Salamanca, Guanajuato, a 17 de septiembre del 2019.

M. en I. HERIBERTO GUTIÉRREZ MARTIN
JEFE DE LA UNIDAD DE ADMINISTRACIÓN ESCOLAR
P R E S E N T E.-

Por medio de la presente, se otorga autorización para proceder a los trámites de impresión, empastado de tesis y titulación al alumno **Yair Alejandro Andrade Ambriz** del ***Programa de Maestría en Ingeniería Eléctrica*** y cuyo número de ***NUA*** es: **801346** del cual soy director. El título de la tesis es: **Extracción de Características en Imágenes Digitales Usando Programación en Paralelo.**

Hago constar que he revisado dicho trabajo y he tenido comunicación con los sinodales asignados para la revisión de la tesis, por lo que no hay impedimento alguno para fijar la fecha de examen de titulación.

A T E N T A M E N T E

NOMBRE Y FIRMA
DIRECTOR DE TESIS
SECRETARIO
Dra. Dora Luz Almanza Ojeda

NOMBRE Y FIRMA
DIRECTOR DE TESIS
Dr. Sergio Eduardo Ledesma Orozco

NOMBRE Y FIRMA
PRESIDENTE
Dr. José Ruiz Pinales

NOMBRE Y FIRMA
VOCAL
Dr. Fernando Enrique Correa Tomé

Universidad de Guanajuato

Campus Irapuato - Salamanca
División de Ingenierías

"Extracción de Características en Imágenes Digitales Usando Programación en Paralelo"

TESIS PROFESIONAL

QUE PARA OBTENER EL GRADO DE
MAESTRO EN INGENIERÍA ELÉCTRICA
(Opción: Instrumentación y Sistemas Digitales)

PRESENTA:

Ing. Yair Alejandro Andrade Ambriz

DIRECTORES:

Dr. Sergio Eduardo Ledesma Orozco
Dra. Dora Luz Almanza Ojeda

Salamanca, Guanajuato. Septiembre, 2019.

Dedicatoria

Dedico este trabajo a mis padres, Sergio Andrade Cahue y Ma. Guadalupe Ambriz Velazquez, quienes siempre han estado ahí para brindar su incondicional apoyo. A mi hermano Sergio Andrade Ambriz, por su apoyo y consejo en mi vida.

Y especialmente dedico este trabajo a la memoria y vida de mi hermano Mario Alberto Andrade Ambriz, quien siempre creyó en mi y me motivó a superarme siempre.

"La muerte no nos roba a los seres amados. Al contrario, nos los guarda y nos los inmortaliza en el recuerdo."
François Mauriac

Agradecimientos

Agradezco a mis asesores, el Dr. Sergio Eduardo Ledesma Orozco y la Dra. Dora Luz Almanza Ojeda, quienes ofrecieron el total apoyo e invaluable consejo para el trabajo presente.

Agradecimientos Institucionales

Al Consejo Nacional de Ciencia y Tecnología de México, bajo la beca otorgada en la convocatoria “Becas Nacionales 2017 Segundo Periodo”, con el número de registro 846226/634545.

A la Dirección de Apoyo a la Investigación y al Posgrado por el apoyo otorgado en la convocatoria “Programa de Apoyo a los Posgrados 2017”, con el número de estudiante 801346.

Resumen

En la actualidad se tiene un gran avance en la construcción de procesadores con un gran número de núcleos, esto conlleva a tener más recursos computacionales a utilizar. De igual manera se tiene en cuenta que el procesamiento de imágenes digitales está en un punto importante de su desarrollo, ya que se utiliza en un gran número de áreas como puede ser en imágenes médicas para la detección de cáncer, en visión computacional para procesos industriales, entre otros.

Este avance en los dos campos está siendo ganado por el hardware, lo que lleva a un desperdicio de recursos. La mayoría de los algoritmos para procesar imágenes digitales usan una arquitectura secuencial, por lo cual no se está exigiendo al máximo a los procesadores. Para evitar este desperdicio, se plantea el procesamiento en paralelo. Este paradigma de programación propone ventajas, como lo es, reducir el tiempo de cómputo usando todo el hardware disponible, obteniendo muy poco o casi un nulo desperdicio de recursos computacionales.

El presente trabajo tiene como objetivo diseñar e implementar algoritmos de bajo nivel para optimizar el procesamiento de una convolución de dos dimensiones, esto con la finalidad de reducir el tiempo de cómputo utilizado por los algoritmos para poder obtener resultados en la menor cantidad de tiempo posible. Se proponen cuatro métodos de estudio usando multihilos, cada método se implementa de forma única y se comparan los tiempos de cómputo de cada uno para conocer que método es el que mejor desempeño obtiene. De igual manera se compara el tiempo de cómputo usado por los casos de estudio propuestos con frameworks generales, para tener una métrica real de la librería implementada con librerías ya estandarizadas como OpenCL y OpenMP.

Para realizar la labor descrita anteriormente, se propuso la implementación de diversas clases usando el lenguaje de programación C++, encapsulando las llamadas a sistema y otorgando una capa de alto nivel para un fácil uso posterior de la librería. Es importante recalcar que actualmente la mayoría de lenguajes de programación soportan multihilos, como pueden ser Java, C#, Python, entre otros.

Abstract

Currently there is a great advance in the construction of processors with many cores, this leads to having more computational resources to use. Similarly, it is considered that digital image processing is at an important point in its development, since it is used in many areas such as medical images for the detection of cancer, in computational vision for industrial processes, among others.

This advance in the two fields is being won by the hardware, which leads to a waste of resources. Most of the algorithms for processing digital images use a sequential architecture, which is why the processors are not being demanded to the maximum. To avoid this waste, parallel processing is considered. This paradigm of programming proposes advantages, such as reducing computation time using all the available hardware, obtaining very little or almost no waste of computational resources.

