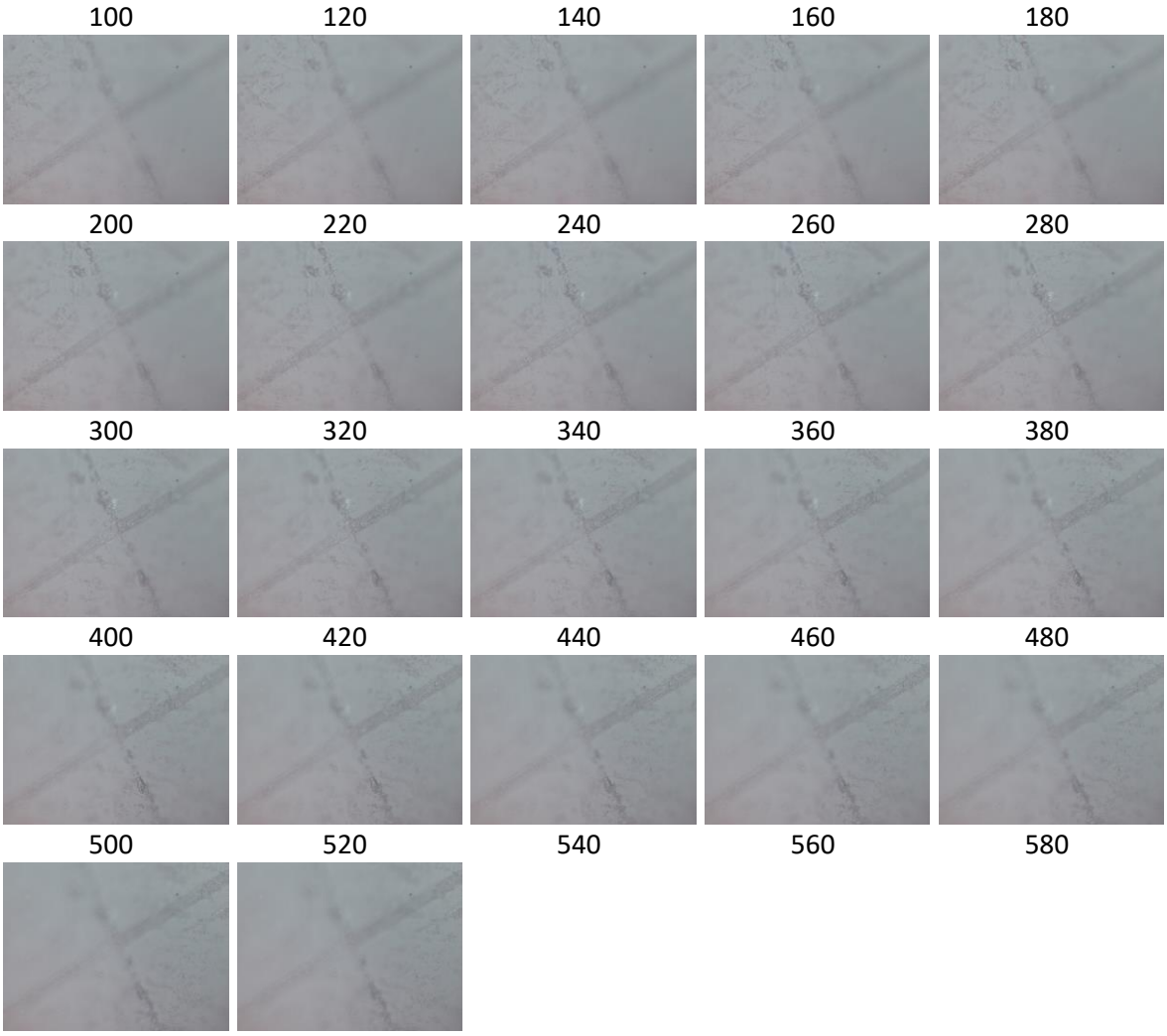
Figura 23. Objeto stack 2: Piedra de fantasía

Imágenes de muestra del stack:

Se presentan algunas de las imágenes del stack 2, a modo de muestra.

Figura 24. Imágenes de muestra del stack 2





Reconstrucción para el Stack 2

Imagen de Textura:

Figura 25. Imagen R (Rojo) del stack 2

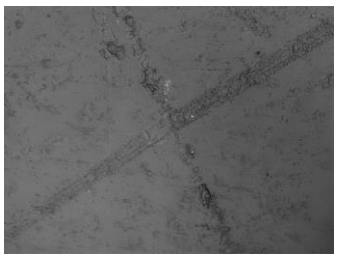


Figura 26. Imagen G (Verde) del stack 2

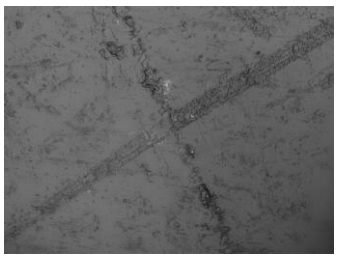


Figura 27. Imagen B (Azul) del stack 2

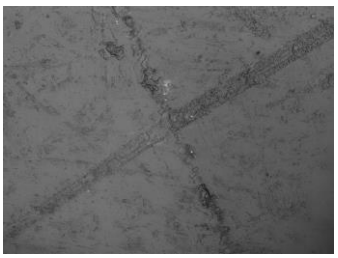


Figura 28. Imagen de Textura del stack 2

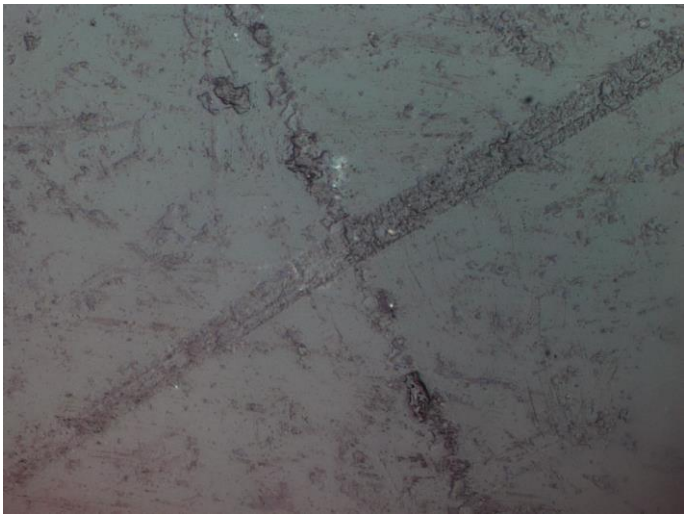


Imagen de Topografía:

