ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELECTRÍCA Y ELECTRÓNICA

BENCHMARKING PARA PROCESAMIENTO PARALELO: MEDIDA DEL DESEMPEÑO DE LA EJECUCIÓN PARALELA UTILIZANDO MATLAB

TRABAJO DE INTEGRACIÓN CURRICULAR PRESENTADO COMO REQUISITO PARA LA OBTENCIÓN DEL TÍTULO DE INGENIERA EN TELECOMUNICACIONES

Daysi Johanna Guano Moscuy
daysi.guano@epn.edu.ec

DIRECTOR: RAMIRO EDUARDO MOREJÓN TOBAR
ramiro.morejon@epn.edu.ec

DMQ, Octubre 2023

CERTIFICACIONES

Yo, Daysi Johanna Guano Moscuy declaro que el trabajo de integración curricular aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

Daysi Johanna Guano Moscuy

Certifico que el presente trabajo de integración curricular fue desarrollado por Daysi Johanna Guano Moscuy, bajo mi supervisión.

Ing. Ramiro Eduardo Morejón Tobar
DIRECTOR

DECLARACIÓN DE AUTORÍA

A través de la presente declaración, afirmamos que el trabajo de integración curricular aquí descrito, así como el (los) producto(s) resultante(s) del mismo, son públicos y estarán a disposición de la comunidad a través del repositorio institucional de la Escuela Politécnica Nacional; sin embargo, la titularidad de los derechos patrimoniales nos corresponde a los autores que hemos contribuido en el desarrollo del presente trabajo; observando para el efecto las disposiciones establecidas por el órgano competente en propiedad intelectual, la normativa interna y demás normas.

DAYSI GUANO RAMIRO MOREJON

DEDICATORIA

A mis padres, por su amor y compañía en cada paso que doy en la búsqueda de ser mejor persona y profesional.

A mí, por el esfuerzo y constancia en el cumplimiento de cada uno de mis objetivos.

AGRADECIMIENTO

He sido bendecido con el apoyo y la guía de muchas personas que han aportado mucho en este viaje académico. Este trabajo no es solo el resultado de mi trabajo; es el resultado de la generosidad y el compromiso de aquellos que me han acompañado a lo largo de este camino. Es un placer expresar mi gratitud más sincera.

A Dios y a la Virgen de Guadalupe, por guiarme en mi camino y por permitirme concluir con mi objetivo.

A mis padres, Margoth y Hugo, por su amor incondicional y por creer en mí desde el primer día. Por sus sacrificios y su apoyo constante que han sido la clave de mi éxito.

A mi hermana Karen, por estar siempre presente y por el apoyo moral que me ha brindado a lo largo de esta etapa.

A la persona que me apoyo durante el transcurso de este trayecto y siempre estuvo presente, Alex.

A mis amigos, Fernanda, Daniel, Pauli, Ibeth, Melanny, Leandro y Wellington, personas con las que compartí en las aulas y fuera de ellas, gracias por su ayuda desinteresada y su apoyo moral.

A mi familia, que siempre estuvo ahí con sus consejos y palabras de aliento, para poder ser mejor persona y lograr todas mis metas.

Finalmente, a la Escuela Politécnica Nacional, mi gratitud al Ing. Ramiro Morejón, quien me brindo su ayuda en la realización de este trabajo, siempre guiándome con sus conocimientos y profesionalismo.

¡Gracias!

ÍNDICE DE CONTENIDO

RESUMEN

En este proyecto, se lleva a cabo un análisis comparativo exhaustivo entre el procesamiento de imágenes en modalidad secuencial y paralela, con el objetivo de evaluar su impacto en términos de eficiencia y velocidad. La investigación se centra en la implementación y comparación de algoritmos de detección de bordes en imágenes, utilizando tanto enfoques secuenciales como paralelos. En el enfoque secuencial, se explora la aplicación convencional de algoritmos de procesamiento de imágenes, que procesan píxeles uno tras otro en un orden determinado. Este método se compara con el enfoque paralelo, que aprovecha los recursos de hardware modernos para procesar múltiples partes de la imagen simultáneamente. Se implementan algoritmos de detección de bordes en ambas modalidades y se comparan los tiempos de ejecución, la calidad de los resultados y el rendimiento general. Se utiliza un conjunto diverso de imágenes, que incluye variaciones de contenido y resolución, para evaluar el comportamiento de los enfoques en diferentes escenarios. Los resultados obtenidos de este análisis proporcionan información valiosa sobre la viabilidad y las ventajas del procesamiento de imágenes paralelo en comparación con el secuencial. Además, se destacan los casos de uso y las aplicaciones en las que uno u otro enfoque puede ser más beneficioso. Estos hallazgos tienen implicaciones directas en la optimización de algoritmos de procesamiento de imágenes para aplicaciones en tiempo real y análisis de grandes conjuntos de datos.

PALABRAS CLAVE: procesamiento de imágenes, secuencial, paralelo, detección de bordes, tiempo.

ABSTRACT

In this project, a comprehensive comparative analysis between sequential and parallel image processing is carried out to evaluate their impact in terms of efficiency and speed. The research focuses on the implementation and comparison of edge detection algorithms in images, using both sequential and parallel approaches. In the sequential approach, the conventional implementation of image processing algorithms, which process pixels one after another in a given order, is explored. This method is compared with the parallel approach, which takes advantage of modern hardware resources to process multiple parts of the image simultaneously. Edge detection algorithms are implemented in both modalities and run times, quality of results and overall performance are compared. A diverse set of images, including variations in content and resolution, is used to evaluate the behavior of the approaches in different scenarios. The results obtained from this analysis provide valuable information on the feasibility and advantages of parallel versus sequential image processing. In addition, use cases and applications where one or the other approach may be more beneficial are highlighted. These findings have direct implications for the optimization of image processing algorithms for real-time applications and analysis of large data sets.

KEYWORDS: image processing, sequential, parallel, edge detection, time.

1 INTRODUCCIÓN

El procesamiento lineal es un campo fundamental en el ámbito del procesamiento de señales y la teoría de sistemas. Se basa en la aplicación de operaciones lineales a las señales, como la convolución, la multiplicación por escalares y la suma ponderada, además tiene ventajas en términos de simplicidad, interpretación y eficiencia computacional.

A pesar de que el procesamiento lineal ha sido una herramienta con características fundamentales para el análisis del procesamiento de datos, se puede decir que a medida que las demandas de procesamientos van aumentando las aplicaciones se van volviendo más complejas, por lo que el procesamiento lineal ha demostrado limitaciones cuando se trata de velocidad y capacidad al procesar los datos. La necesidad de realizar operaciones computacionales de manera más rápida y eficiente llevó al desarrollo del procesamiento paralelo.[1][2][3][4] Este nuevo procesamiento implica realizar numerosos cálculos al mismo tiempo, dividir una tarea en subtareas más pequeñas y asignarlas a diferentes unidades de procesamiento.

En este estudio se va a realizar una comparativa entre el procesamiento lineal y el procesamiento paralelo tomando mayor énfasis en el tiempo que tarda en definir el borde de una imagen al ser ejecutado en el software Matlab. Además, se evaluará el rendimiento en diferentes sistemas empleando benchmarking como técnica adicional para lograr tener una mejor perspectiva de comportamiento línea y paralelo en sistemas de procesamiento.

1.1 OBJETIVO GENERAL

El objetivo principal es analizar la disparidad en el tiempo de ejecución al aplicar distintas funciones matemáticas en los enfoques de procesamiento lineal y paralelo, utilizando benchmarking en distintos sistemas.

1.2 OBJETIVOS ESPECÍFICOS

1. Reducir los tiempos de ejecución mediante el procesamiento paralelo.
2. Incrementar la eficiencia de análisis en una imagen mediante la segmentación.
3. Comparar el comportamiento del procesamiento lineal con respecto al paralelo.

1.3 ALCANCE

El presente Proyecto de Integración Curricular satisface el procesamiento de imágenes mediante funciones básicas que nos permitan entender las tareas múltiples implementadas en Matlab; lo cual se desarrollará para su posterior análisis. Al ejecutar de forma continua el

número de veces que se va a realizar la simulación se deberá comparar la eficiencia entre una ejecución simple y una con paralelismo, esto se realizará una vez cargada la imagen mediante una función propia de Matlab y luego se realizará la segmentación de la imagen la cual estará divida en bloques y esto será en base a los algoritmos previamente revisados como por ejemplo los clasificadores en cascada [1], una vez realizada la simulación se procede a evaluar y realizar las pruebas de los distintos algoritmos; finalmente se observará mediante una gráfica en la cual se indicará el tiempo de ejecución y tamaño de la matriz misma que se relacionará con el número de bloques de intervalos de tiempo. En el proceso de desarrollo de la implementación del código del prototipo matemático para el benchmarking para procesamiento paralelo, se requiere una fase de pruebas.

1.4 MARCO TEÓRICO

1.4.1 PROCESAMIENTO DE IMÁGENES

El procesamiento de imágenes ha experimentado una transformación radical a lo largo de su historia, impulsada por un constante progreso en la tecnología de hardware, software y algoritmos. En sus primeras etapas, este procesamiento se llevaba a cabo de manera analógica, recurriendo a técnicas como la amplificación, el filtrado y la combinación de señales de imagen en tiempo real. Sin embargo, conforme han transcurrido los años, esta disciplina ha evolucionado de manera impresionante, adoptando enfoques cada vez más sofisticados y (amplia difusión).

El panorama inicial del procesamiento de imágenes destacaba por su enfoque analógico, que permitía manipulaciones en tiempo real de la información visual. Las técnicas empleadas se centraban en la corrección de problemas comunes, como el ruido, el desenfoque y la falta de contraste. En este contexto, surgieron filtros tanto lineales como no lineales, como el suavizado para eliminar imperfecciones, la nitidez para realzar detalles y el realce para mejorar la visualización [5].

El concepto de procesamiento de imágenes abarca una amplia gama de métodos y algoritmos destinados a modificar, analizar y mejorar imágenes digitales. En el corazón de este enfoque se encuentra la representación de una imagen como una matriz bidimensional de píxeles, donde cada píxel almacena información de escala de grises o color.

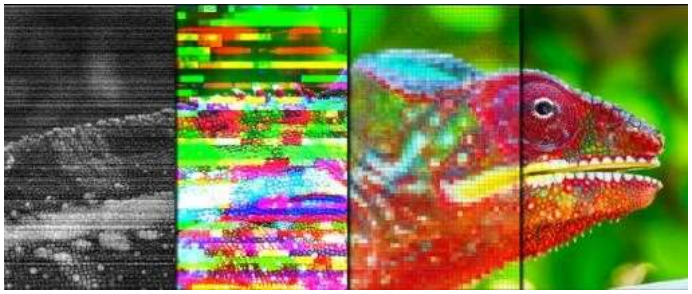

Figura 1. Imagen con varios procesamientos [6]

Estas imágenes digitales pueden provenir de diversas fuentes, como sensores, cámaras digitales o escáneres. El alcance del procesamiento de imágenes es vasto y abarca desde ajustes básicos para mejorar la calidad visual, como la modificación de brillo y contraste, hasta tareas altamente complejas como la segmentación de objetos, detección de bordes, eliminación de ruido, reconocimiento de patrones y análisis de contenido. La convergencia de múltiples factores, como el incremento en la potencia de cálculo, la evolución de algoritmos más sofisticados y la aplicación de técnicas de aprendizaje automático y visión por computadora, ha impulsado el crecimiento exponencial del procesamiento de imágenes.[7]

Este auge ha impulsado la incorporación del procesamiento de imágenes en una gran variedad de disciplinas y sectores. Desde la medicina, donde se utiliza para diagnóstico y análisis de imágenes médicas, hasta la robótica, donde se emplea para navegación y percepción de entornos, el procesamiento de imágenes ha encontrado aplicaciones en campos tan diversos como la seguridad, la automatización industrial, el entretenimiento y más. La capacidad de extraer información significativa de las imágenes y realizar tareas automatizadas ha permitido que el procesamiento de imágenes se posicione como una tecnología fundamental en la sociedad actual.