The objective of this work is to design and implement low-level algorithms to optimize the processing of a two-dimensional convolution, this in order to reduce the computation time used by the algorithms to reach a response in the least amount of time possible. Four case studies are proposed using multithreading, in each case they are implemented in a unique way and the computation times are compared to know which case is the one that obtains the best performance. In the same way, the computation time used by the proposed case studies is compared with general frameworks, in order to have a real metric of the library implemented with standardized libraries like OpenCL and OpenMP.

To carry out the work described above, the implementation of various classes was proposed using the C++ programming language, encapsulating system calls and granting a high-level layer for easy later use of the library. It is important to emphasize that currently most programming languages support multithreading, such as Java, C#, Python, among others.

Índice general

Dedicatoria II

Agradecimientos III

Agradecimientos Institucionales IV

Resumen V

Abstract VI

1. Introducción **1**

1.1. Objetivos 2

1.2. Objetivos Específicos 2

1.3. Justificación 2

1.4. Estado del Arte 3

1.5. Organización de la Tesis 7

2. Fundamentos Teóricos **8**

2.1. Imágenes 8

2.1.1. Características de una Imagen Digital 9

2.2. Convolución 12
2.2.1. Matriz Convolucional 13
2.3. Red Neuronal Convolucional 15
2.4. Cómputo Paralelo 15
2.4.1. Algoritmo 17
2.4.2. Hilo (Thread) 17
2.4.3. Algoritmos Paralelos y Arquitecturas Paralelas 18
2.4.4. Gastos de Comunicación 19
2.4.5. Mecanismos de Sincronización 19
2.4.6. Factor de Aceleración 21
2.4.7. Ley de Amdahl 22
2.4.8. Ley de Gustafson-Barsis 23

3. Metodología **25**

3.1. Metodología Propuesta 25
3.2. Método 1: Uno a uno 28
3.3. Método 2: Bloque fijo 30
3.4. Método 3: Bloque dinámico 32
3.5. Método 4: Todos a uno 34
3.6. Desarrollo de algoritmos 36

4. Pruebas y Resultados **41**

4.1. Extracción de Características 41
4.2. Desempeño Computacional de los Métodos Propuestos 44

5. Conclusión **49**

Índice de figuras **X**

Índice de tablas **XII**

Índice de códigos **XIII**

Bibliografía **XIV**

Capítulo 1

Introducción

Actualmente el procesamiento digital de imágenes tiene aplicación en muchas áreas diferentes desde la medicina, hasta aplicaciones industriales requiriendo visión por computadora, entre otros. Este incremento en el uso del procesamiento de imágenes digitales se puede simplificar en dos propósitos generales [1]:

1. Mejorar la apariencia visual de las imágenes para un humano observador.
2. Preparar las imágenes para la medición o la extracción de las características existentes en ellas, como pueden ser bordes, rectas, puntos, entre otros.

De estos propósitos generales se puede comprender el uso actual del procesamiento de imágenes enfocado en realzar las características que posteriormente un clasificador, ya sea una red neuronal, una máquina de vectores de soporte (Support Vector Machines, SVM), utilice para automatizar un robot industrial, automatizar la conducción de un automóvil, entre otros. Inclusive actualmente la mayoría de los teléfonos inteligentes que incluyen un módulo de inteligencia artificial usan la información de extracción de características para seleccionar la escena de la imagen a capturar y pre-seleccionar los mejores parámetros para esa toma.

Como se puede observar cada día es mayor el uso del procesamiento de imágenes digitales aún en la vida diaria de una persona común, por esta razón es de crucial importancia el tiempo de cómputo usado por los algoritmos, ya que esta extracción de características se usa en tiempo real y no se puede permitir un retraso en la ejecución del procesamiento en la imagen.

1.1. Objetivos

Realizar procesamientos en paralelo de imágenes digitales adquiridas de Internet (cualquier base de datos pre-clasificada), para la extracción de sus características visuales. En este proyecto de tesis se propone el diseño e implementación de una capa de red neuronal convolucional, bajo una arquitectura de programación en paralelo, usando funciones nativas del sistema o funciones de bajo nivel, para la extracción de características visuales como son bordes, rectas, puntos, entre otros. Los algoritmos desarrollados e implementados se compararán con técnicas de ejecución en paralelo y de ejecución lineal para medir su rendimiento, comparando los resultados del tiempo de cómputo usado por las distintas técnicas de procesamiento. De igual manera, se compararán las características extraídas por el algoritmo implementado con librerías de uso general como puede ser OpenCV (Open Source Computer Vision Library), para saber el nivel de extracción presente.

Asimismo, los resultados en exactitud y tiempo del procesamiento de las imágenes digitales está orientado para brindar una mejor respuesta para métodos de redes neuronales de aprendizaje profundo.

1.2. Objetivos Específicos

- Implementar un algoritmo eficiente que calcule la convolución de una matriz en dos dimensiones con diferentes máscaras convolutivas para la extracción de características.
- Diseñar e implementar diferentes arquitecturas paralelas para la extracción de las características visuales de una imagen digital, enfocadas principalmente en la reducción de tiempos de cómputo optimizando los recursos físicos del computador.
- Asignar variables seguras para el intercambio de memoria entre diferentes hilos de procesamiento.
- Elaborar un sistema para la lectura de imágenes digitales, su posterior procesamiento con arquitectura paralela y el guardado de dichas características.

1.3. Justificación

En la actualidad un gran número de áreas se están sumando al procesamiento digital de imágenes, no necesariamente al implementar nuevos algoritmos, sino al ocupar las características extraídas de las imágenes digitales. Cada día un mayor número de usuarios comienzan a usar el procesamiento de imágenes en tiempo real, inclusive los teléfonos inteligentes ya cuentan con un módulo de detección de escenas para pre-seleccionar la mejor configuración para realizar la toma.