Figura 29. Imagen de topografía plana del stack 2

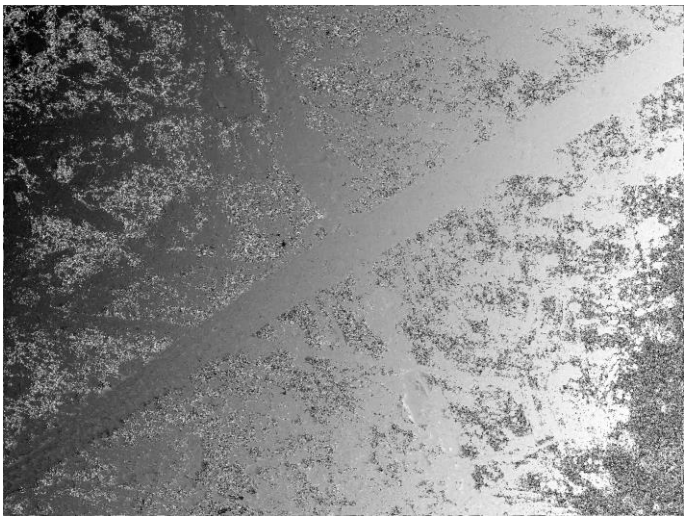


Figura 30. Imagen de topografía del stack 2, vista 1

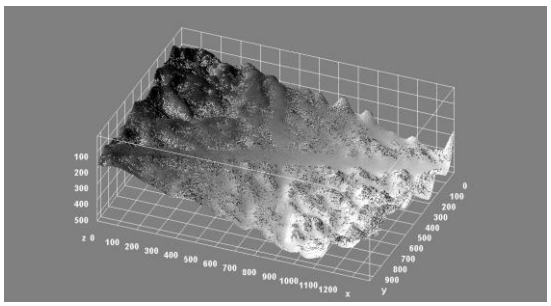


Figura 31. Imagen de topografía del stack 2, vista 2

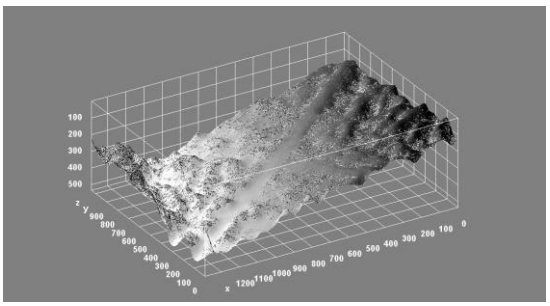


Imagen 3D final:

Figura 32. Imagen 3D del stack 2, vista 1

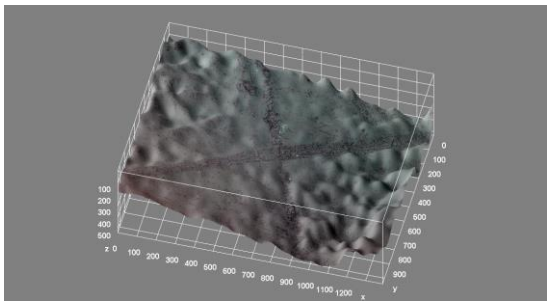
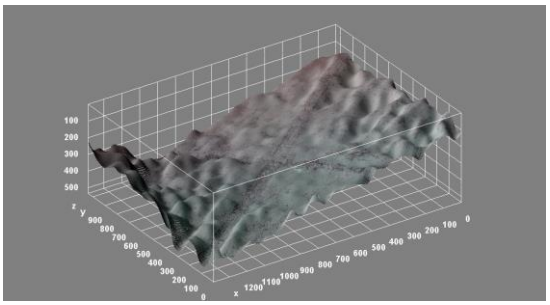


Figura 33. Imagen 3D del stack 2, vista 2



En las figuras presentadas se observa la reconstrucción del objeto, que corresponde a la parte central una piedra de fantasía. Esta reconstrucción no es tan clara como la del stack 1, porque el material no refleja tanto la luz, sin embargo, es posible observar la topografía y algunos detalles de la superficie irregular.

7.1.3. Comparación imágenes Implementación Piloto vs. Implementación para el Supercomputador

Para evaluar la calidad de los resultados obtenidos respecto a una implementación anterior se utilizó la implementación para el supercomputador presentada y descrita anteriormente en este documento (Numeral 2.1.TRATAMIENTO DE IMÁGENES USANDO EL MÉTODO DE PROFUNDIDAD DE CAMPO EXTENDIDA (EDF) EN ARQUITECTURAS PARALELAS [1]). Ahora, se describirán las características de los stacks.

Implementación del Supercomputador:

- Tipo de Microscopio: DIC
- Tamaño del stack: 93 imágenes
- Resolución: 1920 x 2560 píxeles
- Objeto: Piedra de fantasía

Implementación Piloto:

- Tipo de Microscopio: DIC
- Tamaño del stack: 843 imágenes
- Resolución: 1388 x 1040 píxeles
- Objeto: Chip NVIDIA (Stack 1)

Figura 34. Imagen de textura, Implementación Supercomputador [1]

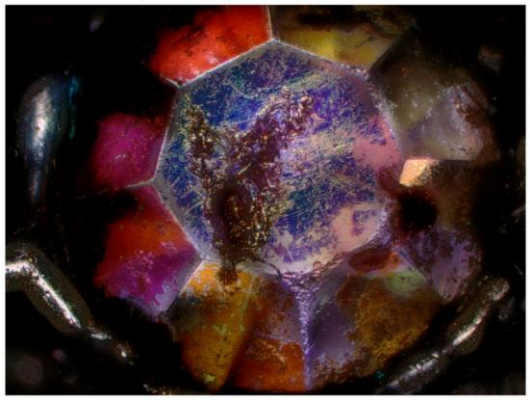


Figura 35. Imagen de textura, Implementación Piloto

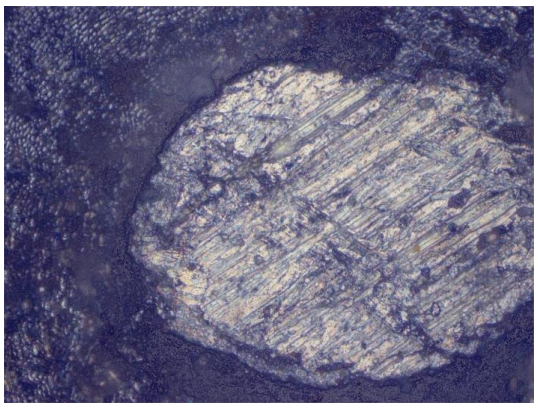


Figura 36. Imagen de topografía, Implementación Supercomputador [1]

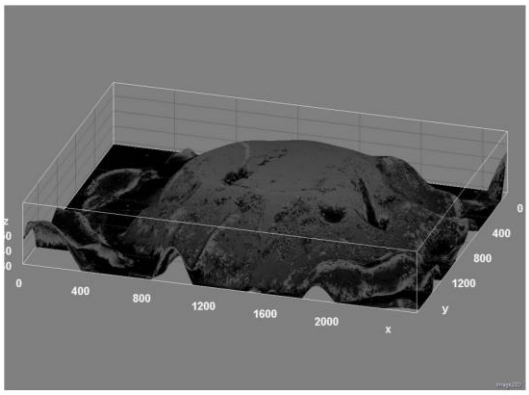


Figura 37. Imagen de topografía, Implementación Piloto

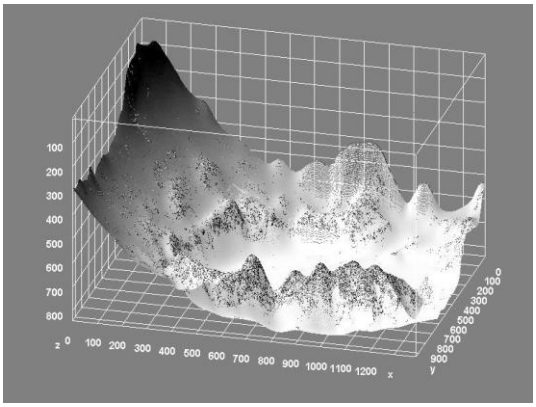


Figura 38. Imagen 3D, Implementación Supercomputador [1]