Figura 2. Imagen de Lena procesada

En resumen, la imagen original de Lena ha sido procesada mediante la aplicación del operador de Canny para la detección de bordes. Esto ha resultado en una imagen en blanco y negro donde los bordes y contornos están resaltados, permitiendo una mejor comprensión de la estructura y los detalles de la imagen. El procesamiento tiene aplicaciones en campos como la visión por computadora y la investigación en imágenes digitales.[8]

1.4.2 PROCESAMIENTO LINEAL O SECUENCIAL

La teoría matemática y los avances en la tecnología de imágenes son los pilares del procesamiento lineal de imágenes. Varios campos como las matemáticas, la física y la informática han contribuido al desarrollo de técnicas de procesamiento de imágenes a lo largo de la historia. El procesamiento lineal de imágenes comenzó cuando los avances en la tecnología de la computación permitieron el procesamiento digital de imágenes en los años 60 y 70. Los investigadores en ese momento descubrieron que las imágenes podían ser tratadas como matrices numéricas, lo que permitió realizar operaciones los procesadores podían realizar una tarea a la vez lo que quiere decir que las tareas se realizaban una después de otra cosa que la tarea no podía ser divididas sobre ellas.[9]

La introducción del campo de la transformada de Fourier fue un hito en el procesamiento lineal de imágenes. La transformada de Fourier es una herramienta matemática que nos permite descomponer una señal en sus componentes de frecuencia. En el campo de las imágenes, las propiedades espaciales de una imagen, como las texturas y los bordes, se examinan mediante la transformada de Fourier. Los investigadores pudieron aplicar filtros lineales a las imágenes para realizar operaciones como suavizar, realzar los bordes y detectar características mediante la transformada de Fourier.[10][11] Estos filtros se basan en convoluciones, que significa que la ventana se desplaza sobre la imagen y en cada posición se realiza las operaciones descritas por la ventana de manera secuencial.

En este estudio, se ha decidido enfocarse en el campo del procesamiento de imágenes debido a su potencial para llevar a cabo operaciones de manera segmentada o en bloques. La elección de esta área se fundamenta en la idea de que las imágenes pueden ser divididas en unidades más pequeñas y manejables, lo que facilita la aplicación de diversos algoritmos y técnicas de procesamiento. Al considerar el procesamiento en bloques, se busca aprovechar la capacidad de los sistemas de cómputo modernos para trabajar en paralelo y de manera eficiente en tareas repetitivas. Este enfoque no solo permite una mayor flexibilidad en la implementación de algoritmos específicos, sino que también puede contribuir significativamente a la optimización del tiempo de ejecución.[12][13] Las

imágenes al estar constituidas por matrices las cuales son procesadas utilizando mascaras que definen funciones especificas pueden ser segmentadas o divididas en bloques donde las operaciones descritas por las mascara solo afectan al segmento de imagen sobre el cual se esté ejecutando.

En comparación con los métodos de procesamiento lineal tradicionales, las técnicas de procesamiento no lineal han demostrado consistentemente un mejor rendimiento y resultados. Han superado las capacidades de los enfoques lineales en una variedad de aplicaciones y han establecido récords de precisión en desafíos de visión por computadora. Se ha producido un cambio de paradigma en el procesamiento de imágenes hacia enfoques paralelos como resultado de esta evidencia.[14]

1.4.3 PROCESAMIENTOS PARALELOS

La evolución del procesamiento paralelo ha sido un proceso continuo que ha experimentado importantes avances a lo largo del tiempo, los primeros procesadores vectoriales aparecieron a finales de la década de 1960 y principios de la de 1970. Estos procesadores se diseñaron para realizar operaciones en paralelo en grandes conjuntos de datos secuenciales, lo que permitía un rendimiento mucho mayor en aplicaciones científicas y de ingeniería.

Las supercomputadoras MPP, que consistían en una gran cantidad de procesadores que trabajaban juntos, se desarrollaron en la década de 1980. Estas máquinas se utilizaron para aplicaciones que requerían un alto nivel de paralelismo, como simulaciones complejas y procesamiento intensivo de datos, lo que permitió una mayor escalabilidad y capacidad de procesamiento [15][16][17]. La Ley de Moore continuó impulsando el aumento de la densidad de los transistores, lo que llevó a los fabricantes de chips a integrar múltiples núcleos de procesamiento en un solo chip. Esto llevó a los procesadores multicore, que tienen más de un núcleo y pueden realizar tareas en paralelo. Las arquitecturas SIMD (Single Instruction, Multiple Data) también fueron desarrolladas para permitir la ejecución simultánea de instrucciones idénticas en múltiples datos.

La computación en clúster, que utiliza múltiples computadoras conectadas, se convirtió en una práctica popular en la década de 1990. Esta técnica se utilizaba ampliamente en campos como la computación distribuida y la investigación científica porque permitía una mayor escalabilidad y flexibilidad al permitir agregar o quitar nodos de manera más fácil. Las unidades de procesamiento gráfico (GPU) pasaron de ser dispositivos especializados para gráficos a poderosas plataformas de procesamiento paralelo a partir de la década de 2000. Las GPU tienen cientos o miles de núcleos que pueden ejecutar tareas en paralelo,

lo que las hace ideales para aplicaciones como el aprendizaje automático y la simulación científica que requieren un alto rendimiento computacional.[18]

La computación heterogénea combina varios tipos de unidades de procesamiento, como CPUs y GPUs, para maximizar el procesamiento paralelo. Además, los aceleradores especializados, como los FPGAs (Field Programmable Gate Arrays), se han desarrollado para mejorar el rendimiento y la flexibilidad en aplicaciones particulares.

1.4.4 BENCHMARKING

El benchmarking, una herramienta esencial para evaluar y mejorar el rendimiento en diversos ámbitos, ha experimentado una constante evolución a lo largo del tiempo. Esta práctica, ahora ampliamente utilizada en diferentes campos como tecnología, gestión empresarial e investigación científica, desempeña un papel crucial en la identificación de mejores prácticas, comparación de rendimientos y toma de decisiones informadas para lograr una optimización continua en diversos contextos.

En sus primeras manifestaciones, el benchmarking se enfocaba en la comparación de procesos de producción y rendimiento industrial. Las empresas analizaban sus prácticas con el fin de identificar mejores enfoques y prácticas adoptadas por competidores o líderes en la industria[19]. Este enfoque permitía a las organizaciones optimizar sus procesos y mejorar su competitividad.

Con el avance tecnológico, el benchmarking se adaptó a nuevos campos, incluyendo la informática y la tecnología de la información. Se desarrollaron herramientas y metodologías para medir y comparar el rendimiento de sistemas informáticos, como CPUs y GPUs. Estos benchmarks tecnológicos permiten evaluar el rendimiento, la velocidad y la eficiencia de dispositivos y sistemas, facilitando decisiones informadas de compra y desarrollo.[20]

El uso de benchmarks se expandió al mundo de los software y aplicaciones, donde se utiliza para evaluar la velocidad de procesamiento, la eficiencia de algoritmos y la capacidad de respuesta. En áreas como la inteligencia artificial y el aprendizaje automático, los benchmarks ayudan a comparar y mejorar los modelos y algoritmos, permitiendo una evolución más rápida de la tecnología.

El benchmarking no se limita a la tecnología; ha encontrado su lugar en la gestión empresarial, donde se aplica para comparar prácticas financieras, operativas y de recursos humanos. Asimismo, en la investigación científica, los benchmarks se emplean para establecer puntos de referencia en campos como la medicina, la física y la biología, asegurando resultados sólidos y confiables.

El avance hacia la era digital ha llevado al benchmarking en línea y a la medición de métricas de rendimiento en plataformas web y redes sociales. Esto permite a las empresas evaluar su impacto en línea y compararse con competidores en términos de presencia digital y participación en línea.[21]

En la actualidad, el benchmarking es una práctica esencial en la optimización y mejora continua en casi todos los ámbitos. Las organizaciones confían en él para identificar oportunidades de mejora, tomar decisiones informadas y permanecer competitivas en un mundo en constante cambio. Ya sea para comparar el rendimiento de sistemas tecnológicos o evaluar prácticas empresariales, el benchmarking sigue siendo una herramienta vital en la búsqueda de la excelencia.[22]

2 METODOLOGÍA

Este capítulo mostrará la comparativa del código de prueba (Benchmark) en distintas plataformas con distintos procesadores y Arquitecturas mediante el análisis del procesamiento secuencial y paralelo.

Para esta comparación se realizará 4 pruebas donde se analizará los respectivos tiempos de procesamiento al obtener el borde de una imagen mediante un procesamiento secuencial y paralelo.

1. Prueba de obtención del tiempo de procesamiento para un procesador Intel Celeron
2. Prueba de obtención del tiempo de procesamiento para un procesador Intel Core i3
3. Prueba de obtención del tiempo de procesamiento para un procesador Intel Core i5
4. Prueba de obtención del tiempo de procesamiento para un procesador AMD Ryzen 7

Este análisis proporciona una visión general de cómo los diferentes procesadores se desempeñan en términos de tiempo de ejecución para las tareas dadas. Sin embargo, para tomar decisiones informadas, es importante tener en cuenta otros factores, como el tiempo y los recursos disponibles, antes de seleccionar un procesador específico para un caso de uso particular.

La siguiente tabla muestra los tipos de procesadores, así como la división de la imagen y los tiempos de procesamiento de acuerdo con el tipo secuencial o paralelo utilizado en las pruebas.

Tabla 1. Tabla de parámetros empleados en cada prueba

Procesador	N° Divisiones Imagen
Intel Celeron	2 divisiones
	4 divisiones
	8 divisiones
Intel Core i3	2 divisiones
	4 divisiones
	8 divisiones
Intel Core i5	2 divisiones
	4 divisiones
	8 divisiones
AMD Ryzen 7	2 divisiones
	4 divisiones
	8 divisiones

Para el análisis y comparación del tiempo de procesamiento se hará uso de la función propia de Matlab *tic toc,* la función tic registra el tiempo inicial, mientras que la función toc calcula el tiempo transcurrido utilizando el valor tomado. Esto se llevará a cabo para cada uno de los métodos para que tengan valores comparables.

Los equipos que se utilizaron para que se lleve a cabo las pruebas fueron una portátil que tenían las siguientes características:

Tabla 2. Procesadores utilizados

Tipo de Procesador	Memoria RAM
Intel(R) Celeron (R) CPU N3350	8GB
Intel(R) Core (TM) i3-4130	4GB
Intel(R) Core (TM) i5-7200U	8GB
AMD Ryzen 7 5700U	16GB

Es importante destacar que, aunque la función tic toc indica el tiempo de procesamiento de cada código, no incluye la demora en la visualización de las gráficas. Este tiempo fue superior al tiempo utilizado por la función en algunos métodos. El Capítulo 3 (Resultados) abordará estos detalles.

2.1 IMPLEMENTACION DEL PROCESAMIENTO LINEAL O SECUENCIAL SOBRE UNA IMAGEN

Para llevar a cabo la implementación del procesamiento secuencial, será fundamental utilizar los parámetros previamente establecidos en la Tabla 1. Además, se plantea realizar una división de dos imágenes, en la que una mostrará la imagen original y la otra, en escala de grises, se enfocará únicamente en los bordes. Este enfoque permitirá obtener un campo visual óptimo y detallado de los resultados.

El proceso se inicia con la carga de la imagen que fungirá como modelo. Para lograrlo, se empleará la función "imread". Esta instrucción selecciona la imagen desde una ubicación específica, teniendo en cuenta que es recomendable guardar la imagen en la misma carpeta que contiene el archivo de implementación. Antes de aplicar el procesamiento, se llevará a cabo la conversión de la imagen a escala de grises. Esto es esencial para el siguiente paso, donde se aplicará la función propia de Matlab denominada "edge". Esta función es crucial para la detección de bordes en la imagen. Vale la pena mencionar que "edge" utiliza el algoritmo de detección de bordes de Sobel, que consiste en convolucionar la imagen con dos máscaras o filtros conocidos como operadores de Sobel. Esta técnica es ampliamente utilizada en el procesamiento de imágenes para lograr una detección precisa de bordes. Es importante resaltar que uno de los parámetros clave de la función "edge" es la propia imagen convertida a escala de grises. Además, se requieren dos umbrales, que tienen un impacto significativo en la calidad de los resultados obtenidos. La elección adecuada de estos umbrales puede influir en la precisión de la detección de bordes y, por ende, en la calidad general de la imagen resultante.