En procesos industriales, se ha incrementado el uso de visión por computadora, dando en tiempo real, una perspectiva de la calidad de sus productos. De igual manera, en las ciencias médicas ha aumentado el uso del procesamiento de imágenes digitales, debido a que una malformación en alguna parte del cuerpo es detectada y segmentada por un computador alcanzando hasta un 98 % de exactitud [2].

Del mismo modo, debido al avance en las ópticas de cámaras fotográficas y sensores ahora se cuenta con imágenes de un mayor tamaño, incrementando así las necesidades técnicas requeridas para procesarlas. Con esto en mente es que la velocidad de procesamiento o tiempo de cómputo usado por los algoritmos toma una parte fundamental en esta área. Ha llegado el momento en que no basta solamente hacer eficiente un proceso o un algoritmo de procesamiento, ahora se tiene que hacer uso de todos los recursos computacionales existentes.

De la misma manera que las diferentes áreas que ocupan el procesamiento de imágenes, el manejo autónomo ya sea desde vehículos a 4 ruedas o drones, ha ido en aumento gracias al avance en el poder computacional y gracias a las redes neuronales de aprendizaje profundo, que pueden obtener un amplio conjunto de características que combinado con un clasificador, se está alcanzando el verdadero manejo autónomo.

En la Figura 1.1, se muestra cómo han ido evolucionando los procesadores actuales, su frecuencia de reloj se ha quedado en niveles similares al pasar de los años, mientras que el número de núcleos en el procesador ha ido en aumento, permitiendo así el uso de la programación en paralelo.

De esta forma se tiene que adoptar un paradigma de la programación, el cual es el cómputo paralelo, el cual permite ejecutar más de una instrucción a la vez, reduciendo los tiempos de cómputo requeridos para procesar la imagen digital, obteniendo de manera eficaz las características extraídas.

Este trabajo se realiza con el fin de entregar un módulo capaz de realizar en tiempo real la extracción de características de imágenes digitales usando la programación en paralelo, este será capaz de crear un modelo a bajo nivel de un conjunto de hilos, los cuales se ejecutarán simultáneamente cuando se requiera procesar las imágenes. Siendo así que el usuario del sistema no deberá tener conocimientos de procesamiento en paralelo, ya que la librería hará todo el trabajo.

1.4. Estado del Arte

Uno de los principales objetivos científicos ha sido diseñar y construir máquinas que sean capaces de realizar procesos con cierta inteligencia, por esta razón se han desarrollado las redes neuronales artificiales.

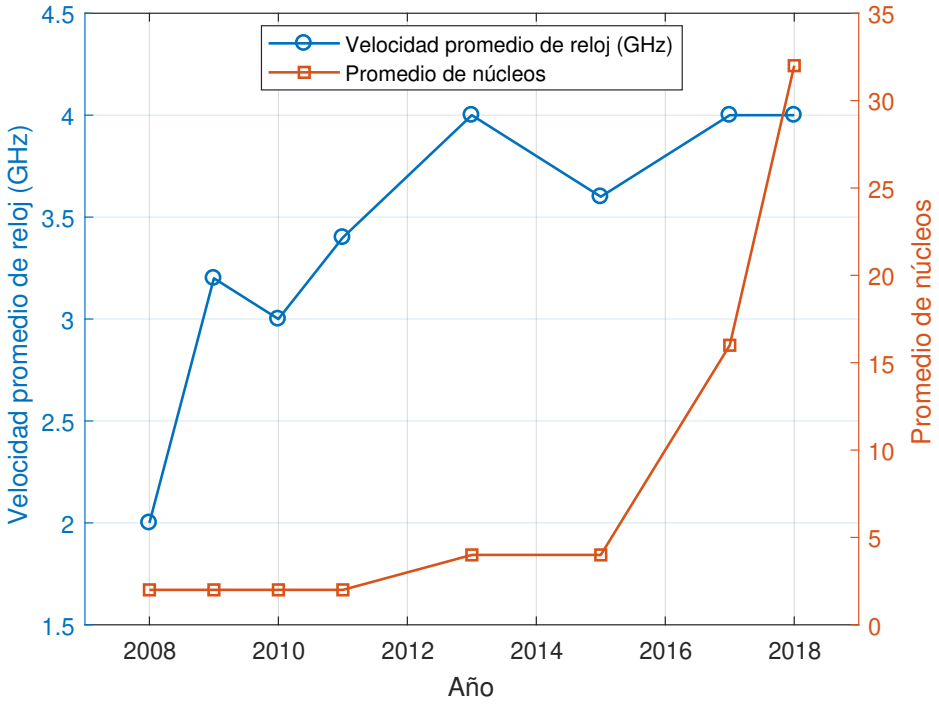


Figura 1.1: Evolución de la frecuencia de reloj y núcleos de los procesadores.

En 1958 Rosenblatt propuso el Perceptron [3], la unidad desde la cual nacerían y se potenciarían las redes neuronales. Este perceptron toma varias entradas y produce una salida binaria. Para calcular las salidas, se introduce el concepto de peso, el cual es un número real que expresa la importancia de esa respectiva entrada con la salida deseada. En la Figura 1.2 se muestra la estructura de entradas y salidas de un perceptron.

En 1980 Fukushima propuso una red neuronal multicapa para tareas de reconocimiento de patrones [4], que sentaron las bases para las redes neuronales convolucionales. Implementaron de igual manera una red neuronal auto organizada, lo que da a lugar a "aprender" sin un supervisor, o sin un maestro.