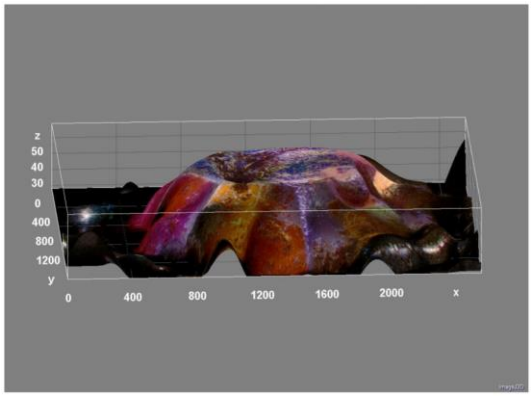
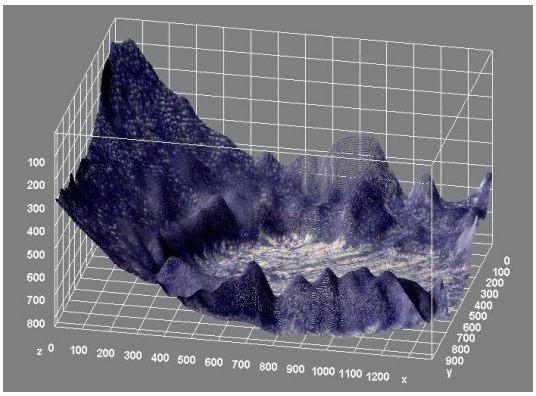


Figura 39. Imagen 3D, Implementación del Piloto



En las figuras anteriores se presenta los resultados obtenidos para las dos implementaciones, utilizando diferentes stacks.

En la implementación del Piloto se pueden observar más detalles en la topografía, como los pequeños desgastes del metal, y la textura presenta un contraste bastante alto, esto se debe principalmente que a la cantidad de imágenes que se pueden procesar es mucho mayor, 843 contra 93, sin embargo, la resolución es más baja, 1388x1040 contra 1920x2560.

7.2. PRUEBAS DE TIEMPOS DE EJECUCIÓN

7.2.1. Prueba 1: Tiempo de ejecución Serial vs. CUDA

Para analizar el rendimiento del método implementado en paralelo en C CUDA contra el mismo método implementado de manera serial en C, ejecutándose sobre la tarjeta Jetson TK1, todo esto con el objetivo de observar el rendimiento obtenido al paralelizarlo. Teniendo en cuenta que ambos códigos están programados siguiendo la misma lógica y utilizan las mismas librerías, a excepción de las necesarias para CUDA.

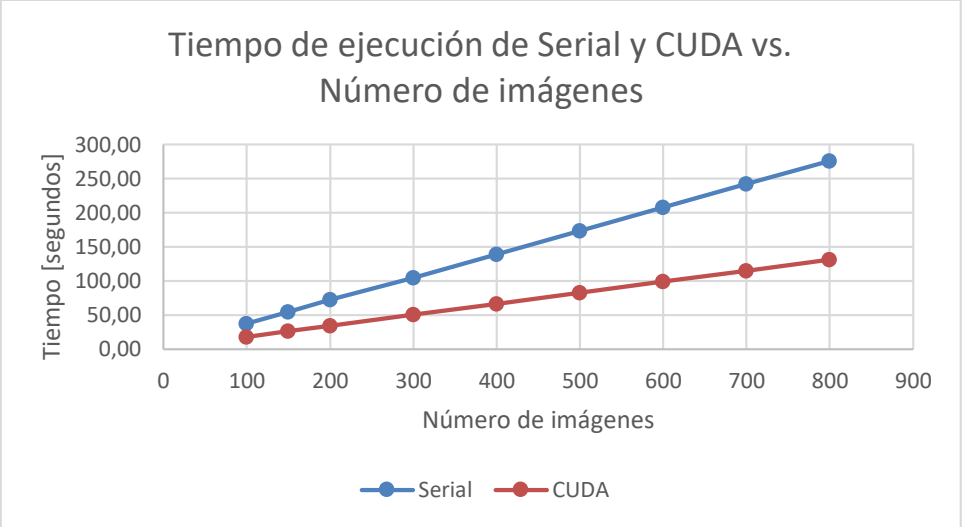
Tiempo de ejecución:

Se realizó la toma de los tiempos para diferente número de imágenes procesadas, las cuales se tomaron desde 100 hasta 800 en intervalos de 100.

Tabla 3. Tiempo de ejecución para Serial y CUDA, según el número de imágenes procesadas

Número de imágenes	Serial [segundos]	CUDA [segundos]
100	37,10	17,85
200	72,26	34,14
300	104,87	50,20
400	139,14	66,49
500	173,41	82,50
600	207,64	98,75
700	241,91	114,81
800	275,98	131,14

Figura 40. Tiempo de ejecución Serial vs. CUDA, según el número de imágenes procesadas



En la **Figura 40** se muestra el comportamiento de cada paradigma utilizado para implementar el algoritmo, serial y masivamente paralelo en GPU con CUDA, para diferente número de imágenes a procesar.

Como se puede observar, para ambos casos se presenta un crecimiento directamente proporcional y lineal del tiempo de ejecución respecto al número de imágenes. Sin embargo, el procesamiento con CUDA ofrece un tiempo de ejecución menor que el Serial para todos los tamaños de stack experimentados, es decir, mayor eficiencia.

Aceleración:

Para calcular la aceleración se utiliza la pendiente de la recta de cada una de las implementaciones, siendo

$$m = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1}$$

Donde, m es la pendiente, x es el eje de las abscisas, y es el eje de las ordenadas.

Así, la pendiente de la recta para el algoritmo serial es

$$m_{Serial} = \frac{275{,}98 - 37{,}10}{800 - 100} = 0{,}341$$

y para el algoritmo en CUDA es

$$m_{Serial} = \frac{131{,}14 - 17{,}85}{800 - 100} = 0{,}162$$

Ahora, se dividen las pendientes de la siguiente manera, para obtener la aceleración:

$$Aceleración_{Serial/CUDA} = \frac{m_{Serial}}{m_{CUDA}}$$

$$Aceleración_{Serial/CUDA} = \frac{0{,}341}{0{,}162} = 2{,}105$$

Por lo tanto, se tiene una aceleración de 2,1X con el algoritmo paralelo en CUDA, es decir, es 2,1 veces más rápido el algoritmo en CUDA que el algoritmo serial.