En última instancia, para evaluar y registrar los tiempos de procesamiento, se ha implementado la combinación de funciones "tic" y "toc". Estas funciones proporcionan un mecanismo para medir el tiempo transcurrido desde el inicio de un proceso hasta su finalización. Esto resulta esencial para evaluar el rendimiento del procesamiento secuencial y determinar si es necesario implementar estrategias de optimización.

En resumen, la implementación del procesamiento secuencial implica una serie de pasos cruciales, desde la carga de la imagen modelo hasta la detección de bordes utilizando el

algoritmo de Sobel. El uso de funciones específicas de Matlab y la elección adecuada de parámetros son esenciales para lograr resultados precisos. La medición de los tiempos de procesamiento a través de las funciones "tic" y "toc" agrega un componente fundamental de evaluación y optimización en esta implementación. A continuación, se muestra el segmento de código utilizado para cargar y sacra el borde una imagen, este será utilizado para las respectivas pruebas, ajustando los parámetros según indica la Tabla 1.

```
clc,
clear all,

% Leer la imagen
tic;
imagen = imread('imagen.jpg');
tiempo_carga = toc;

% Convertir la imagen a escala de grises
tic;
imagen_grises = rgb2gray(imagen);
tiempo_conversion = toc;

% Aplicar el operador de detección de bordes de Canny
tic;
umbral_min = 0.1; % Umbral mínimo
umbral_max = 0.3; % Umbral máximo
bordes = edge(imagen_grises, 'Canny', [umbral_min, umbral_max]);
tiempo_procesamiento = toc;

% Mostrar la imagen original y los bordes encontrados
subplot(1, 2, 1);
imshow(imagen);
title('Imagen original');

subplot(1, 2, 2);
imshow(bordes);
title('Bordes de la imagen');

% Mostrar los tiempos de ejecución
disp(['Tiempo de carga: ', num2str(tiempo_carga), ' segundos']);
disp(['Tiempo de conversión a escala de grises: ',
num2str(tiempo_conversion), ' segundos']);
disp(['Tiempo de procesamiento: ', num2str(tiempo procesamiento), '
segundos']);
```

La Figura 3. en el lado izquierdo se muestra la imagen seleccionada para la prueba número 1 donde en el lado izquierdo al utilizar la función Edge se logró obtener los bordes visibles en una escala de grises.

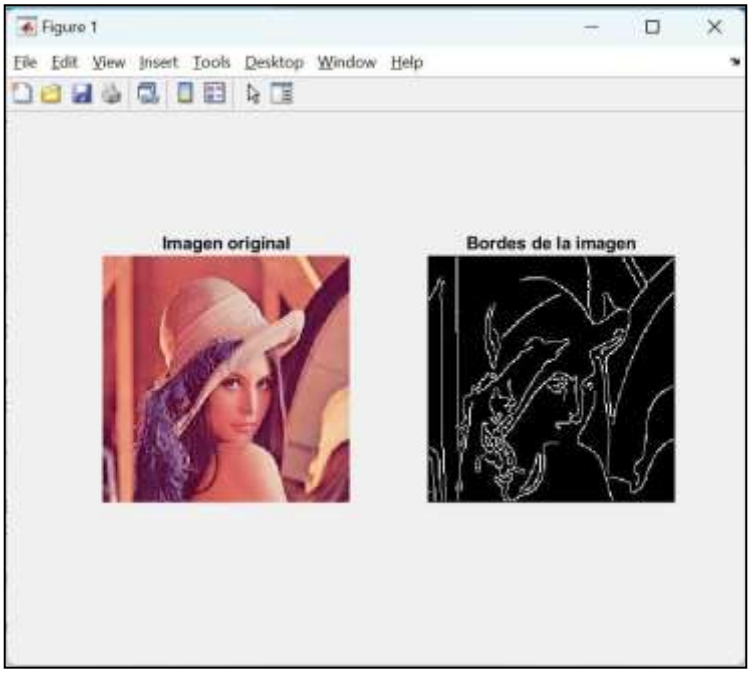

Figura 3. Imagen procesada

La Figura 4. muestra el command window de Matlab posterior a la ejecución del código de la prueba 1, donde se tiene el primer tiempo al obtener la imagen desde el directorio, el segundo tiempo muestra la duración al transformar la imagen de escala normal a escala de grises y finalmente el tiempo de procesamiento es el tiempo total de la ejecución del código.

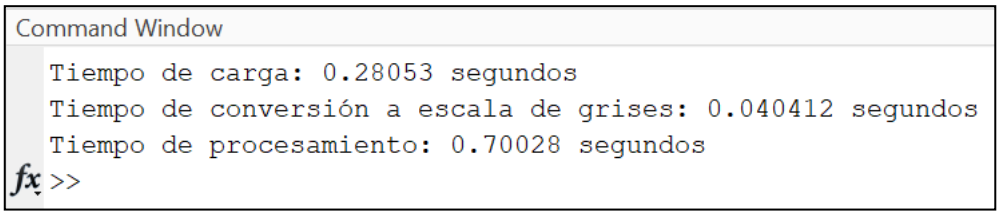

Figura 4. Tiempo de procesamiento ideal para la prueba 1.

2.2 IMPLEMENTACION DEL PARALELISMO

"Parallel Computing Toolbox" representa una herramienta crucial dentro del entorno de Matlab, diseñada estratégicamente para capitalizar la capacidad de sistemas equipados con múltiples núcleos de CPU o incluso nodos interconectados en un clúster informático. Su enfoque radica en optimizar el rendimiento de cálculos y procesamientos que demandan

una alta capacidad de procesamiento. Esta caja de herramientas se erige como una solución que propulsa la eficiencia y la velocidad en operaciones intensivas de datos.

La gran virtud de esta herramienta recae en su capacidad para habilitar la paralelización de algoritmos y aplicaciones. En esencia, permite a los usuarios descomponer tareas computacionales en segmentos más pequeños y ejecutarlos simultáneamente en múltiples núcleos de CPU o inclusive en un conjunto de computadoras interconectadas. Esta estrategia de paralelización da como resultado una mejora substancial en la velocidad de procesamiento, ya que múltiples operaciones pueden llevarse a cabo al mismo tiempo, en lugar de secuencialmente.

A continuación, se muestra algunos conceptos y funciones clave relacionadas con el paralelismo en MATLAB:[23]

a. **Pools paralelos:** Un pool paralelo es un conjunto de trabajadores que tienen la capacidad de llevar a cabo tareas de manera simultánea y coordinada. En el contexto de MATLAB, esta estructura es especialmente útil para aprovechar al máximo el potencial de procesamiento de sistemas modernos, permitiendo una distribución eficiente de las tareas para acelerar la ejecución de procesos intensivos en cómputo.

   La creación de un pool paralelo se convierte en una estrategia esencial para implementar el procesamiento paralelo en MATLAB. Al habilitar el paralelismo, MATLAB inicia un grupo de trabajadores que pueden operar de manera independiente pero coordinada para ejecutar las tareas asignadas. Esto representa una manera altamente eficaz de reducir el tiempo necesario para completar tareas complejas al dividirlas en unidades manejables y aprovechar el poder de múltiples núcleos de CPU.

   La función central que facilita la creación de un pool paralelo en MATLAB es la función "parpool". A través de esta función, se puede establecer la cantidad de trabajadores que se desean en el pool. Por ejemplo, la llamada "parpool('local', N)" generará un pool paralelo con N trabajadores en la máquina local. Cada uno de estos trabajadores puede ejecutar tareas de manera independiente, lo que permite una ejecución simultánea y una mayor eficiencia en el procesamiento.

b. **División de tareas:** La división de tareas en partes más pequeñas e independientes, con el objetivo de ejecutarlas en paralelo, es una estrategia fundamental en el ámbito del procesamiento paralelo. Esta técnica se utiliza para aprovechar al máximo los recursos de hardware disponibles y reducir significativamente el tiempo de procesamiento.

   La función clave que facilita esta división y ejecución paralela en MATLAB es la función "parfor". Similar en su sintaxis a un bucle "for" convencional, el "parfor" tiene la ventaja crucial de permitir que cada iteración del bucle se ejecute en paralelo, distribuyendo automáticamente las tareas a diferentes núcleos de CPU o incluso a máquinas en un clúster de computadoras. Por ejemplo, si ese está procesando imágenes y aplicando un filtro a cada píxel, el "parfor" dividiría la imagen en segmentos más pequeños y asignaría estos segmentos a diferentes núcleos. Cada núcleo aplicaría el filtro a su segmento de manera independiente, lo que resulta en un procesamiento general más rápido.

c. **Comunicación entre trabajadores:** Dentro del entorno de MATLAB, se establecen mecanismos sólidos y eficientes para la comunicación y sincronización entre los trabajadores que conforman un pool paralelo. Estos mecanismos son esenciales para lograr una ejecución fluida y coordinada de tareas paralelas, permitiendo el intercambio de información y la colaboración entre los procesos en funcionamiento.

   En el proceso de ejecución paralela, es común que surja la necesidad de compartir datos entre diferentes trabajadores para realizar cálculos o tomar decisiones basadas en información previamente procesada. MATLAB ofrece una serie de recursos para abordar esta situación. Una de las formas más comunes de lograr la comunicación entre los trabajadores es a través del uso de variables compartidas. Estas variables pueden ser accedidas y modificadas por diferentes trabajadores, lo que facilita el intercambio de datos en tiempo real.

d. **Control de trabajadores:** El control de los trabajadores en un entorno de procesamiento paralelo es un aspecto crítico para lograr una ejecución eficiente y coordinada de tareas. En MATLAB, se proporcionan funciones y herramientas específicas para gestionar la asignación de trabajadores y el flujo de tareas en un pool paralelo, permitiendo un control preciso sobre cómo se distribuyen las operaciones y cómo interactúan los trabajadores entre sí.

Una de las funciones clave para controlar los trabajadores es "parfor". Esta función permite establecer cómo se deben asignar las iteraciones del bucle paralelo a los trabajadores individuales. Con "parfor", es posible especificar de manera explícita qué trabajadores se encargan de cuáles iteraciones del bucle, lo que es útil cuando ciertas iteraciones requieren más recursos o tiempo de cálculo que otras. Esto permite un nivel avanzado de optimización y distribución de la carga de trabajo.

El control preciso de los trabajadores y las tareas es crucial para evitar cuellos de botella y maximizar la utilización de recursos en un entorno paralelo. Sin embargo, es importante mencionar que un control excesivo podría llevar a una administración ineficiente de los trabajadores y, en algunos casos, ralentizar el proceso debido a la sobrecarga administrativa.

En esta instancia, se emplearon la función "parpool" y el bucle "parfor" como parte de la metodología de procesamiento. Esta combinación puede visualizarse y representarse de la siguiente manera:

La función "*parpool*" y el bucle paralelo "*parfor"* son dos conceptos clave en la Parallel Computing Toolbox de MATLAB que utilizan para realizar cálculos en paralelo y aprovechar múltiples núcleos de CPU para mejorar el rendimiento de ciertos tipos de tareas.

La función "parpool" se utiliza para crear un pool de trabajadores, que consiste en múltiples núcleos de CPU disponibles en la máquina local o en un clúster de computadoras. Los trabajadores son instancias independientes del entorno de MATLAB que pueden ejecutar cálculos de manera simultánea e independiente entre sí, la sintaxis es la siguiente:

```
1 | pool = parpool(numWorkers);
```

Donde '*numWorkers'* es el número de trabajadores que se desea crear en el pool, si no se especifica este valor, MATLAB utilizará automáticamente el número máximo de núcleos de CPU disponibles en la máquina.

El bucle paralelo "*parfor*" es una variante del bucle "*for"* de MATLAB que se utiliza para dividir automáticamente las iteraciones entre los trabajadores en el pool de trabajadores creado con parpool. Esto permite que las iteraciones se realicen en paralelo en múltiples núcleos de CPU, lo que acelera el procesamiento para cálculos intensivos.