En 1985 Rumelhart *et al.* desarrollaron los perceptrones multicapa, los cuales eran la agrupación de múltiples perceptrones como se muestra en la Figura 1.2 para entrada o salida, y capas entre las mismas [5]. En la Figura 1.3 se muestra la representación de un perceptron multicapa, donde se puede apreciar la composición de múltiples perceptrones tanto en la entrada como en la salida. Para cada perceptron simple se define su salida y de la ecuación 1.1.

$$y_i = f\left(\sum_{j=1}^{n} w_{ij}x_{ij}\right) \tag{1.1}$$

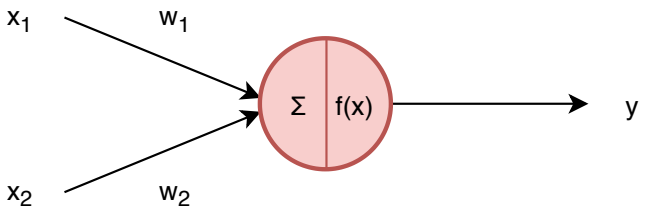


Figura 1.2: Representación de un perceptron, donde x_1 y x_2 son las entradas, w_1 y w_2 los pesos de cada entrada, $\sum$ la sumatoria de los pesos con su respectiva entrada y $f(x)$ la función de activación del perceptron.

Además propusieron el algoritmo backpropagation, el cual hizo posible entrenar redes neuronales multicapa de manera supervisada. Este algoritmo calcula el error obtenido en la salida, el se cual va propagando hacia capas anteriores para ir haciendo ajustes en los pesos, o minimizando costos en cada iteración, para lograr que la red pueda clasificar las entradas correctamente.

En 1992 Weng *et al.* implementaron una red neuronal multicapa capaz de aprender patrones nuevos, además de ejecutar un algoritmo para la reducción de dimensionalidad junto a la reducción de datos innecesarios para la detección de características, dejando solamente información útil [6]. Los resultados indican que es posible obtener de forma automática una inteligencia artificial capaz de crecer, organizarse y aprender de forma autónoma.

El 2009 Sankaradas *et al.* presentaron un sistema para acelerar el procesamiento de las redes neuronales convolucionales. El procesamiento consistía en primitivas para una convolución 2D en paralelo usando sub muestreo y funciones no lineales específicas para CNN (Convolutional Neural Networks) [7]. Implementaron un prototipo de CNN en un FPGA (Field-Programmable Gate Array) el cual fue 31 veces más rápido que su implementación en software. Viendo así la importancia de los procesos en paralelo para el factor de aceleramiento en los procesos de redes neuronales convolucionales.

El 2011 Cireşan *et al.* presentaron un diseño totalmente parametrizable implementado en un GPU (Graphics Processing Unit), aprovechando el alto grado de paralelización que un dispositivo de este tipo permite, consiguiendo un entrenamiento de una red neuronal de aprendizaje profundo [8], que sin el desempeño computacional de un coprocesador paralelo no sería posible.

El 2014 Kim *et al.* implementaron un algoritmo para convolución 2D de forma paralela demandando el alto desempeño que una computadora multi-núcleo provee. Para la evaluación del desempeño usaron la convolución con un kernel de $3x3$ para comparar su velocidad de ejecución, observando que al usar tecnologías basadas en multi-hilos obtenían un mejor

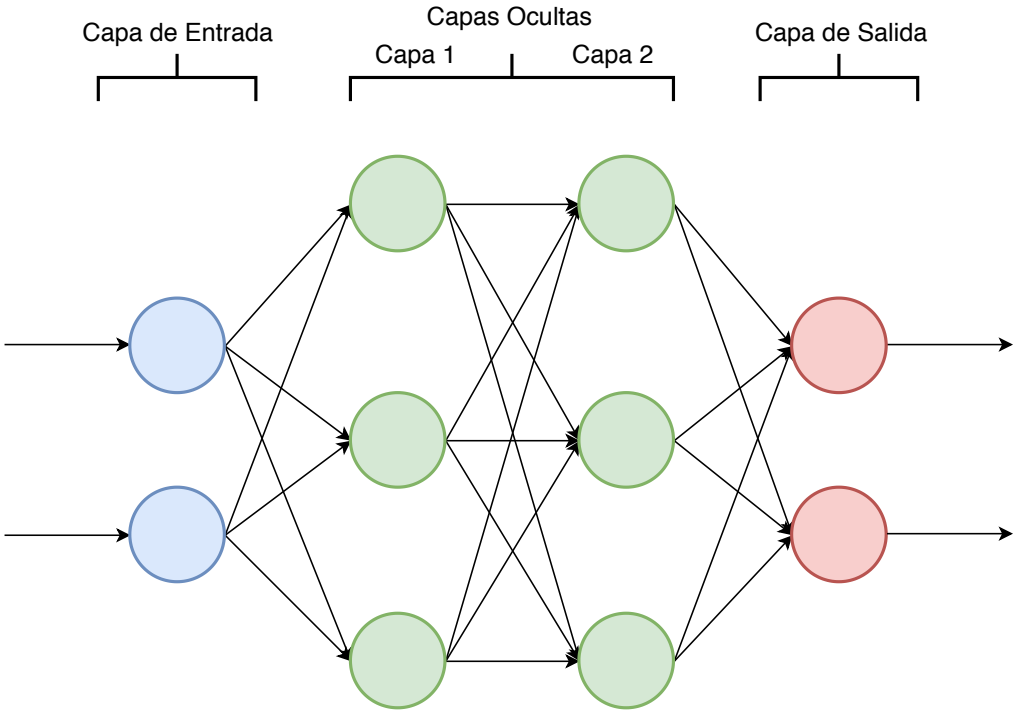


Figura 1.3: Representación de un perceptron multicapa.

desempeño [9].