Tiempo promedio por imagen:

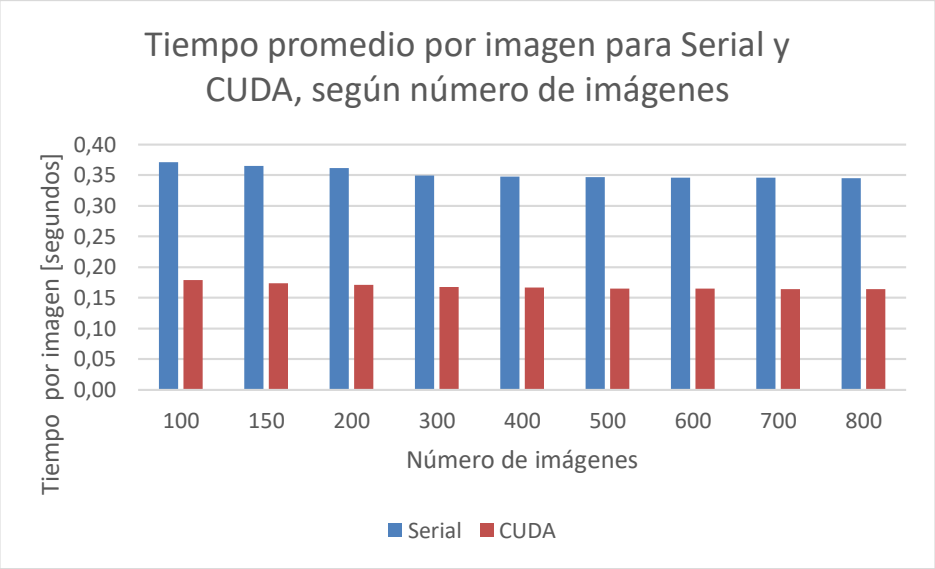
También se realizó el cálculo del tiempo promedio en que cada algoritmo tarda para procesar una imagen, tomando el tiempo total y dividiéndolo en el número de imágenes.

Tabla 4. Tiempo promedio por imagen para Serial y CUDA, según el número de imágenes

Número de imágenes	Serial [segundos]	CUDA [segundos]
100	0,37	0,18
150	0,36	0,17

200	0,36	0,17
300	0,35	0,17
400	0,35	0,17
500	0,35	0,16
600	0,35	0,16
700	0,35	0,16
800	0,34	0,16

Figura 41. Tiempo promedio por imagen para Serial y CUDA, según el número de imágenes



En la **Figura 41** se observa que el tiempo promedio al utilizar cada algoritmo tiende a disminuir ligeramente al aumentar el número de imágenes, siendo de aproximadamente 0,35 segundos para el Serial y 0,17 segundos para CUDA. Es claro que el tiempo para el procesamiento con CUDA es menor que el tiempo para el Serial.

7.2.2. Prueba 2: Tiempo de ejecución de CUDA vs. Código implementado para el Supercomputador

Ahora, se desea comparar los tiempos de procesamiento obtenidos con los códigos en CUDA ejecutados en el Supercomputador [1] con los tiempos para el código implementado en CUDA y ejecutado en la Jetson TK1.

Los stacks utilizados para cada uno se encuentran descritos en el numeral 7.1.3.

Sin embargo, de este trabajo no se tiene tablas de datos con los resultados de los tiempos para diferente número de imágenes, pero si se tiene 2 mediciones hechas para los stacks analizados en él. De esta manera se construye una pequeña tabla de tiempo.

Tabla 5. Tiempo de ejecución del Código para el supercomputador y el código implementado en CUDA

Número de Imágenes	Código para Supercomputador				Código para Jetson TK1
	Varianza	Topografía	RGB	**Total**	
93	350,66	165,92	198,67	**715,25**	**16,61**
149	534,52	309,08	319,3	**1162,9**	**26,13**

La **Tabla 5** muestra que la ganancia en tiempo con el nuevo código implementado en CUDA es realmente grande comparada con el código anterior, como se puede observar para procesar 149 imágenes el código en el supercomputador tardó 19,38 mientras que el nuevo código en la plataforma Jetson TK1 tardó sólo 26,13 segundos, por lo que hubo un ahorro de 18,95 minutos.

Pero ésta no es una comparación que se pueda realizar directamente, porque para el código del supercomputador no se toma en cuenta el tiempo del proceso de obtención del RGB para cada una de las imágenes en Matlab, el cuál era considerablemente alto, y ésta etapa del proceso sí está presente en el tiempo tomado para el nuevo código en CUDA. Además, la resolución de las imágenes procesadas, y por ende el tamaño de las matrices, fue mayor para la implementación del supercomputador, 1388x1040 contra 1920x2560 pixeles.

Por tanto, las comparaciones de tiempos y aceleración sirven para dar una idea de cada implementación, pero no pueden ser tomadas de manera exacta.

Aceleración:

Para calcular la aceleración se utiliza el método de las pendientes, explicado en la prueba 1 (7.2).

Como ya se comentó, no se tienen las tablas de datos, pero en las gráficas presentadas de ese trabajo se observa que el tiempo de ejecución es lineal y directamente proporcional al número de imágenes, por tanto, teniendo 2 puntos se puede obtener su pendiente.

Así, la pendiente de la recta para el código CUDA ejecutado en el supercomputador es

$$m_{Supercomputador} = \frac{1162{,}9 - 715{,}25}{149 - 93} = 7{,}994$$

y para el código CUDA ejecutado en la Jetson TK1

$$m_{Jetson\ TK1} = \frac{26{,}13 - 16{,}61}{149 - 93} = 0{,}170$$

Así, la aceleración es:

$$Aceleración_{Supercomputador/Jetson\ TK1} = \frac{7{,}994}{0{,}170} = 47{,}002$$

Por lo tanto, se tiene una aceleración de 47X para el tiempo de ejecución del código implementado en CUDA para la Jetson TK1, respecto a la implementación ejecutada en el supercomputador. Es decir, es 47 veces más rápido, considerando que para el código del supercomputador no se toma en cuenta el tiempo del proceso de obtención del RGB, pero que la resolución de las imágenes es mayor, como ya se explicó anteriormente.

7.2.3. Prueba 3: Tiempo de adquisición de imágenes

La prueba estuvo enfocada en mostrar el tiempo que toma el microscopio en obtener las imágenes. Utilizando el software AxioVision se define el punto más bajo y más alto a focalizar el objeto en el eje z, y se indica el tamaño del paso al que va a obtener las imágenes, lo que determina el número de imágenes que tendrá el stack.

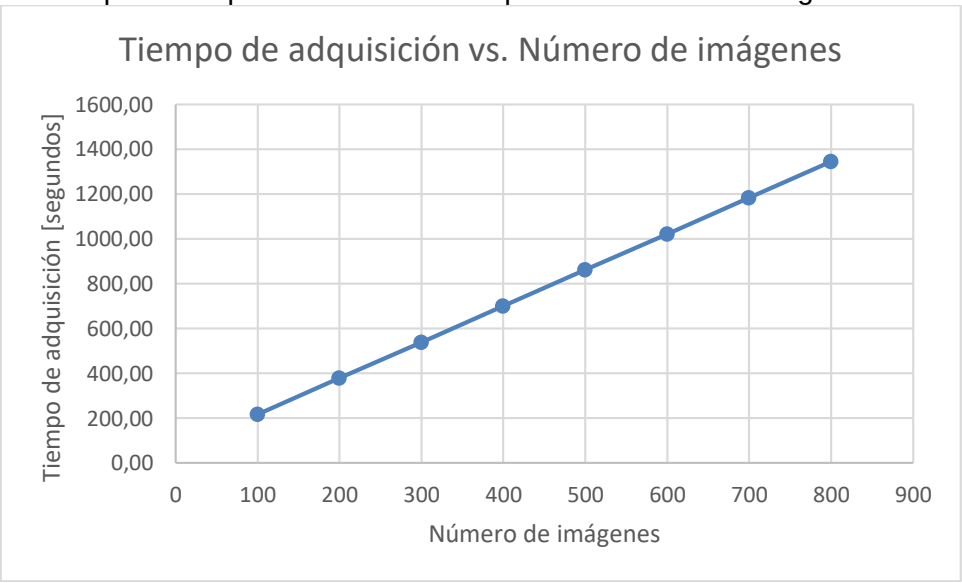
Tiempo total de ejecución:

Se realizó la toma de los tiempos para diferentes tamaños del stack, es decir, diferente número de imágenes, desde 100 hasta 800, en intervalos de 100.