En el código implementado se utilizó este bucle, donde un parámetro principal es "*numPartes*" ya que aquí se especificará el número de partes en las que se dividirá la

imagen para los distintos procesadores que se utiliza, en este bucle también se tiene en cuenta que las iteraciones se realizarán en paralelo en diferentes trabajadores, lo que acelera el procesamiento.

Con el siguiente parámetro "*filaInicio = (i - 1) * alturaParte + 1*" se calcula el índice de la primera fila que corresponde a la parte actual, "i" representa el número de parte actual del bucle, y "*alturaParte*" es una variable que contiene la altura (número de filas) de cada parte en la que se ha dividido la tarea. El cálculo "*(i - 1) * alturaParte + 1*" determina el índice de la primera fila en cada parte; mientras que el parametro "*filaFin = filaInicio + alturaParte – 1*", calcula el índice de la última fila que corresponde a la parte actual, se suma "*alturaParte – 1*" al índice de inicio para obtener el índice de la última fila en cada parte.

```
% Habilitar el paralelismo
parpool();

% Aplicar la detección de bordes en paralelo
parfor i = 1:numPartes
    % Calcular los índices de fila correspondientes a
la parte actual
    filaInicio = (i - 1) * alturaParte + 1;
    filaFin = filaInicio + alturaParte - 1;
    if i == numPartes
        filaFin = h;
    end
```

Es importante tener en cuenta que la función parpool y el bucle paralelo parfor deben usarse con precaución y solo para cálculos que sean lo suficientemente grandes y complejos como para justificar la sobrecarga de la paralelización. También es importante considerar el tamaño de los datos y la carga computacional para obtener el máximo beneficio del paralelismo. Si los cálculos son demasiado pequeños o la sobrecarga de la paralelización es alta, es posible que no se obtenga una mejora significativa en el rendimiento.

Es importante mencionar que al ejecutar el código queda iniciada una sesión activa de parpool por lo que se va a utilizar una función que nos permite verificar si existe una sesión activa de "*parpool*" y en caso de que sea afirmativo, cierra esa sesión para tener liberados los recursos asociados, el código utilizado es el siguiente:

```
if ~isempty(gcp('nocreate'))
    % Si existe, cerrar la sesión de parpool
    delete(gcp);
end
```

if ~isempty(gcp('nocreate'))	verifica si existe una sesión de parpool activa utilizando la función gcp('nocreate').
gcp('nocreate')	intenta obtener la sesión actual del pool de trabajadores, pero no crea una nueva sesión si no existe ninguna
delete(gcp)	Utiliza el comando delete para cerrar la sesión y liberar los recursos asociados. Al cerrar la sesión, los trabajadores del pool se liberan y quedan disponibles para ser utilizados por otras tareas o procesos.

En resumen, el código comprueba si existe una sesión activa de parpool y, en caso afirmativo, la cierra correctamente utilizando el comando delete(gcp). Esto es útil para asegurarse de que los recursos del pool de trabajadores se liberen adecuadamente cuando ya no sean necesarios, evitando problemas de consumo innecesario de memoria o núcleos de CPU en futuras ejecuciones.

2.3 IMPLEMENTACION DEL PROCESAMIENTO PARALELO

Una vez definido las funciones que nos permiten realizar el paralelismo en Matlab se realiza el código general donde muestra unificado cada proceso realizado. A continuación, se presenta el código empleado:

```
% clc,
% clear all,
% Verificar si existe una sesión de parpool activa
```

```
4  if ~isempty(gcp('nocreate'))
5      % Si existe, cerrar la sesión de parpool
6      delete(gcp);
7  end
8
9  % Leer la imagen
10 imagen = imread('imagen.jpg');
11
12 % Convertir la imagen a escala de grises si es necesario
13 if size(imagen, 3) == 3
14     imagen_grises = rgb2gray(imagen);
15 else
16     imagen_grises = imagen;
17 end
18
19 % Dividir la imagen en partes iguales para el procesamiento
20 paralelo
21 numPartes = 2; % Número de partes en las que se dividirá la
22 imagen
23 [h, w] = size(imagen_grises);
24 alturaParte = floor(h / numPartes);
25
26 % Inicializar celdas para almacenar los bordes detectados en cada
27 parte
28 bordesPartes = cell(numPartes, 1);
29
30 % Habilitar el paralelismo
31 parpool();
32
33 % Aplicar la detección de bordes en paralelo
34 parfor i = 1:numPartes
35
36     % Calcular los índices de fila correspondientes a la parte
37 actual
38     filaInicio = (i - 1) * alturaParte + 1;
39     filaFin = filaInicio + alturaParte - 1;
40     if i == numPartes
41         filaFin = h;
42     end
43
```

```
    % Obtener la parte de la imagen correspondiente
    parteImagen = imagen_grises(filaInicio:filaFin, :);

    % Aplicar la detección de bordes en la parte actual
    tic;
    bordesPartes{i} = edge(parteImagen); %función de detección de
bordes
    tiempo ejecucion= toc;
    disp(['Tiempo de ejecución: ', num2str(tiempo_ejecucion), '
segundos']);

end

% Combinar los bordes de todas las partes
bordes = false(h, w);
for i = 1:numPartes
    filaInicio = (i - 1) * alturaParte + 1;
    filaFin = filaInicio + alturaParte - 1;
    if i == numPartes
        filaFin = h;
    end
    bordes(filaInicio:filaFin, :) = bordesPartes{i};

end


% Mostrar la imagen original y los bordes encontrados
subplot(1, 2, 1);
imshow(imagen);
title('Imagen original');

subplot(1, 2, 2);
imshow(bordes);
title('Bordes de la imagen');

% Dibujar una cuadrícula sobre la imagen
hold on;

for i = 1:numPartes-1
    fila = i * alturaParte + 0.5;
```

```
84      line([0.5, w+0.5], [fila, fila], 'Color', 'red');
85  end
86
87  for j = 1:w-1
88      columna = j * alturaParte + 0.5;
89      line([columna, columna], [0.5, h+0.5], 'Color', 'red');
90  end
91  hold off;
92
93  % Cerrar el parpool
    delete(gcp);
```

El fragmento de código presentado ilustra de manera efectiva cómo se puede implementar el procesamiento paralelo en MATLAB para optimizar la detección de bordes en una imagen. Esta técnica se basa en la división de la tarea global en segmentos más manejables, los cuales son procesados de manera simultánea en varios núcleos de CPU disponibles en la máquina. La incorporación de la "Parallel Computing Toolbox" de MATLAB resulta fundamental para simplificar y potenciar el aprovechamiento del paralelismo, lo que a su vez contribuye a una considerable mejora en el rendimiento de tareas que demandan un alto poder de cómputo.

La representación gráfica proporcionada en la Figura 5 brinda una visión clara de la imagen original y, de manera simultánea, presenta la imagen que ha sido sometida a un proceso de extracción de bordes en escala de grises. La elección de escala de grises es significativa, ya que contribuye a un análisis visual más efectivo de los resultados. Asimismo, se puede notar la estrategia de dividir la imagen en múltiples partes, lo cual se refleja en la Figura 5. Esta división es una etapa crucial en la implementación de procesamiento paralelo, ya que permite distribuir la carga de trabajo de manera uniforme en los recursos disponibles.

Es importante resaltar la Figura 6, que ilustra el tiempo de procesamiento requerido para obtener el borde de la imagen al dividirla en las partes recomendadas. Esta representación visual del tiempo de ejecución ofrece una perspectiva tangible de la eficiencia y la velocidad ganadas mediante la implementación de procesamiento paralelo. Cabe mencionar que este código actuará como una herramienta clave en la fase de pruebas, permitiendo realizar evaluaciones detalladas de rendimiento y eficacia en diferentes escenarios.

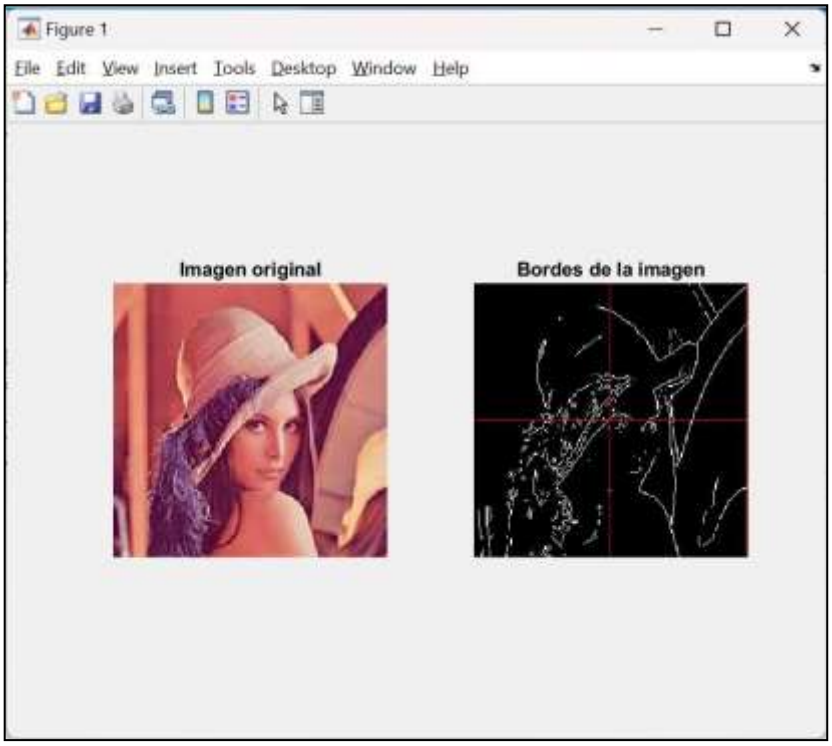

Figura 5. Imagen procesada con paralelismo

```
>> ImagParallel
Starting parallel pool (parpool) using the 'local' profile ...
Connected to the parallel pool (number of workers: 8).
Tiempo de ejecución: 0.04352 segundos
Tiempo de ejecución: 0.045601 segundos
Parallel pool using the 'local' profile is shutting down.
```