El 2017 Tousimojarad *et al.* compararon el desempeño de los algoritmos para convolucionar imágenes en dos dimensiones. Compararon el desempeño paralelo de distintas librerías como OpenMP (Open Multi-Processing), OpenCL (Open Computing Language) y GPRM implementados en un procesador Intel Xeon Phi 5110P [10].

Al analizar el estado del arte queda claro que se requiere un alto poder computacional para el procesamiento de imágenes, precisamente en las redes neuronales, siendo estas por su arquitectura altamente paralelizables. El actual incremento de núcleos físicos y lógicos en las computadoras accesibles por cualquier persona proveen el hardware necesario para la implementación de algoritmos paralelos para poder extraer todo el desempeño que los equipos puedan otorgar.

1.5. Organización de la Tesis

El trabajo está divido en 5 capítulos. El texto en supra líneas proporcionó una pequeña introducción al mundo del procesamiento de imágenes digitales.

El capítulo 2 contiene el marco teórico necesario y que pone las bases para realizar este proyecto.

En el capítulo 3 se describe la metodología usada para llevar a cabo los objetivos generales y particulares del trabajo.

El capítulo 4 detalla cada una de las pruebas y procedimientos para la obtención de los resultados.

En el capítulo 5 se expone el análisis de los resultados obtenidos en el capítulo 4, llegando así a las conclusiones del proyecto y perspectivas a futuro.

Capítulo 2

Fundamentos Teóricos

En este capítulo se describen los principales trabajos relacionados al uso del procesamiento de imágenes digitales enfocado a la extracción de características. Se describen de igual manera, la teoría que impone las bases para el paradigma de la programación en paralelo necesarios para comprender los alcances de este proyecto.

2.1. Imágenes

Una imagen puede ser definida como una función en dos dimensiones, $f(x, y)$, donde x y y son coordenadas espaciales, y la amplitud de f en cualquier par de coordenadas es llamado intensidad de nivel de gris de la imagen en ese punto [11]. De esta forma cuando se tiene un número finito de coordenadas espaciales y niveles de intensidad será una imagen digital. Cada valor finito de una imagen digital puede ser referido como elemento de la imagen, o píxeles [11]. Este píxel permitirá conocer el valor de la intensidad en un punto específico de una imagen digital.

En la actualidad un campo muy grande está adoptando el uso del procesamiento de imágenes digitales, esto para trasladar la información de la vida real a un conjunto de números o píxeles entendibles y procesables por un equipo computacional. La imagen capturada y trasladada contiene muchos datos generales por lo cual se tiene que procesar, esto permite realzar las características principales de la imagen, dando como resultado una conversión de los datos generales a información útil para una posterior interpretación o clasificación.

Existen diversos formatos con los cuales se guardará la información contenida en los píxeles a un disco duro o a un contenedor para su posterior uso. Cada formato tiene ventajas y desventajas que van desde el número de canales, hasta su paleta de colores, que pueden representar de una mejor forma la información capturada por el sensor original a una forma

matricial.

Uno de los formatos más sencillos es el BMP (Windows Bitmap), este formato soporta imágenes con un tamaño de 1, 4, 8, 16, 24 y 32 bits por píxel [12]. Este formato guarda generalmente la información sin compresión o compresión sin pérdida RLE (Run-Length Encoding) para un tamaño de 4 u 8 bits por píxel, por lo cual se puede decir que es un formato sin pérdidas, o sea que traslada de forma fidedigna la información del mundo real a un conjunto de píxeles.

Un formato muy usado para el almacenamiento de imágenes es GIF (Graphics Interchange Format), este archivo permite el poner en un solo archivo múltiples imágenes, la desventaja es que el número máximo de colores es de 256 [12], por lo cual tiene una calidad baja para la correcta interpretación de las características capturadas por el sensor original.

Otro formato, el cual es de los más usados en la actualidad es el JPEG (Joint Photographic Experts Group). Este formato fue creado por la organización del mismo nombre para crear un estándar para comprimir imágenes [12], el cual da una gran compresión hasta del 90 % con respecto al formato BMP. La desventaja de este formato es la pérdida en sus datos, debido a su gran compresión se debe sacrificar la información contenida en la misma.

PNG (Portable Network Graphics) es uno de los formatos más distribuidos actualmente a través de Internet. Este formato usa una comprensión sin pérdida que soporta hasta 48 bits por píxel en las imágenes a color. Incluye un canal alpha, que controlará la transparencia de la imagen [12].

El procesamiento digital de imágenes consta de una serie de elementos que en conjunto, permite ver, procesar, analizar las distintas imágenes digitales obtenidas del mundo real. En la Figura 2.1 se pueden ver los componentes de un sistema de procesamiento de imágenes, este inicia con un sensor para capturar los datos del mundo real para su posterior uso en una computadora o un hardware especializado para dicho procesamiento. Para poder realizar un correcto procesamiento se tienen que comprender las características de una imagen digital.

2.1.1. Características de una Imagen Digital

Una imagen digital es una representación matricial de las intensidades de luz que pueden ser capturadas desde un sensor CMOS (Complementary Metal - Oxide - Semiconductor) o CCD (Charged - Coupled Device) de una cámara. Las características de una imagen van desde el tamaño, así como, el número de colores usados o espacio de color. Existen también características que describen la información contenida en la imagen digital, estas mostrarán información como los bordes, líneas, puntos de interés, como cambios de intensidad, entre otros.