Tabla 6. Tiempo de adquisición de imágenes en segundos y minutos

Número de Imágenes	Tiempo [segundos]
100	216,95
200	378,10
300	539,26
400	700,41
500	861,56
600	1022,71
700	1183,87
800	1345,02

Figura 42. Tiempo de adquisición del microscopio vs. Número de imágenes



Como se observa en la **Figura 42**, el tiempo de adquisición por parte del microscopio es directamente proporcional al número de imágenes y es relativamente alto, ya que para obtener un stack de 800 imágenes tarda 1345.02 segundos, aproximadamente 22,42 minutos.

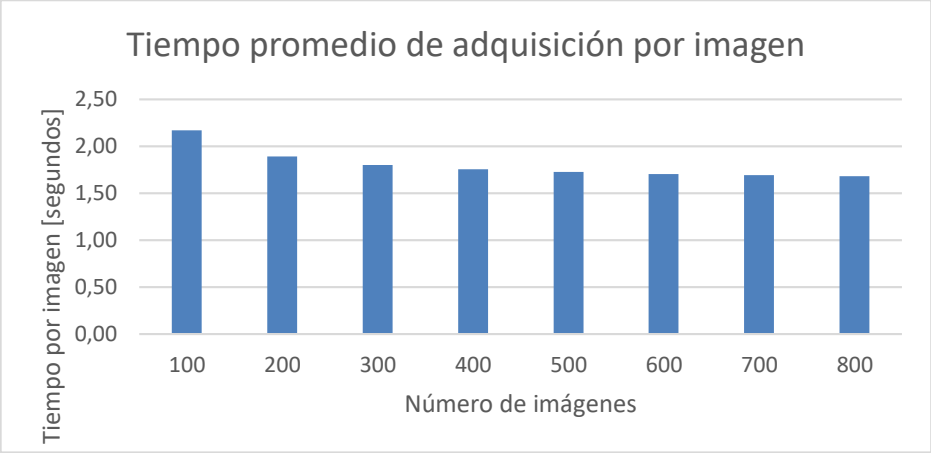
Tiempo promedio por imagen:

Ahora, para conocer el tiempo promedio por imagen, se toma el tiempo total y se divide en el número de imágenes para dicho tiempo.

Tabla 7. Tiempo promedio de adquisición por imagen

Número de Imágenes	Tiempo por imagen [segundos]
100	2,169522761
200	1,890522761
300	1,797522761
400	1,751022761
500	1,723122761
600	1,704522761
700	1,691237047
800	1,681272761

Figura 43. Tiempo promedio de adquisición por imagen vs. Número de imágenes



En la **Figura 43** se muestra que el tiempo promedio de adquisición de una imagen por parte del microscopio disminuye un poco a medida que aumenta el número de imágenes, y en promedio para las pruebas realizadas es de 1,7 segundos.

7.2.4. Prueba 4: Tiempo total del Piloto vs. Implementación del Laboratorio

Analizar el rendimiento de toda la implementación realizada para el piloto, incluyendo el tiempo de adquisición de imágenes, comunicación y procesamiento en C CUDA sobre la Jetson TK1 contra la implementación que ya tenían y utilizaban en el laboratorio, la cual está desarrollada en Visual Basic y procesa las imágenes a medida que las obtiene el software AxioVision, esto con el objetivo de observar el rendimiento total obtenido al implementar la arquitectura. La implementación existente será nombrada como “Implementación del Laboratorio”.

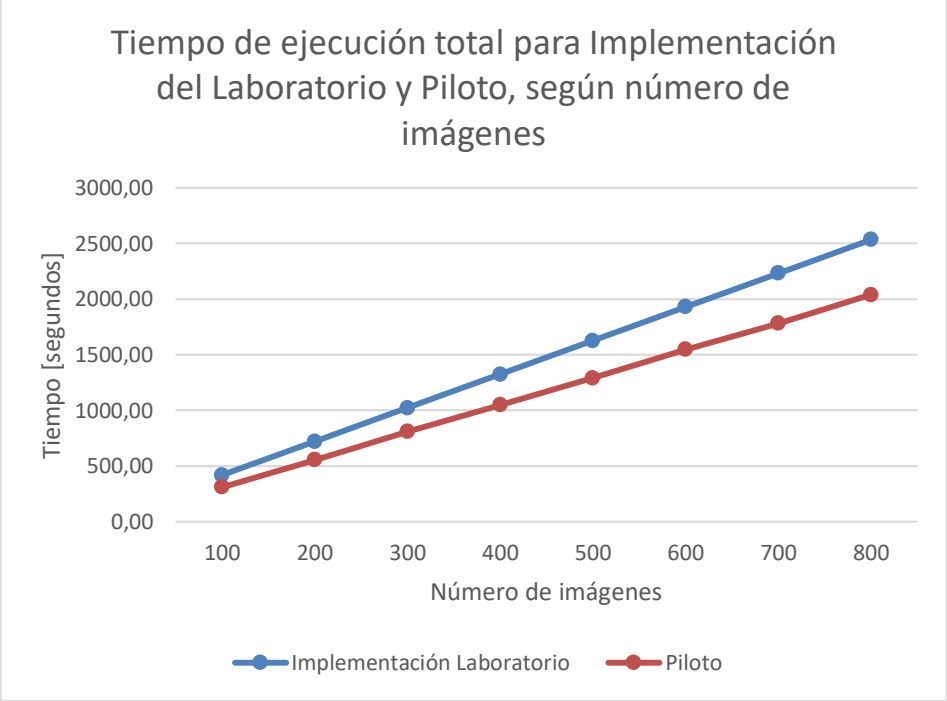
Tiempo de ejecución:

Se realizó la toma de los tiempos para diferente número de imágenes procesadas, desde 100 hasta 800, en intervalos de 100.