Figura 6. Tiempos calculados con procesamiento paralelo

2.4 IMPLEMENTACION DEL PROCESAMIENTO SECUENCIAL Y PARALELO MEDIANTE BLOQUES

La implementación del procesamiento secuencial y paralelo a través de bloques es abordada en la sección 2.4. En este apartado, se detalla cómo se aplican y comparan ambos enfoques en términos de segmentación de la imagen en bloques individuales, con el propósito de evaluar su eficiencia y rendimiento. A continuación, se muestra el código empleado:

```
1  clear all;
2  clc;
3  % Cargar la imagen
4  imagen = imread('imagen.jpg'); % Reemplaza con la ruta de tu imagen
```

```
imshow(imagen);
title('Imagen Original');
% Convertir la imagen a escala de grises
tic;
imagen_grises = rgb2gray(imagen);
tiempo_conversion = toc;
% Aplicar el operador de detección de bordes de Canny
tic;
umbral_min = 0.1; % Umbral mínimo
umbral_max = 0.3; % Umbral máximo
bordes = edge(imagen_grises, 'Canny', [umbral_min, umbral_max]);
tiempo_procesamiento = toc;
% Mostrar la imagen original y los bordes encontrados
subplot(1, 2, 1);
imshow(imagen);
title('Imagen original');

subplot(1, 2, 2);
imshow(bordes);
title('Bordes de la imagen');
% Mostrar los tiempos de ejecución
disp('IMAGEN COMPLETA')
disp(['Tiempo de procesamiento: ', num2str(tiempo_procesamiento), '
segundos']);
%%Imagen dividida y secuencial%%
% Dividir la imagen en 4 partes
[m, n, ~] = size(imagen);
parte1 = imagen(1:fix(m/2), 1:fix(n/2), :);
parte2 = imagen(1:fix(m/2), fix(n/2)+1:end, :);
parte3 = imagen(fix(m/2)+1:end, 1:fix(n/2), :);
parte4 = imagen(fix(m/2)+1:end, fix(n/2)+1:end, :);
% Convertir a escala de grises
gris1 = rgb2gray(parte1);
gris2 = rgb2gray(parte2);
gris3 = rgb2gray(parte3);
gris4 = rgb2gray(parte4);
% Aplicar el operador edge a cada parte
tic;
borde1 = edge(gris1, 'canny');
borde2 = edge(gris2, 'canny');
```

```
borde3 = edge(gris3, 'canny');
borde4 = edge(gris4, 'canny');
tiempo = toc;
disp('IMAGEN DIVIDA EN 4 PARTES Y SECUENCIAL');
fprintf('Tiempo de procesamiento: %.6f segundos\n', tiempo);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%
%imagen dividia y paralelismo

% Verificar si existe una sesión de parpool activa
if ~isempty(gcp('nocreate'))
    % Si existe, cerrar la sesión de parpool
    delete(gcp);
end
suma_tiempos_promedio_total = 0;
% Convertir la imagen a escala de grises si es necesario
if size(imagen, 3) == 3
    imagen = rgb2gray(imagen);
end
% Obtener las dimensiones de la imagen
[filas, columnas] = size(imagen);
% Dividir la imagen en 4 partes
filas_por_bloque = floor(filas / 2);
columnas_por_bloque = floor(columnas / 2);
% Valor constante a sumar a cada tiempo
%valor_constante = 0.005; % Cambia este valor según tus necesidades
parpool();
% Bucle para recorrer cada bloque de la imagen
parfor i = 1:2
    for j = 1:2
        % Obtener los límites del bloque actual
        inicio_fila = (i - 1) * filas_por_bloque + 1;
        fin_fila = min(i * filas_por_bloque, filas);
        inicio_columna = (j - 1) * columnas_por_bloque + 1;
        fin_columna = min(j * columnas_por_bloque, columnas);

        % Inicializar una variable para sumar los tiempos de cada
píxel en el bloque
        suma_tiempos_bloque = 0;
        % Bucle para recorrer cada píxel del bloque
```

```
85
86          for k = inicio_fila:fin_fila
87
88              for l = inicio_columna:fin_columna
89                  tic; % Comenzar a medir el tiempo
90                  % Procesar el píxel utilizando la función canny
91                  borde_pixel = edge(imagen(k, l), 'canny'); % Cambia
92  aquí el tipo de operador de borde si lo deseas
93
94                  %tiempo = toc + valor_constante; % Sumar el valor
95  constante al tiempo medido
96                  tiempo = toc;
97                  suma_tiempos_bloque = suma_tiempos_bloque + tiempo;
98  % Sumar el tiempo al total
99              end
100
101         end
102         % Calcular el tiempo promedio para el bloque actual
103         cantidad_pixeles_bloque = (fin_fila - inicio_fila + 1) *
104 (fin_columna - inicio_columna + 1);
105         tiempo_promedio_bloque = suma_tiempos_bloque /
106 cantidad_pixeles_bloque;
107
108         % Mostrar el tiempo promedio para el bloque actual
109         disp(['Tiempo de procesamiento para el bloque ',
110 num2str(i), 'x', num2str(j), ': ',
111 num2str(tiempo_promedio_bloque)]);
112         % Agregar el tiempo promedio del bloque actual a la suma
113 total
114         suma_tiempos_promedio_total = suma_tiempos_promedio_total +
115 tiempo_promedio_bloque;
116         %timeit(borde_pixel)
117     end
118 end
119 % Mostrar los bordes
120 figure;
121 subplot(2, 2, 1), imshow(borde1), title('Borde Parte 1');
122 subplot(2, 2, 2), imshow(borde2), title('Borde Parte 2');
123 subplot(2, 2, 3), imshow(borde3), title('Borde Parte 3');
124 subplot(2, 2, 4), imshow(borde4), title('Borde Parte 4');
```

```
125
126 % Cerrar el parpool
127 delete(gcp);
128
129 % Calcular el tiempo
130 disp('IMAGEN DIVIDIDA EN 4 PARTES Y PARALELISMO');
131 % Mostrar la suma total de los tiempos promedio de bloque en el
132 command window
133 disp(['Tiempo de proesamiento: ', num2str(tiempofinal)]);
134 fprintf('Tiempo de procesamiento: %.6f segundos\n', tiempofinal);
```

El proceso se inicia con la conversión de la imagen original a escala de grises, seguido por la aplicación del operador de detección de bordes de Canny. Se presenta una visualización comparativa de la imagen original y la resultante con bordes resaltados. Además, se aborda la subdivisión de la imagen en cuatro partes para ambos enfoques, primero secuencialmente y luego con paralelismo. Este último enfoque se beneficia de la distribución de carga en múltiples núcleos de procesador, agilizando el análisis. Los resultados cuantitativos de tiempos de procesamiento se presentan, destacando las ventajas de la implementación paralela en términos de eficiencia.

De manera complementaria, en la Figura 7 se muestra la imagen dividida en bloques. Sin embargo, es relevante destacar que este proceso se llevó a cabo en dos modalidades distintas: la imagen completa y su correspondiente versión dividida en bloques. La aplicación secuencial del algoritmo de detección de bordes a la imagen fragmentada permitió el análisis independiente de cada bloque, enfatizando así la variación en la detección de bordes en diferentes regiones de la imagen. Esta contrastante visualización pone de manifiesto cómo el enfoque de segmentación en bloques posibilita una comprensión más precisa de la influencia de los bordes en distintas secciones de la imagen, contribuyendo a una percepción más detallada de la estructura y características presentes.

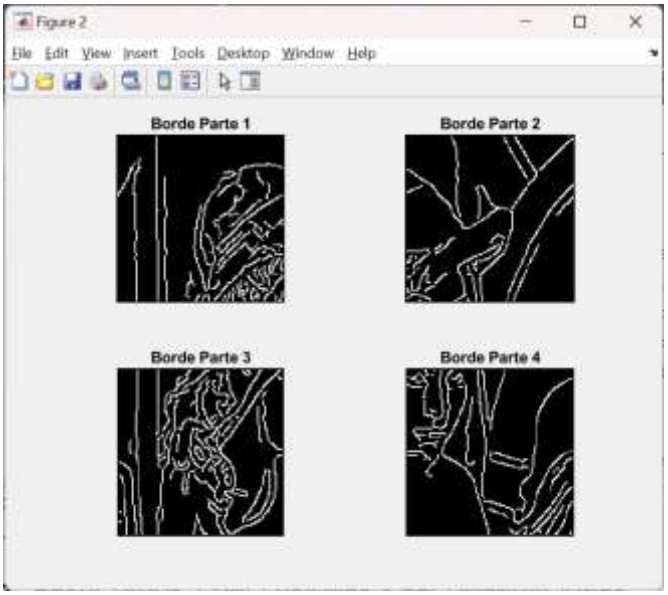

Figura 7. Imagen dividida en bloques

3 RESULTADOS, CONCLUSIONES Y RECOMENDACIONES

Este capítulo está destinado a la discusión de los resultados obtenidos con cada una de las pruebas tratadas, enfocándose en los tiempos de procesamiento que se obtiene al extraer el borde de la imagen y a la vez dividiéndola en diferentes partes.

3.1 RESULTADOS

3.1.1 COMPARACIÓN DE PROCESAMIENTOS – PRUEBA1- Intel Celeron

Al emplear este procesador se obtuvo los siguientes resultados:

- **2 DIVISIONES**

Figura 8. Análisis procesamiento paralelo y lineal con 2 divisiones

- **4 DIVISIONES**

Figura 9. Análisis procesamiento paralelo y lineal con 2 divisiones

- **8 DIVISIONES**

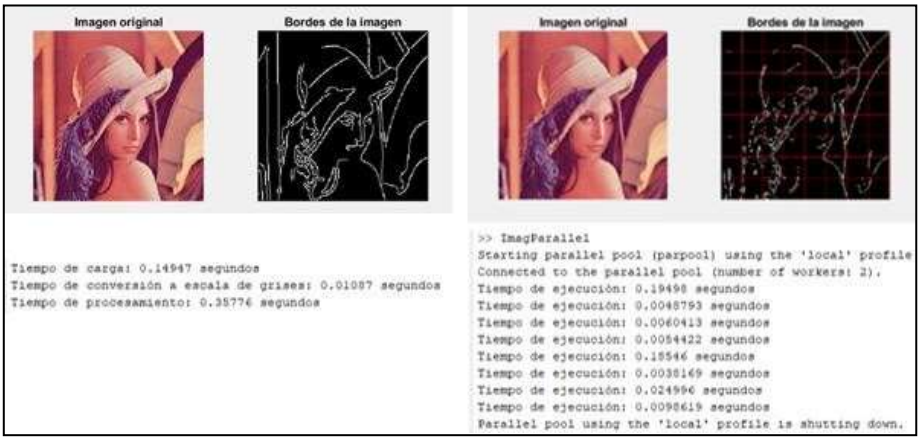

Figura 10. Análisis procesamiento paralelo y lineal con 2 divisiones

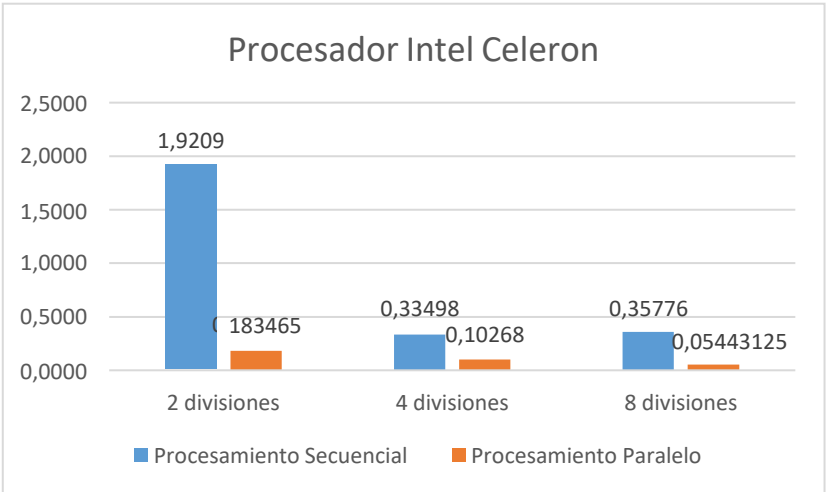

Figura 11. Histograma de análisis procesamiento paralelo y lineal con 2 divisiones

La figura de barras es esencial para exponer los resultados de la prueba 1, realizada con un procesador Intel Celeron. Esta representación gráfica resalta la relación entre los tiempos de ejecución en procesamiento paralelo y secuencial. La eficacia del procesamiento paralelo en términos de tiempo promedio se evidencia claramente, permitiendo una comprensión intuitiva, incluso para quienes no están familiarizados con aspectos técnicos. La figura destaca la influencia del procesador Intel Celeron en esta comparación y muestra de manera impactante la optimización conseguida mediante el enfoque paralelo, resaltando su valor en la mejora del rendimiento.

3.1.2 COMPARACIÓN DE PROCESAMIENTOS – PRUEBA 2 - Intel Core i3

- **2 DIVISIONES**

Figura 12. Análisis procesamiento paralelo y lineal con 2 divisiones

- **4 DIVISIONES**

Figura 13. Análisis procesamiento paralelo y lineal con 4 divisiones

- **8 DIVISIONES**

Figura 14. Análisis procesamiento paralelo y lineal con 8 divisiones

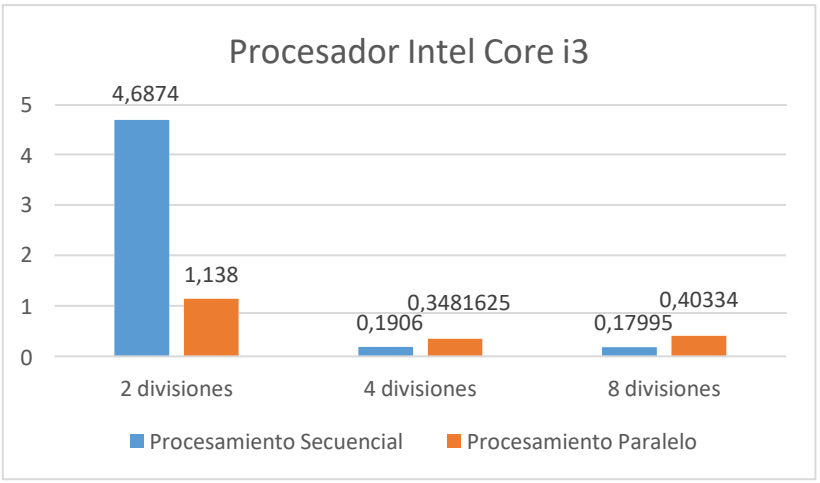

Figura 15. Histograma de análisis procesamiento paralelo y lineal

En la Figura 15, se presenta una situación análoga. No obstante, cabe destacar que en la barra de 8 divisiones se produce una variación sustancial. En este escenario, es esencial considerar la influencia de la eficiencia de la memoria RAM, ya que, en ocasiones, se presume que una porción del núcleo se asigna a otras tareas, lo que resulta en una disminución de su compromiso total en la ejecución de nuestro código. La ausencia de un control absoluto sobre esta dinámica puede impactar los resultados de tiempo de ejecución y resalta la complejidad subyacente en la optimización de los procesos paralelos en sistemas dinámicos.