El tamaño de una imagen digital permite saber el número de píxeles contenidos en la

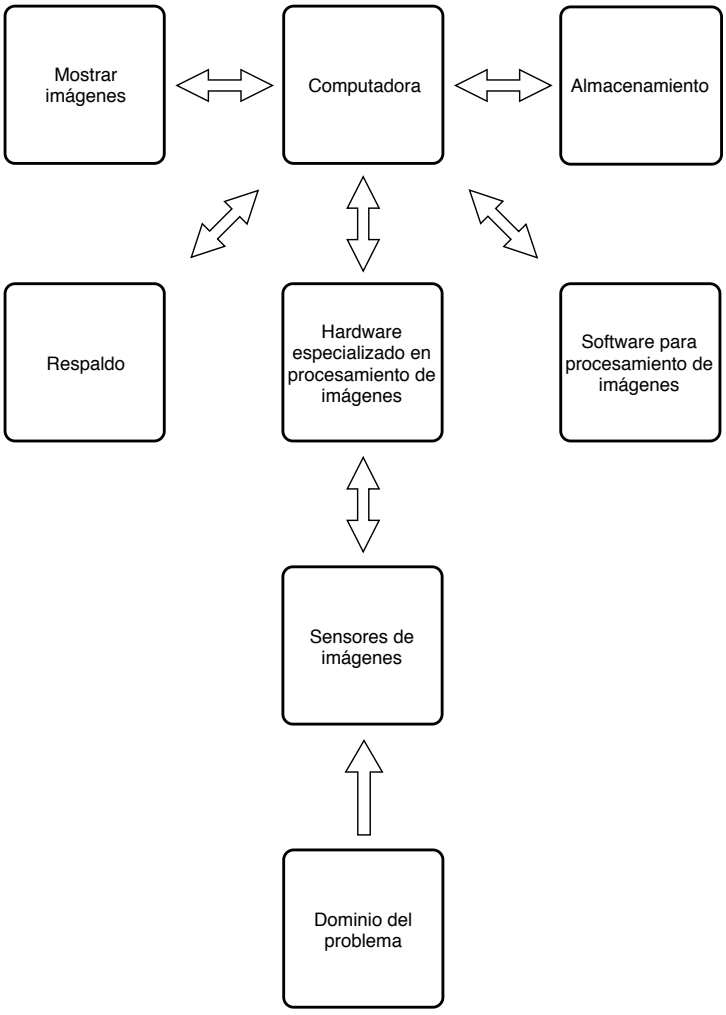


Figura 2.1: Componentes de un sistema general de procesamiento de imágenes [11].

imagen. Cuando se tiene un mayor número de píxeles, esta contiene más información que una con un menor número de píxeles. En la Figura 2.2 se presenta una imagen con dos tamaños diferentes, en 2.2a se visualiza la imagen en su tamaño original y en 2.2b se visualiza la imagen con un tamaño reducido, dejando en claro que a mayor cantidad de píxeles se puede representar de forma más fidedigna la información de la vida real capturada por el sensor.

Además del tamaño de una imagen, existe otro parámetro fundamental para el estudio de éstas. El espacio del color es el que define el número de canales o capas que contendrá una imagen en su interior, principalmente se hace uso del modelo RGB (Red, Green, Blue), este modelo permite a la imagen contener tres capas de colores como su nombre lo indica, una vez que se superponen estas tres capas generan lo que se conoce como una imagen a color. Una imagen que contenga una sola capa o canal, será una imagen en escala de grises, si es que cada píxel tiene un tamaño de ocho bits, ya que puede tomar un valor de 0 a 255 niveles

(a) 1836 x 3264 píxeles [13] (b) 46 x 82 píxeles

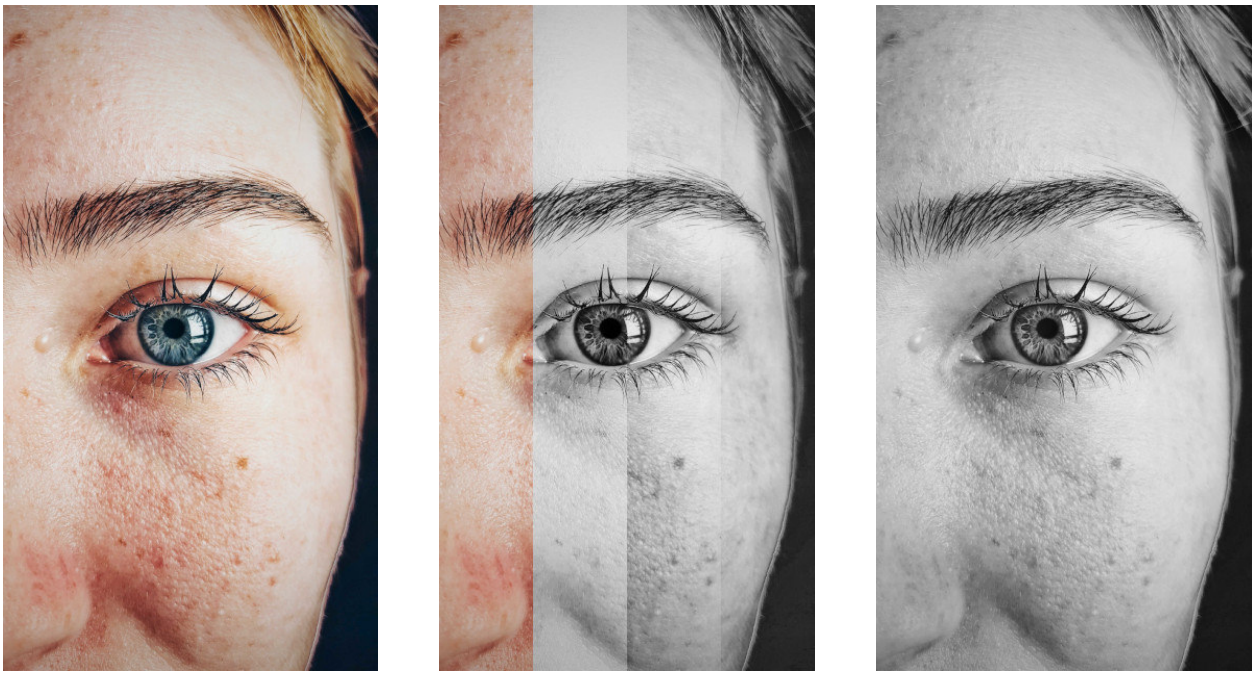
Figura 2.2: Ejemplo de una imagen con diferente tamaño.

de intensidad. Una imagen con un tamaño de píxel de un bit, es una imagen en blanco y negro, ya que cada píxel toma solamente valores binarios.