Tabla 8. Tiempo de ejecución total de la Implementación del Laboratorio vs. el Piloto, según número de imágenes

Número de imágenes	Implementación del Laboratorio [segundos]	Piloto [segundos]
100	418,06	309,63
200	720,32	555,14
300	1022,59	807,97
400	1324,85	1047,65
500	1627,11	1289,71
600	1929,37	1546,36
700	2231,64	1781,14
800	2533,90	2037,74

Figura 44. Tiempo de ejecución total de la Implementación del Laboratorio y el Piloto, según número de imágenes



La **Figura 44** permite observar que los tiempos de ejecución para ambas implementaciones son directamente proporcionales al número de imágenes, y el tiempo que emplea el piloto implementado es menor a la implementación que tenían en el laboratorio, por ejemplo, para procesar 800 imágenes la implementación del laboratorio requiere 42,23 minutos, mientras

que el piloto 33,96, ahorrando 8,27 minutos. Sin embargo, cabe resaltar que el tiempo del piloto está sujeto a la comunicación entre los componentes.

Aceleración:

Para calcular la aceleración se utiliza el método de las pendientes, explicado en la prueba 1 (7.2).

Así, la pendiente de la recta para la implementación del Laboratorio es

$$m_{Laboratorio} = \frac{2533{,}90 - 418{,}06}{800 - 100} = 3{,}023$$

y para la implementación del Piloto es

$$m_{Piloto} = \frac{2037{,}74 - 309.63}{800 - 100} = 2{,}469$$

Así, la aceleración es:

$$Aceleración_{Laboratorio/Piloto} = \frac{3{,}023,}{2{,}469} = 1{,}224$$

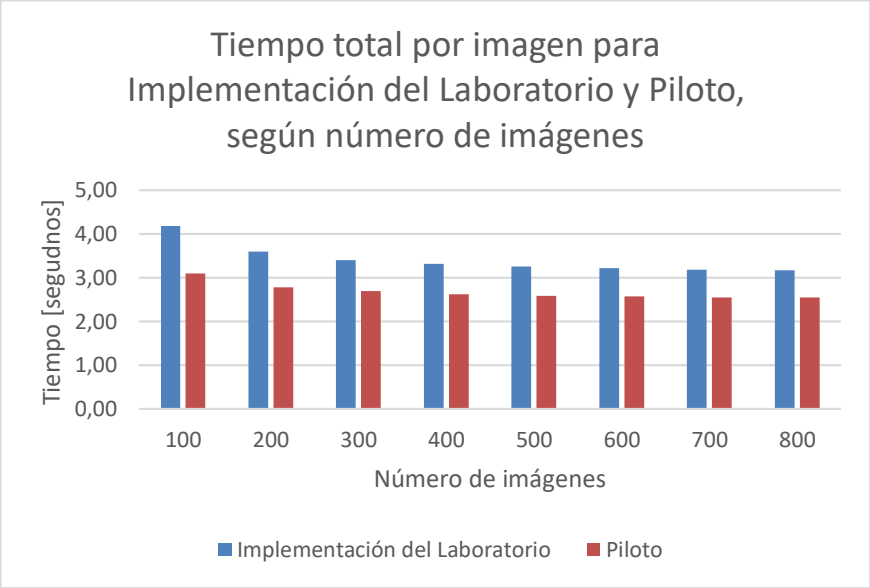
Por lo tanto, se tiene una ganancia en aceleración de 1,2x con el piloto implementado, respecto a la implementación del laboratorio, para el tiempo total de ejecución de las implementaciones, que incluye la adquisición de las imágenes.

Tiempo promedio por imagen:

Tabla 9. Tiempo por imagen de la Implementación del Laboratorio y el Piloto (incluido tiempo de adquisición), según número de imágenes

Número de imágenes	Implementación del Laboratorio [segundos]	Piloto [segundos]
100	4,18	3,10
200	3,60	2,78
300	3,41	2,69
400	3,31	2,62
500	3,25	2,58
600	3,22	2,58
700	3,19	2,54
800	3,17	2,55

Figura 45. Tiempo total por imagen de la Implementación del Laboratorio y el Piloto, según número de imágenes



En la **Figura 45** se observa que el tiempo promedio que tarda en procesar las imágenes con ambas implementaciones tiende a disminuir un poco al aumentar el número de imágenes, y para todos los casos medidos el tiempo del Piloto es inferior que el de la Implementación del Laboratorio.

7.2.5. Prueba 5: Tiempo de procesamiento del Piloto vs. Implementación del Laboratorio

Debido a que el tiempo de adquisición de las imágenes depende del microscopio y es fijo para cualquier implementación, se busca analizar el rendimiento del procesamiento, porque esta es la sección de la implementación que sí se puede modificar, y por ende acelerar.

Tiempo de ejecución:

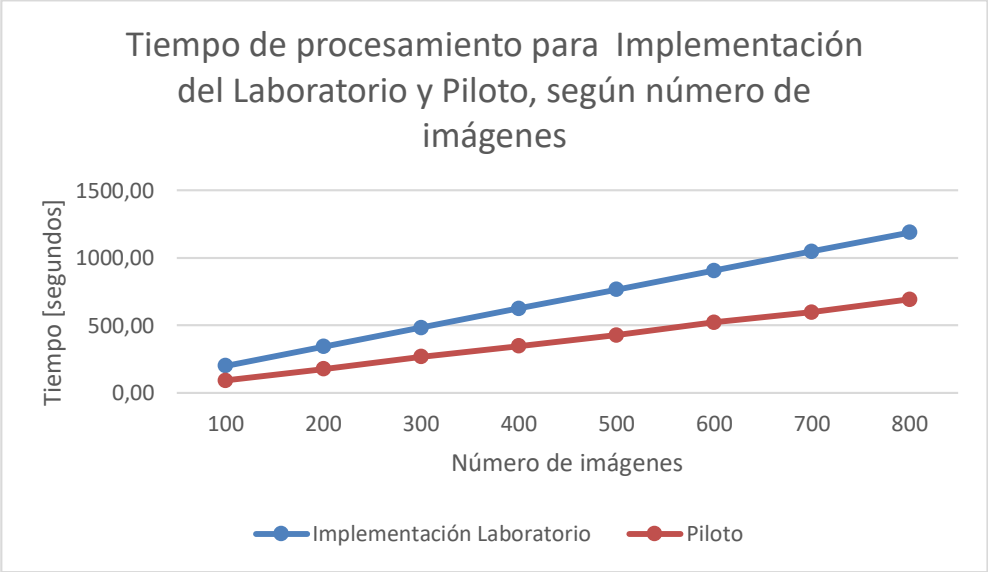
Se realizó la toma de los tiempos para diferente número de imágenes procesadas, desde 100 hasta 800, en intervalos de 100.

Tabla 10. Tiempo de procesamiento para la Implementación del Laboratorio y el Piloto, según número de imágenes

Número de imágenes	Implementación Laboratorio [segundos]	Piloto [segundos]
100	201,11	92,67
200	342,22	177,04
300	483,33	268,71
400	624,44	347,24
500	765,55	428,15

600	906,66	523,64
700	1047,77	597,27
800	1188,86	692,72

Figura 46. Tiempo de procesamiento para la Implementación del Laboratorio y el Piloto, según número de imágenes



En la **Figura 46** se observa que los tiempos de procesamiento para ambas implementaciones son directamente proporcionales al número de imágenes, y el tiempo de procesamiento para el piloto implementado es menor que el de la implementación del laboratorio. Por ejemplo, para procesar 800 imágenes la implementación del laboratorio requiere 19.81 minutos, mientras que el piloto 11.55, ahorrando 8.26 minutos.