3.1.3 COMPARACIÓN DE PROCESAMIENTOS – PRUEBA 3 - Intel Core i5

- **2 DIVISIONES**

Figura 16. Análisis procesamiento paralelo y lineal con 2 divisiones

- **4 DIVISIONES**

Figura 17. Análisis procesamiento paralelo y lineal con 4 divisiones

- **8 DIVISIONES**

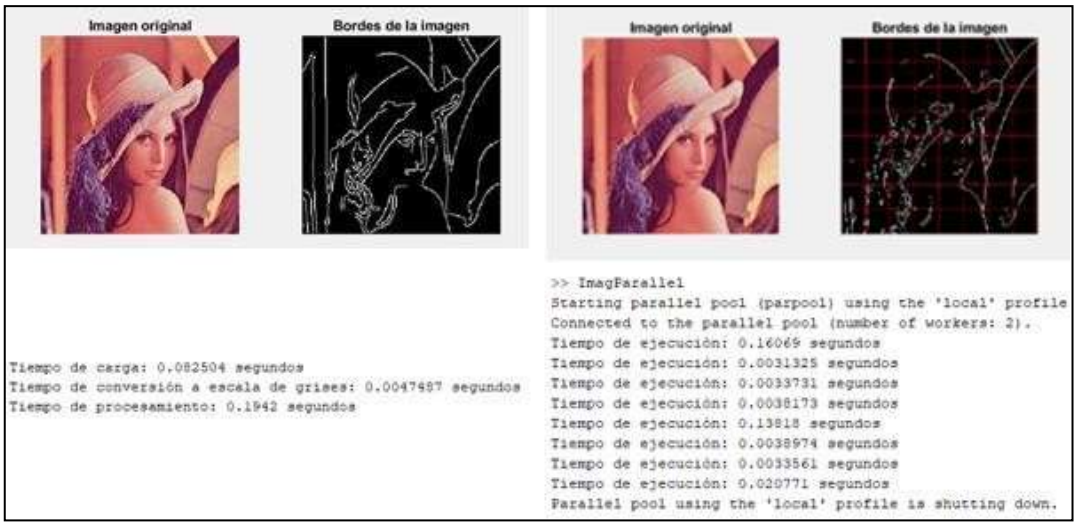

Figura 18. Análisis procesamiento paralelo y lineal con 8 divisiones

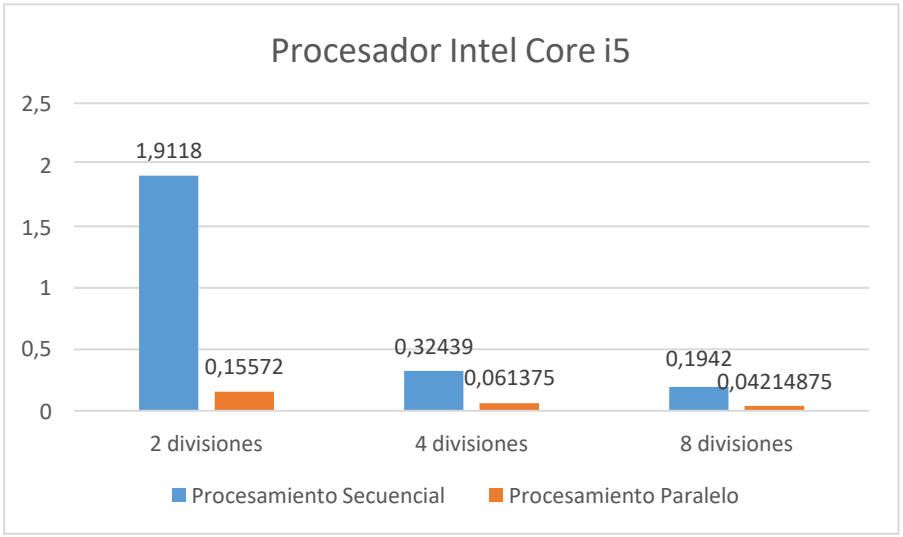

Figura 19. Histograma de análisis procesamiento paralelo y lineal

En la Figura 19, se revela una mejora significativa en el rendimiento del procesador empleado en comparación con los casos previos. Evidentemente, se distingue un rendimiento excepcional al calcular los tiempos mediante el uso del paralelismo. Esta observación resalta la influencia directa del hardware en los resultados de tiempo de ejecución y subraya la importancia de seleccionar componentes óptimos para la implementación de procesamiento paralelo. La notabilidad de los resultados también sugiere una mayor capacidad de cómputo y un mejor aprovechamiento de los recursos disponibles en este entorno particular.

3.1.4 COMPARACIÓN DE PROCESAMIENTOS – PRUEBA 4 - AMD Ryzen

- **2 DIVISIONES**

Figura 20. Histograma de análisis procesamiento paralelo y lineal con 2 divisiones

- **4 DIVISIONES**

Figura 21. Histograma de análisis procesamiento paralelo y lineal con 4 divisiones

- **8 DIVISIONES**

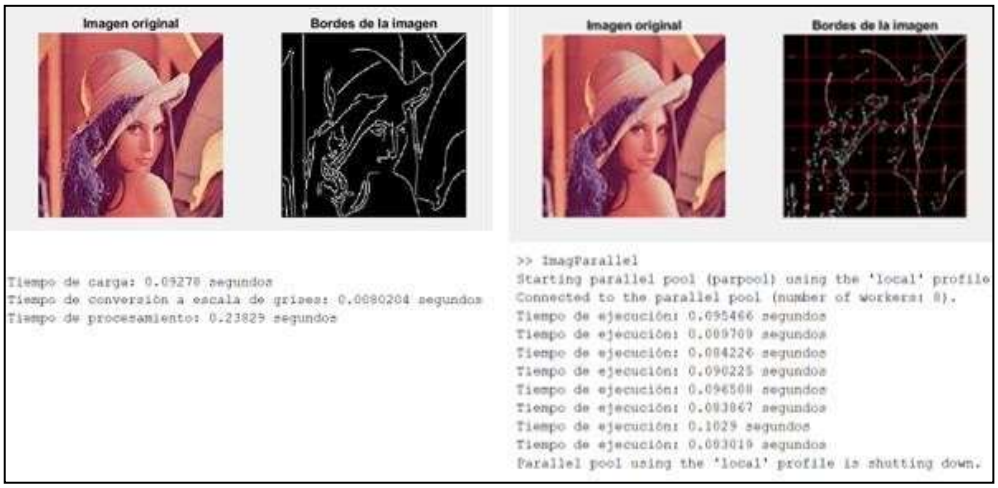

Figura 22. Análisis procesamiento paralelo y lineal con 8 divisiones

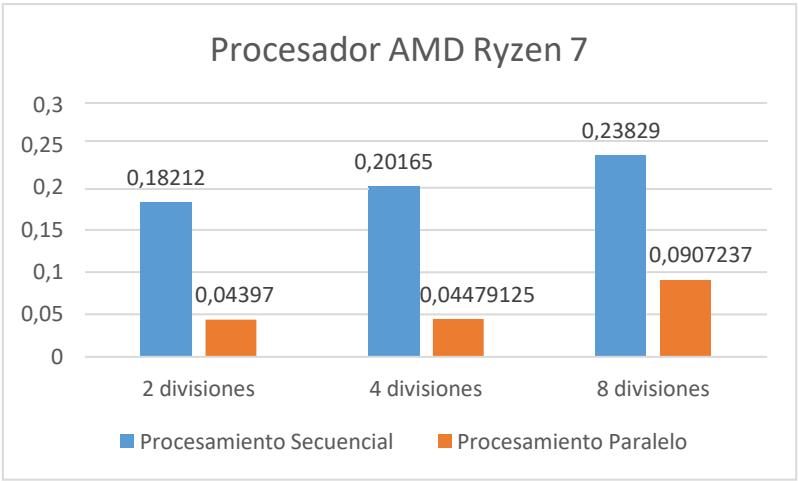

Figura 23. Histograma de análisis procesamiento paralelo y lineal

En la etapa conclusiva, se revelan cambios sumamente marcados que destacan el impacto del hardware en el procesamiento paralelo. En este escenario particular, se optó por una configuración que cuenta con un procesador de alto rendimiento y una memoria RAM de calidad. Esta elección estratégica tiene un efecto determinante en el desempeño del procesamiento paralelo, ya que se dispone de múltiples núcleos que pueden ser empleados para dividir y abordar eficazmente las tareas. La presencia de un procesador potente, respaldado por una memoria RAM adecuada, contribuye a la ejecución eficiente de las operaciones paralelas. Esta observación resalta la importancia de considerar las características del hardware al implementar enfoques de procesamiento paralelo, ya que un sistema robusto puede potenciar la capacidad de cómputo y ofrecer mejoras significativas en los tiempos de ejecución.

3.1.5 COMPARACIÓN DE PROCESAMIENTO CON BLOQUES

Durante esta etapa de experimentación, se ha decidido repetir los procedimientos de procesamiento, pero con un enfoque más detallado. Para lograrlo, la imagen se ha subdividido en bloques discernibles. Mediante este enfoque, se llevará a cabo un análisis exhaustivo de los tiempos requeridos para la obtención de bordes en cada uno de estos bloques. En esta instancia, la imagen se segmentará en cuatro regiones distintas, cada una de las cuales se almacenará de manera independiente en formato .jpg. Esta metodología permitirá un análisis más profundo de la distribución y velocidad del procesamiento a nivel de cada bloque, proporcionando una visión más completa y detallada de cómo los algoritmos de detección de bordes interactúan con diferentes secciones de la imagen original.