En la Figura 2.3 se muestra la separación de una imagen en el espacio de color RGB, a sus componentes, cada componente representa una matriz de píxeles que contienen una intensidad de 0 a 255, cuando se juntan los tres componentes se forma una imagen a color como se exhibe en la Figura 2.3a. De igual manera en la Figura 2.3c se puede ver una imagen en escala de grises obtenida usando un ajuste de canales $R = G = B$.

Las características de la imagen como su resolución, número de canales, entre otros, difieren de las características contenidas en la imagen digital. Estas últimas requieren un algoritmo de procesamiento de imagen para poder obtenerlas, estos algoritmos conocidos como descriptores de características procesan la imagen realzando puntos de interés que pueden ser líneas, puntos, entre otros.

Estos puntos de interés obtenidos por los descriptores de características son ampliamente usados en la actualidad, estas características alimentan sistemas de clasificación o detección de objetos en un vehículo autónomo para detectar el carril por el que circula, así como peatones u obstáculos a evitar. De igual manera en áreas como la industria usan descriptores de características para obtener información en la línea de producción, comprobando la calidad de sus productos. En áreas médicas igualmente usan estos descriptores para clasificar imágenes médicas con enfermedades que a un especialista médico le tomaría más tiempo segmentar que a un equipo computacional.



(a) Imagen Original [13]

(b) Imagen Original, Componente R, Componente G, Componente B

(c) Imagen en escala de grises

Figura 2.3: Ejemplo de una imagen a color, sus componentes y en escala de grises.

El tamaño de una imagen es importante, ya que a mayor número de píxeles es una mayor carga computacional, por lo cual, para procesarla se plantea el uso de un paradigma de la programación como lo es el procesamiento en paralelo, esta forma de estructurar la programación permite utilizar de forma eficiente los recursos computacionales existentes en un sistema de procesamiento de imágenes.

2.2. Convolución

La convolución es una forma matemática de combinar dos señales para formar una tercer señal [14]. La convolución de una imagen con una matriz de números reales puede ser usada para realzar bordes o suavizar una imagen dependiendo de la matriz utilizada [10].

Si A es una imagen de tamaño $M \times N$ y K es una matriz de convolución de tamaño $n \times n$, entonces B, la imagen convolucionada es calculada como se muestra en la ecuación 2.1 [10].

$$B_{y,x} = \sum_i \sum_j A_{y+i,x+j} K_{i,j} \tag{2.1}$$

Donde x y y representan la posición de la imagen A, cubierta por la ventana de convolución K; i y j se mueven a través de cada posición de K y el valor correspondiente en la imagen A, al finalizar, el tamaño de B es del mismo tamaño que A.

La convolución de una imagen es un filtrado que permite obtener las principales características para su posterior uso, esto es, que realza los datos importantes, dejando así, sólo la información que se usará para el entrenamiento de un clasificador.

Como se muestra en la Figura 2.4, en 2.4a se visualiza la imagen original, a la cual se le aplicó una matriz convolucional, dando como resultado las figuras 2.4b, 2.4c. En 2.4b, 2.4c se muestran las características que describen el contenido de la imagen, resaltando contornos y líneas de expresión que a simple vista no se logran percibir.

(a) Imagen Original [13] (b) Convolución Sobel Gy (c) Convolución Sobel Gx

Figura 2.4: Ejemplo de una imagen aplicando una matriz convolucional.

2.2.1. Matriz Convolucional

En el procesamiento de imágenes, la máscara, matriz o kernel de convolución es una pequeña matriz con un conjunto de ponderaciones que se aplica a los valores de cada píxel de una imagen, para crear o extraer un conjunto de características como la detección de bordes, aumentar el enfoque o la nitidez, entre otros [15].

En la Figura 2.5, se muestra la aplicación de una matriz convolucional a una imagen, teniendo en la parte inferior la representación de la imagen original, de la cual se substrae

temporalmente una matriz del mismo tamaño que el kernel convolucional llamada ventana de convolución (M), en la parte media se muestra la matriz de convolución (K) y en la parte superior se muestra la matriz de características, resultado de la convolución de la imagen. En la ecuación 2.2 se muestra el procedimiento para obtener el valor del píxel px al convolucionar la ventana de convolución con una matriz convolucional.