Aceleración:

Para calcular la aceleración se utiliza el método de las pendientes, explicado en la prueba 1 (7.2).

Así, la pendiente de la recta para la implementación del Laboratorio es

$$m_{Laboratorio} = \frac{1188{,}86 - 201{,}11}{800 - 100} = 1{,}411$$

y para la implementación del Piloto es

$$m_{Piloto} = \frac{692{,}72 - 92{,}67}{800 - 100} = 0{,}857$$

Así, la aceleración es:

$$Aceleración_{Laboratorio/Piloto} = \frac{1{,}411}{0{,}857} = 1{,}646$$

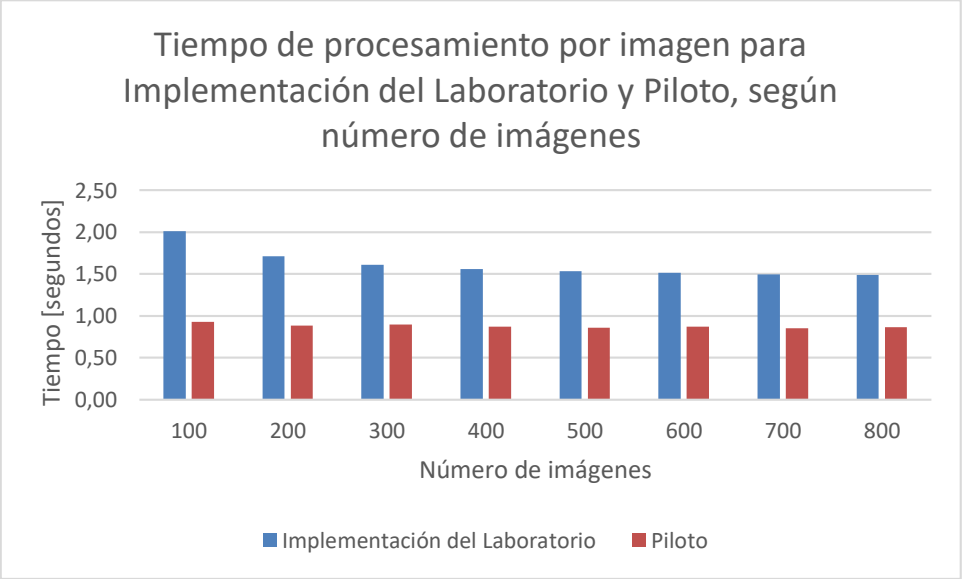
Por lo tanto, se tiene una ganancia en aceleración de 1,6x con el piloto implementado, respecto a la implementación del laboratorio, para el tiempo de procesamiento como tal, es decir, el procesamiento de la imagen y la comunicación entre los componentes.

Tiempo promedio por imagen:

Tabla 11. Tiempo de procesamiento por imagen para la Implementación del Laboratorio y el Piloto, según número de imágenes

Número de imágenes	Implementación Laboratorio [segundos]	Piloto [segundos]
100	2,01	0,93
200	1,71	0,89
300	1,61	0,90
400	1,56	0,87
500	1,53	0,86
600	1,51	0,87
700	1,50	0,85
800	1,49	0,87

Figura 47. Tiempo de procesamiento por imagen para la Implementación del Laboratorio y el Piloto, según número de imágenes



La **Figura 47** muestra que tiempo por imagen de procesamiento para ambas implementaciones tiende a disminuir un poco al aumentar el número de imágenes, y para todos los casos medidos el tiempo del Piloto es inferior que el de la Implementación del Laboratorio.

8. VENTAJAS Y LIMITACIONES

Luego de las pruebas y análisis realizados del diseño y la implementación del piloto, se presentan las principales ventajas y limitaciones que se encontraron, explicando las razones de cada una de ellas.

Ventajas:

- El piloto implementado efectivamente acelera el tiempo de ejecución para el procesamiento de imágenes mediante el uso de la plataforma embebida, específicamente 1,6X para el procesamiento como tal y 1,2X para la ejecución de todo el sistema, respecto a la implementación anterior del laboratorio.

- La plataforma embebida permite posee un bajo costo de implementación, ya que no requiere componentes muy especializados o costosos, la Nvidia Jetson TK1 cuesta 192 dólares, y puede ser acoplada al sistema del microscopio de una manera relativamente sencilla.

- La calidad de las imágenes obtenidas fue la requerida por los científicos, porque el tener una imagen de rango permite juzgar sobre las variaciones espaciales del objeto y las intensidades focalizadas juzgan todos los puntos en iguales condiciones.

Limitaciones:

- La memoria de este tipo de plataformas embebidas es una gran limitación, tanto la memoria RAM como el almacenamiento total del sistema, porque restringe el tamaño de problemas que pueden ser trabajos.

- La comunicación entre el computador y la plataforma embebida es un factor crítico en el tiempo, porque depende de las capacidades de la red y la vía por la que sea enviado, como wifi o por cable de red. Además, es superior al tiempo de procesamiento, por ejemplo, para 800 imágenes el procesamiento toma 2,19 minutos y la comunicación 9,36 minutos. Aun así, el tiempo total obtenido fue mejor que para las implementaciones previas.

9. CONCLUSIONES

Los sistemas embarcados o embebidos son una alternativa eficiente para acelerar el procesamiento y visualización de imágenes en el área de microscopía, ya que como se pudo observar no requieren componentes muy especializados, utilizando los componentes con que ya se cuenta en laboratorios se puede incluir una plataforma embebida de HPC para brindar mayor capacidad de cómputo sin hacer una inversión monetaria muy grande, ni incrementar considerablemente el consumo energético.

Se diseñó una arquitectura embarcada de cómputo de alto desempeño para el procesamiento y visualización de imágenes de microscopía, que se caracteriza por su bajo costo, facilidad de implementación arquitectural, y capacidades para acelerar el procesamiento de imágenes utilizando el modelo de flujo de ejecución de CUDA. También se identificó ciertas limitaciones, como la dependencia que tiene de la comunicación de los componentes y, la pequeña memoria que posee la plataforma embebida Jetson TK1. Además, se identificó otras posibilidades de diseño, como aprovechar las capacidades de la GPU de la plataforma embebida para visualizar los resultados, para lo cual se requiere incluir más componentes a la arquitectura, pero que para el caso implementado no fue necesario.

La implementación del piloto para el laboratorio de óptica del grupo GOTS cumplió con los resultados deseados, acelerar el tiempo de ejecución para el algoritmo de Profundidad de Campo Extendida (EDF) respecto a la implementación para este algoritmo que tenían en el laboratorio, específicamente 1,6X para el procesamiento como tal y 1,2X para la ejecución de todo el sistema, brindándole más potencia de cómputo al sistema; y validar la utilidad de la arquitectura propuesta. Además de ofrecer la posibilidad de utilizarlo para la implementación de más algoritmos que puedan ser programados de manera masivamente paralela sobre GPUs.

La implementación del algoritmo EDF paralelo aprovecha las ventajas de la GPU que posee la tarjeta Jetson TK1 utilizando C CUDA, y al compararlo con la implementación serial en C, en cuanto a calidad de imagen los resultados fueron iguales para ambos, y en cuanto a la velocidad de ejecución la implementación paralela fue mejor, teniendo una aceleración 2,1X. La ganancia en aceleración se debe a la alta cantidad de operaciones en el cálculo de la varianza sobre matrices que fueron masivamente paralelizadas con la GPU.