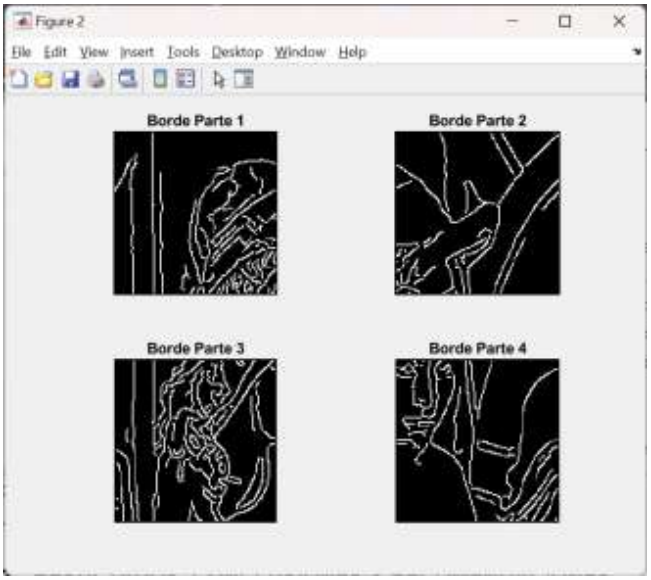

Figura 24. Imagen dividida en bloque

La Figura 24. ilustra los resultados de la ejecución del código, revelando una división de la imagen en cuatro bloques individuales. Cada bloque será objeto de análisis desde diferentes perspectivas. En el primer escenario, se evaluará el tiempo requerido para extraer los bordes de la imagen completa con procesamiento secuencial. Posteriormente, se abordará la fragmentación de la imagen en cuatro bloques separados, cada uno guardado como una imagen independiente. Para esta configuración, se calculará el tiempo empleado en extraer los bordes de cada bloque de manera secuencial. Por último, se llevará a cabo el procesamiento de la imagen y se calculará el tiempo necesario para la detección de bordes en cada bloque, en esta ocasión empleando la función de paralelismo.

```
IMAGEN COMPLETA
Tiempo de procesamiento: 0.14766 segundos
IMAGEN DIVIDA EN 4 PARTES Y SECUENCIAL
Tiempo de procesamiento: 0.134409 segundos
Starting parallel pool (parpool) using the 'local' profile
Connected to the parallel pool (number of workers: 8).
Tiempo de procesamiento para el bloque 1x1: 0.00089526
Tiempo de procesamiento para el bloque 2x1: 0.0009071
Tiempo de procesamiento para el bloque 1x2: 0.00091874
Tiempo de procesamiento para el bloque 2x2: 0.00083082
Parallel pool using the 'local' profile is shutting down.
```

Figura 25. Tiempos de los distintos escenarios

La Figura 25 revela un panorama interesante en cuanto a los tiempos de ejecución en los dos primeros escenarios. En ambos casos, la imagen es procesada secuencialmente, generando una diferencia mínima en los tiempos. Sin embargo, el segundo escenario destaca al dividir la imagen en bloques, lo cual no altera sustancialmente los tiempos. En cambio, el tercer escenario exhibe un cambio marcado en los tiempos de procesamiento para cada bloque. Es evidente que la implementación de paralelismo introduce una notable mejora en la eficiencia temporal. La variación en los tiempos al adoptar el enfoque paralelo subraya la ventaja de este método, demostrando de manera palpable su capacidad para agilizar significativamente el proceso de detección de bordes en cada bloque de la imagen.

3.1.6 COMPARACIÓN DE PROCESAMIENTOS CON IMÁGENES DE DISTINTO TAMAÑO

En este contexto de estudio, se tomó la decisión de emplear la icónica imagen de "Lena" como caso de análisis. Para ello, se mantuvieron intactas sus dimensiones originales, que constan de 512 píxeles de ancho por 512 píxeles de alto. No obstante, con el propósito de explorar distintos escenarios también se optó por considerar una imagen de 225 x 225 y luego se procedió a dividir la imagen en fracciones más pequeñas. En primer lugar, se efectuó una partición que implicó tomar la mitad horizontal de la imagen, dando lugar a una nueva representación de 112 píxeles de alto y 225 píxeles de ancho. Posteriormente, se llevó a cabo una nueva división en la que se seleccionó únicamente una cuarta parte de la imagen original, resultando en una matriz de 112x112 píxeles.

Figura 26. Imágenes analizadas

En esta etapa del análisis, se han contemplado ambos métodos de procesamiento previamente discutidos. En esta ocasión, se procede a exponer los resultados temporales obtenidos mediante el proceso de cálculo de bordes de manera secuencial. Es importante destacar que se ha aplicado esta metodología a las tres distintas imágenes, abarcando tanto la imagen completa como las divisiones en mitades y cuartos de estas.

La Figura 27 revela una variabilidad significativa en los tiempos de procesamiento, lo cual podría atribuirse a la complejidad inherente de la imagen. Al someter la imagen al proceso de procesamiento, se aplica una máscara que captura diversos detalles presentes en la imagen. Estos detalles pueden influir de manera considerable en la velocidad de procesamiento, ya sea ralentizando o acelerando los tiempos requeridos. Esta variabilidad en los tiempos subraya la importancia de considerar la complejidad intrínseca de la imagen y su influencia en la eficiencia del proceso de detección de bordes.

```
Tiempo de procesamiento: 0.16588 segundos
Tiempo de procesamiento: 0.039976 segundos
Tiempo de procesamiento: 0.031134 segundos
Tiempo de procesamiento: 0.020234 segundos
```

Figura 27. Tiempos con procesamiento secuencial

La etapa consecutiva involucró el cálculo de los intervalos temporales necesarios para extraer los bordes de las imágenes, esta vez empleando la técnica del paralelismo. En consonancia con la metodología previa, se ha incorporado las tres imágenes en el análisis y, una vez más, trazado un enfoque gráfico para visualizar cómo los tiempos se relacionan con las dimensiones de las imágenes. Esta representación permite evaluar de manera más holística cómo el uso del paralelismo impacta los tiempos de procesamiento en comparación con el enfoque secuencial.

```
Starting parallel pool (parpool) using the 'local' profile
Connected to the parallel pool (number of workers: 8).
Tiempo de procesamiento para imagen 1: 0.010058 segundos
Tiempo de procesamiento para imagen 2: 0.010017 segundos
Tiempo de procesamiento para imagen 3: 0.0093823 segundos
Tiempo de procesamiento para imagen 4: 0.0092856 segundos
Parallel pool using the 'local' profile is shutting down.
```

Figura 28. Tiempos con procesamiento paralelo

Al comparar la Figura 27 con la Figura 28, se evidencia una notable disparidad en los tiempos de procesamiento. Este contraste sustancial respalda la conclusión de que la

implementación de procesamiento paralelo es significativamente más eficiente en comparación con su contraparte secuencial.

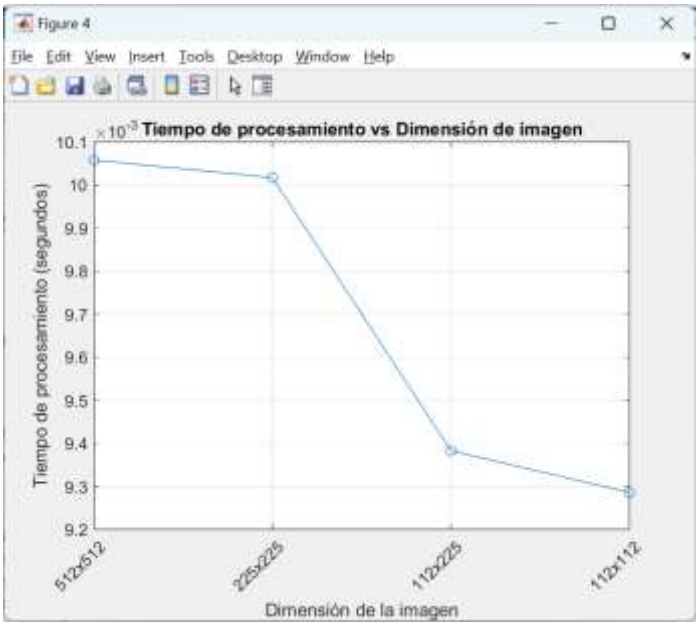

Figura 29. Tiempo procesamiento vs Dimensión con paralelismo

3.2 Conclusiones

El análisis de imágenes se ve notablemente simplificado gracias a la capacidad de dividir las imágenes en partes más pequeñas y manejables. Esta estrategia de segmentación no solo acelera los tiempos de procesamiento, como hemos observado, sino que también proporciona una visión detallada de áreas específicas.

Se concluyo que el análisis de la disparidad en los tiempos de ejecución al aplicar funciones matemáticas en enfoques de procesamiento lineal y paralelo proporciona una comprensión valiosa de la optimización de recursos computacionales en tareas de procesamiento de imágenes.

Al comparar el procesamiento lineal con el enfoque paralelo en la Figura 27 y la Figura 28, hemos observado que el último demuestra una capacidad superior para manejar operaciones intensivas en términos de tiempo. Este descubrimiento subraya la importancia de adoptar el paralelismo en situaciones en las que la eficiencia es primordial.

La comparación de los tiempos de ejecución entre los enfoques lineal y paralelo pone de manifiesto la necesidad de adaptar las estrategias de procesamiento según la complejidad

de la tarea y los recursos disponibles. Esto permite una asignación más eficiente de recursos en función de las demandas del análisis de imágenes.

Al realizar las pruebas en los distintos procesadores, se puede concluir que al comprobar el benchmarking de los distintos procesamientos los tiempos también cambian notablemente, esto también depende de la memoria RAM de cada computador y es un factor del que no se tiene control.

En el contexto del procesamiento, se tienen varias opciones disponibles; sin embargo, se optó por trabajar con imágenes debido a su alta susceptibilidad para dividirse en bloques o segmentos y a su vez no importaría el orden en el que se la analiza.

3.3 Recomendaciones

Al aplicar segmentación a las imágenes, es recomendable utilizar técnicas de segmentación inteligente que se adapten a las características específicas de la imagen y del análisis deseado. Esto permitirá una distribución equilibrada del trabajo entre los bloques y evitará ineficiencias en el procesamiento.

En el caso de segmentar las imágenes en bloques, es esencial optimizar el tamaño y la cantidad de bloques para maximizar la eficiencia. Bloques demasiado pequeños pueden llevar a una sobrecarga de procesamiento, mientras que bloques demasiado grandes pueden no aprovechar al máximo el paralelismo.

Durante la implementación, es esencial realizar un monitoreo constante de los tiempos de procesamiento y las eficiencias obtenidas. Si es necesario, realizar ajustes en los algoritmos, la división de bloques o los recursos hardware para optimizar aún más el rendimiento.

4 REFERENCIAS BIBLIOGRÁFICAS

[1] N. Alexandridis, H-A Choi, B. Narahari, S. Rotensteich and A. Youssef, "A Hierarchical partitionable knowledge based parallel processing system", 3rd Annual Parallel Processing Symposium, 1989-March-29-31.

[2] J. Dongarra and P. Newton, "Overview of VPE: A Visual Environment for Message-Passing Parallel Programming", Heterogeneous Computing Workshop '95 Proceedings of the 4th Heterogeneous Computing Workshop, 1995-April-25.

[3] H. J. Siegel, J. B. Armstrong and D. W. Watson, "Mapping Computer-Vision Related Tasks onto Reconfigurable Parallel Processing Systems", Computer, vol. 25, no. 2, pp. 54-63, Feb 1992.

[4] X. Zhang and Y. Yan, "A Framework of Performance Prediction of Parallel Computing on Nondedicated Heterogeneous NOW", Proceedings of the 1995 International Conference on Parallel Processing, vol. 1k, 1995-Aug.

[5] B Basavaprasad and M Ravi, "A study on the importance of image processing and its applications" in International Journal of Research in Engineering and Technology, vol. 03, no. 03/may-2014/NCRIET-2014, ISSN 2319-1163.

[6] John G. Proakis y Dimitris G. Manolakis, "Digital Signal Processing", pp. 87-93, 2018.

[7] Patrick Gaydecki, "Foundations of Digital Signal Processing: Theory, Algorithms and Hardware Design". 2014 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, pp. 580-587.

[8] Bernard Widrow y Samuel D. Stearns. "Adaptive Signal Processing", vol. 30, no. 2, pp. 88-97, 2009.

[9] T. D. Braun, Parallel Algorithms for Singular Value Decomposition as Applied to Failure Tolerant Manipulators, Dec. 1997.

[10] A. A. Maciejewski and J. M. Reagin, "A parallel algorithm and architecture for the control of kinematically redundant manipulators", IEEE Transactions on Robotics and Automation, vol. 10, no. 4, pp. 405-414, Aug. 1994.

[11] R. R. Ulrey, A. A. Maciejewski and H. J. Siegel, "Parallel algorithms for singular value decomposition", Eighth International Parallel Processing Symposium (IPPS '94), pp. 524-533, 1994-Apr.

[12] JaJa, Joshep. (n.d.). Designing efficient parallel algorithms: models and paradigms with applications to image processing. [1993]

[13] W. Gropp and E. Lusk. Dynamic process management in an mpi setting. Proceedings of the 7th IEEE Symposium on Parallel and Distributeed Processing, pages 530-533, San Antonio, USA, 1995.

[14] D. Juvin, J.L. Basille, H. Essafi and J.Y. Latil, "SYMP ATI 2 A 15D Processor array for image application", Signal Processing IV: Theories and Applications EURASIP, pp. 311-314, 1988.

[15] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff and Hartwig Adam, "Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation", ECCV 2018 LNCS 11211, pp. 833-851, 2018.

[16] E.E. Pissaloux and P. Bonnin, "On the Evolution of Parallel Computer Dedicated to Image Processing through Examples of Some French Computer", Digital Signal Processing Journal, vol. 7, no. 1, pp. 13-27, January 1997.

[17] R. C. Gonzalez, R. E. Woods y S. L. Eddins, Digital Image Processing Using Matlab, Prentice Hall, 1999.

[18] D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagum, et al., "The nas parallel benchmarks", The International Journal of Supercomputing Applications, vol. 5, no. 3, pp. 63-73, 1991.

[19] A.M. Khan and S. Ravi, "Image segmentation methods: a comparative study", International Journal of Soft Computing and Engineering, pp. 84-93, 2013.

[20] C. Coleman, D. Narayanan, D. Kang, T. Zhao, J. Zhang, L. Nardi, et al., "Dawnbench: An end-to-end deep learning benchmark and competition", Training, vol. 100, no. 101, pp. 102, 2017.

[21] V. J. Reddi, C. Cheng, D. Kanter, P. Mattson, G. Schmuelling, C.-J. Wu, et al., "Mlperf inference benchmark", arXiv preprint, 2019.

[22] J.-Y. Zhu, T. Park, P. Isola and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks", Proceedings of the IEEE international conference on computer vision, pp. 2223-2232, 2017.

[23] Shisanu T. and Prabhas C., "Parallel Genetic Algorithm for Finite-State Machine Synthesis from Input/Output Sequences", Evolutionary Computation Parallel Processing Workshop, Genetic and Evolutionary Computation Conf, Las Vegas, Nevada, USA, pp.20-24, 2000.

5 ANEXOS

ANEXO I. Procesamiento Lineal con imágenes de distinto tamaño (Matlab)

ANEXO II. Procesamiento Paralelo con imágenes de distinto tamaño (Matlab)

ANEXO III. Procesamiento Lineal y secuencial con bloques (Matlab)

ANEXO I. PROCESAMIENTO LINEAL CON IMÁGENES DE DISTINTO TAMAÑO (MATLAB)

```
clc,
clear all,

% Leer la imagen
imagen = imread('imagen.jpg');
imagen1 = imread('borde_bloque_1.jpg');
imagen2 = imread('bordes_partes_1_y_2.jpg');
% Convertir la imagen a escala de grises
imagen_grises = rgb2gray(imagen);
% Obtener dimensiones de las imágenes
[dim_filas, dim_columnas, ~] = size(imagen);
[dim_filas1, dim_columnas1, ~] = size(imagen1);
[dim_filas2, dim_columnas2, ~] = size(imagen2);
% Aplicar el operador de detección de bordes de Canny
%Imagen completa
tic;
umbral_min = 0.1; % Umbral mínimo
umbral_max = 0.3; % Umbral máximo
bordes = edge(imagen_grises, 'Canny', [umbral_min, umbral_max]);
tiempo_procesamiento = toc;

% Inicializar vectores para almacenar los tiempos de procesamiento
y las dimensiones
dimensiones = [dim_filas, dim_filas1, dim_filas2];
tiempos = zeros(1, 3);

%Mitad de la imagen
tic;
umbral_min = 0.1; % Umbral mínimo
umbral_max = 0.3; % Umbral máximo
bordes1 = edge(imagen1, 'Canny', [umbral_min, umbral_max]);
tiempo_procesamiento1 = toc;
%Cuarta parte de la imagen
tic;
umbral_min = 0.1; % Umbral mínimo
umbral_max = 0.3; % Umbral máximo
bordes2 = edge(imagen2, 'Canny', [umbral_min, umbral_max]);
```

```
tiempo_procesamiento2 = toc;
% Mostrar los tiempos de ejecución
disp(['Tiempo de procesamiento: ', num2str(tiempo_procesamiento), '
segundos']);
disp(['Tiempo de procesamiento: ', num2str(tiempo_procesamiento1),
' segundos']);
disp(['Tiempo de procesamiento: ', num2str(tiempo_procesamiento2),
' segundos']);
```

ANEXO II. PROCESAMIENTO PARALELO CON IMÁGENES DE DISTINTO TAMAÑO (MATLAB)

```
clc;
clear all;
% Verificar si existe una sesión de parpool activa
if ~isempty(gcp('nocreate'))
    % Si existe, cerrar la sesión de parpool
    delete(gcp);
end
% Leer las tres imágenes en escala de grises
imagen1 = imread('imagen.jpg'); % Reemplaza con la ruta de tu
primera imagen en escala de grises
imagen2 = imread('mitad1.jpg'); % Reemplaza con la ruta de tu
segunda imagen en escala de grises
imagen3 = imread('bloque1.jpg'); % Reemplaza con la ruta de tu
tercera imagen en escala de grises

% Parámetros para el operador Canny
umbral_min = 0.1;
umbral_max = 0.3;

% Convertir imágenes a escala de grises si no lo están
if size(imagen1, 3) == 3
    imagen1 = rgb2gray(imagen1);
end
if size(imagen2, 3) == 3
    imagen2 = rgb2gray(imagen2);
end
if size(imagen3, 3) == 3
    imagen3 = rgb2gray(imagen3);
end

% Crear una celda con las imágenes
imagenes = {imagen1, imagen2, imagen3};

% Inicializar celdas para los bordes y tiempos
bordes = cell(1, 3);
```

```
tiempos = zeros(1, 3);

parpool();
% Procesamiento paralelo para cada imagen y cálculo de tiempos
parfor i = 1:3
    imagen_actual = imagenes{i};

    tic;
    % Aplicar el operador de detección de bordes de Canny
    bordes_actual = edge(imagen_actual, 'Canny', [umbral_min,
umbral_max]);
    bordes{i} = bordes_actual;

    % Calcular el tiempo de procesamiento y almacenarlo en el
vector
    tiempos(i) = toc/25;
end

% Mostrar los tiempos de procesamiento
for i = 1:3
    disp(['Tiempo de procesamiento para imagen ', num2str(i), ': ',
num2str(tiempos(i)), ' segundos']);
end

% Graficar tiempo vs dimensión de imagen con puntos unidos
dimensiones = cell(1, 3);
for i = 1:3
    [filas, columnas] = size(imagenes{i});
    dimensiones{i} = [num2str(filas), 'x', num2str(columnas)];
end

figure;
plot(1:3, tiempos, 'o-'); % Puntos unidos con líneas
xticks(1:3);
xticklabels(dimensiones);
xlabel('Dimensión de la imagen');
ylabel('Tiempo de procesamiento (segundos)');
title('Tiempo de procesamiento vs Dimensión de imagen');
grid on;

```

```
xtickangle(45); % Rotar las etiquetas del eje x para mayor
legibilidad
% Cerrar el parpool
delete(gcp);
```

ANEXO III. PROCESAMIENTO LINEAL Y SECUENCIAL CON BLOQUES (MATLAB)

```
clear all;
clc;

% Cargar la imagen
imagen = imread('imagen.jpg'); % Reemplaza con la ruta de tu imagen
imshow(imagen);
title('Imagen Original');
% Convertir la imagen a escala de grises
tic;
imagen_grises = rgb2gray(imagen);
tiempo_conversion = toc;

% Aplicar el operador de detección de bordes de Canny
tic;
umbral_min = 0.1; % Umbral mínimo
umbral_max = 0.3; % Umbral máximo
bordes = edge(imagen_grises, 'Canny', [umbral_min, umbral_max]);
tiempo_procesamiento = toc;

% Mostrar la imagen original y los bordes encontrados
subplot(1, 2, 1);
imshow(imagen);
title('Imagen original');

subplot(1, 2, 2);
imshow(bordes);
title('Bordes de la imagen');

% Mostrar los tiempos de ejecución
disp('IMAGEN COMPLETA')
disp(['Tiempo de procesamiento: ', num2str(tiempo_procesamiento), '
segundos']);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%Imagen dividida y secuencial%%

% Dividir la imagen en 4 partes
[m, n, ~] = size(imagen);
```

```
parte1 = imagen(1:fix(m/2), 1:fix(n/2), :);
parte2 = imagen(1:fix(m/2), fix(n/2)+1:end, :);
parte3 = imagen(fix(m/2)+1:end, 1:fix(n/2), :);
parte4 = imagen(fix(m/2)+1:end, fix(n/2)+1:end, :);

% Convertir a escala de grises
gris1 = rgb2gray(parte1);
gris2 = rgb2gray(parte2);
gris3 = rgb2gray(parte3);
gris4 = rgb2gray(parte4);
% Aplicar el operador edge a cada parte
tic;
borde1 = edge(gris1, 'canny');
borde2 = edge(gris2, 'canny');
borde3 = edge(gris3, 'canny');
borde4 = edge(gris4, 'canny');
tiempo = toc;

disp('IMAGEN DIVIDA EN 4 PARTES Y SECUENCIAL');
fprintf('Tiempo de procesamiento: %.6f segundos\n', tiempo);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%
%imagen dividia y paralelismo
% Verificar si existe una sesión de parpool activa
if ~isempty(gcp('nocreate'))
    % Si existe, cerrar la sesión de parpool
    delete(gcp);
end
suma_tiempos_promedio_total = 0;
% Convertir la imagen a escala de grises si es necesario
if size(imagen, 3) == 3
    imagen = rgb2gray(imagen);
end

% Obtener las dimensiones de la imagen
[filas, columnas] = size(imagen);

% Dividir la imagen en 4 partes
filas_por_bloque = floor(filas / 2);
columnas_por_bloque = floor(columnas / 2);
```

```
78
79  % Valor constante a sumar a cada tiempo
80  %valor_constante = 0.005; % Cambia este valor según tus necesidades
81
82  parpool();
83  % Bucle para recorrer cada bloque de la imagen
84
85  parfor i = 1:2
86      for j = 1:2
87          % Obtener los límites del bloque actual
88          inicio_fila = (i - 1) * filas_por_bloque + 1;
89          fin_fila = min(i * filas_por_bloque, filas);
90          inicio_columna = (j - 1) * columnas_por_bloque + 1;
91          fin_columna = min(j * columnas_por_bloque, columnas);
92
93          % Inicializar una variable para sumar los tiempos de cada
94  píxel en el bloque
95          suma_tiempos_bloque = 0;
96
97
98          % Bucle para recorrer cada píxel del bloque
99
100         for k = inicio_fila:fin_fila
101
102             for l = inicio_columna:fin_columna
103                 tic; % Comenzar a medir el tiempo
104                 % Procesar el píxel utilizando la función canny
105                 borde_pixel = edge(imagen(k, l), 'canny'); % Cambia
106 aquí el tipo de operador de borde si lo deseas
107
108                 %tiempo = toc + valor_constante; % Sumar el valor
109 constante al tiempo medido
110                 tiempo = toc;
111                 suma_tiempos_bloque = suma_tiempos_bloque + tiempo;
112 % Sumar el tiempo al total
113             end
114
115         end
116        % Calcular el tiempo promedio para el bloque actual
117
```

```
        cantidad_pixeles_bloque = (fin_fila - inicio_fila + 1) *
(fin_columna - inicio_columna + 1);
        tiempo_promedio_bloque = suma_tiempos_bloque /
cantidad_pixeles_bloque;

        % Mostrar el tiempo promedio para el bloque actual
        disp(['Tiempo de procesamiento para el bloque ',
num2str(i), 'x', num2str(j), ': ',
num2str(tiempo_promedio_bloque)]);
        % Agregar el tiempo promedio del bloque actual a la suma
total
        suma_tiempos_promedio_total = suma_tiempos_promedio_total +
tiempo_promedio_bloque;
        %timeit(borde_pixel)
    end
end

% Mostrar los bordes
figure;
subplot(2, 2, 1), imshow(borde1), title('Borde Parte 1');
subplot(2, 2, 2), imshow(borde2), title('Borde Parte 2');
subplot(2, 2, 3), imshow(borde3), title('Borde Parte 3');
subplot(2, 2, 4), imshow(borde4), title('Borde Parte 4');


% Cerrar el parpool
delete(gcp);
```