Las imágenes generadas y visualizadas brindaron el resultado que se deseaba, utilizando la herramienta ImageJ se visualiza la imagen 3D focalizada en todos los puntos, en la que se observa tanto la imagen de rango de topografía que permite juzgar las variaciones del objeto en el espacio, como la imagen de textura que permite tener las intensidades focalizadas y así juzgar todos los puntos en iguales condiciones, brindando una visualización más cercana a como vemos los objetos en “la vida real” y no como se observa en una imagen plana.

10. RECOMENDACIONES

El piloto propuesto para el laboratorio de óptica brinda la posibilidad de implementar nuevos algoritmos de procesamiento de imágenes utilizando los mismos componentes, lo que sería de utilidad, ya que además del método EDF, se realiza otro tipo de estudios y análisis de materiales que requieren diferentes algoritmos de procesamiento; como algoritmos para superresolución basado en stack de imágenes, los cuales están diseñados para aumentar la resolución espacial de una imagen.

Se recomienda utilizar otras plataformas para sistemas embebidos, como la nueva Nvidia Jetson TX1, que posee una CPU más veloz, más memoria RAM y más núcleos de GPU.

Considerando la limitante de la comunicación, se recomienda trabajar en un protocolo de comunicación para aumentar la velocidad de transmisión de datos entre los componentes.

También, se podría diseñar una arquitectura que trabaje directamente todos los procesos sobre la plataforma embebida de HPC, es decir, que prescinda del computador acoplado, pero para ello se requiere del apoyo e interés del fabricante, ya que es este quien posee todos los componentes para el manejo del microscopio.

REFERENCIAS

[1] HERNÁNDEZ, Mónica Liliana. *Tratamiento de Imágenes usando el Método de Profundidad de Campo Extendida (EDF) en Arquitecturas Paralelas*. Bucaramanga: Universidad Industrial de Santander, 2011.

[2] Nvidia Developer. *Stereolabs ZED Stereo Camera combined with Jetson TX1 brings advanced 3D mapping to drones*. Disponible en: https://developer.nvidia.com/embedded/learn/success-stories/stereolabs

[3] BJÖRGVINSDOTTIR, Hanna, *Face Recognition Based on Embedded Systems*. Lund: Lund University, 2016.

[4] THOMPSON, Andrew David. *Design and Implementation of an Embedded H.264 Color Video Encoding Pipeline for a Mobile Processing Platform*. Dayton: University of Dayton, 2016.

[5] Nvidia. *Jetson Embedded Hardware*. Disponible en: https://developer.nvidia.com/embedded/develop/hardware

[6] Nvidia. *Nvidia Jetson TK1, El kit de desarrollo integrado*. Disponible en: http://www.nvidia.es/object/jetson-tk1-embedded-dev-kit-es.html

[7] Nvidia. *Jetson TX1 Developer Kit*. Disponible en: http://www.nvidia.com/object/jetson-tx1-dev-kit.html

[8] Carl Zeiss. *Operating Manual Axio Imager, Upright Microscope*. Disponible en: http://biochimie.umontreal.ca/wp-content/uploads/sites/37/2016/02/zeiss_axio_imager_manual.pdf

[9] Lukas Microscope. AxioCam HR - Technical Specification. Disponible en: http://lukasmicroscope.com/html/axiocam_hrc-hrm.html

[10] Carl Zeiss. *AxioVision Users Guide*. Disponible en: http://twiki.cis.rit.edu/twiki/pub/MVRL/BioGigapan/AxioVision_Users_Guide_Rel_4.1.pdf

[11] ImageJ. *ImageJ*. Disponible en: https://imagej.net/Welcome

BIBLIOGRAFÍA

BJÖRGVINSDOTTIR, Hanna, *Face Recognition Based on Embedded Systems*. Lund: Lund University, 2016.

Carl Zeiss. *Operating Manual Axio Imager, Upright Microscope*. Disponible en: http://biochimie.umontreal.ca/wp-content/uploads/sites/37/2016/02/zeiss_axio_imager_manual.pdf

Carl Zeiss. *AxioVision Users Guide*. Disponible en: http://twiki.cis.rit.edu/twiki/pub/MVRL/BioGigapan/AxioVision_Users_Guide_Rel_4.1.pdf

eLinux wiki. *Jeston TK1*. Disponible en: http://elinux.org/Jetson_TK1

EYNARD, Aldo; VALENTICH, Mirta; ROVASIO, Roberto. *Histología y embriología del ser humano: bases celulares y moleculares.* Buenos Aires: Editorial Médica Panamericana, 2008. ISBN: 9789500606028

HERNÁNDEZ, Mónica Liliana. *Tratamiento de Imágenes usando el Método de Profundidad de Campo Extendida (EDF) en Arquitecturas Paralelas*. Bucaramanga: Universidad Industrial de Santander, 2011.

Inside HPC. *What is high performance computing?.* Disponible en: http://insidehpc.com/hpc-basic-training/what-is-hpc/

ImageJ. *ImageJ*. Disponible en: https://imagej.net/Welcome

Lukas Microscope. AxioCam HR - Technical Specification. Disponible en: http://lukasmicroscope.com/html/axiocam_hrc-hrm.html

MAESTÚ, Fernando, RÍOS, Marcos; CABESTRERO, Raúl. *Neuroimagen. Técnicas y procesos cognitivos*. Barcelona: Elsevier Masson, 2007. ISBN: 9788445817766

Nvidia. *Jetson Embedded Hardware*. Disponible en: https://developer.nvidia.com/embedded/develop/hardware

Nvidia. *GPU computing*. Disponible en: http://www.nvidia.es/object/gpu-computing-es.html

Nvidia. *Jetson TX1 Developer Kit*. Disponible en: http://www.nvidia.com/object/jetson-tx1-dev-kit.html

Nvidia. *Nvidia Jetson TK1, El kit de desarrollo integrado*. Disponible en: http://www.nvidia.es/object/jetson-tk1-embedded-dev-kit-es.html

Nvidia. *Procesamiento paralelo y CUDA*. Disponible en: http://www.nvidia.es/object/cuda-parallel-computing-es.html

Nvidia. *What is CUDA?.* Disponible en: https://blogs.nvidia.com/blog/2012/09/10/what-is-cuda-2/

Nvidia Developer. *Stereolabs ZED Stereo Camera combined with Jetson TX1 brings advanced 3D mapping to drones*. Disponible en: https://developer.nvidia.com/embedded/learn/success-stories/stereolabs

Técnica en Laboratorios. *Microscopía de Contraste Diferencial interferencial.* Disponible en: http://www.tecnicaenlaboratorios.com/Nikon/Info_dic.htm

THOMPSON, Andrew David. *Design and Implementation of an Embedded H.264 Color Video Encoding Pipeline for a Mobile Processing Platform*. Dayton: University of Dayton, 2016.

VENDRELL, Mario. Estudio Óptico por Microscopía de Reflexión. Disponible en: http://www.fempatrimoni.cat/www-crista/OPTICA/10-micro-reflexio.pdf