$$px = \sum_i M_i K_i \tag{2.2}$$

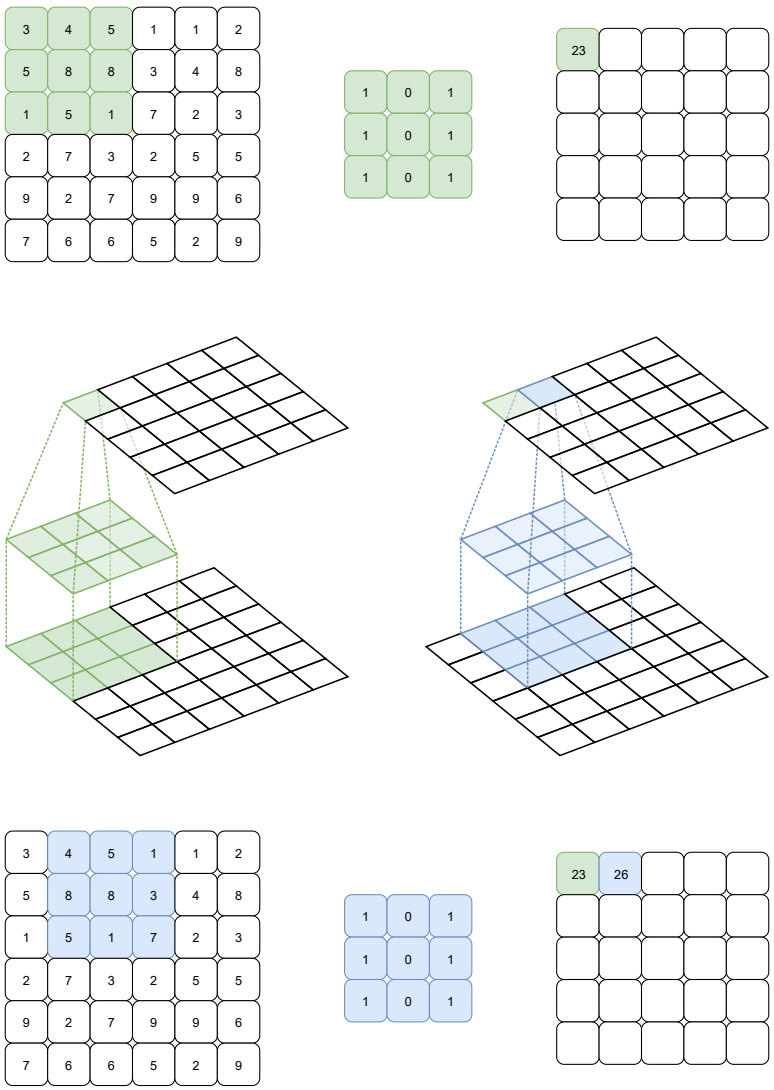


Figura 2.5: Aplicación de una matriz convolucional.

2.3. Red Neuronal Convolucional

Existen diferentes descriptores de características como pueden ser SIFT (Scale-Invariant Feature Transform) o basados en redes neuronales como puede ser una red neuronal convolucional. El 2014 Fischer *et al.* en [16] compararon el método SIFT con el método de red neuronal convolucional, pudieron concluir que las características obtenidas con el método de redes neuronales convolucionales claramente superaban con gran margen al método SIFT. Por esta conclusión es la decisión de usar métodos basados en redes neuronales convolucionales para este trabajo.

Una red neuronal convolucional es un tipo de red multicapa y que puede ser de aprendizaje profundo [17]. Una red neuronal convolucional es eficiente para el uso de reconocimiento de patrones y procesamiento digital de imágenes. En la Figura 2.6 se muestra un diagrama de una arquitectura general de una red neuronal convolucional, esta puede tener muchas más capas para su funcionamiento. La capa de convolución es la encargada de la extracción de las características que describen el objeto a clasificar o reconocer, por lo tanto este tipo de capa es de suma importancia para la implementación de este tipo de redes neuronales. La capa de agrupamiento se encarga de reducir la dimensionalidad de las imágenes o matrices de características, esto con la finalidad de convertir los datos de la imagen en información pura, eliminando datos que no importan en un clasificador, además de, al ser una matriz más pequeña, disminuir el tiempo computacional del algoritmo.

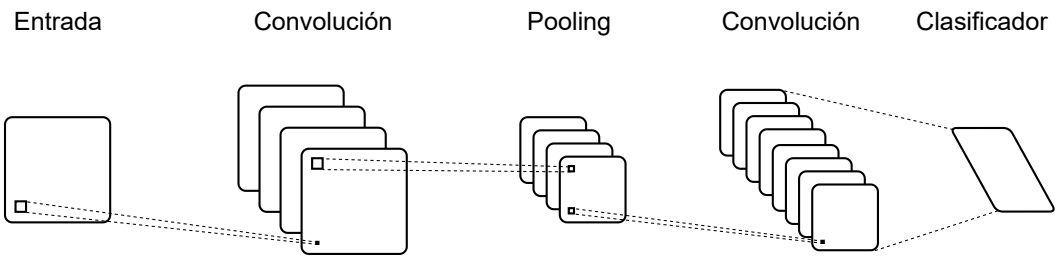


Figura 2.6: Diagrama general de una red neuronal convolucional.

2.4. Cómputo Paralelo

Las máquinas paralelas proveen una gran oportunidad para aplicaciones con grandes requerimientos computacionales [18]. Para realizar un trabajo en paralelo se debe tener en cuenta si se realizará en hardware dedicado o en software. Para hardware se tienen dispositivos dedicados como GPU(Graphics Processing Unit), plataformas multinúcleo, entre otros.

Para software se tienen lenguajes como C/C++, interfaz de programación de aplicaciones como OpenMP, CUDA, entre otros.

En la Figura 2.7 se muestra el proceso para el desarrollo de una aplicación para cómputo paralelo. El primer paso es definir la aplicación o problema a resolver. Después se diseña el algoritmo para resolver la problemática propuesta en el primer paso, aquí aún pueden usarse tareas secuenciales. Hasta el siguiente paso es donde se extraerá el paralelismo latente en el algoritmo implementado. Para después escoger el tipo de implementación del cómputo en paralelo, si será por hardware dedicado o software. Siendo por hardware dedicado se usan herramientas VLSI (Very Large-Scale Integration) para las tareas y la asignación de procesos. Y la implementación por software incluyen plataformas como OpenMP, CUDA, en lenguajes de programación C/C++, Java, entre otros.

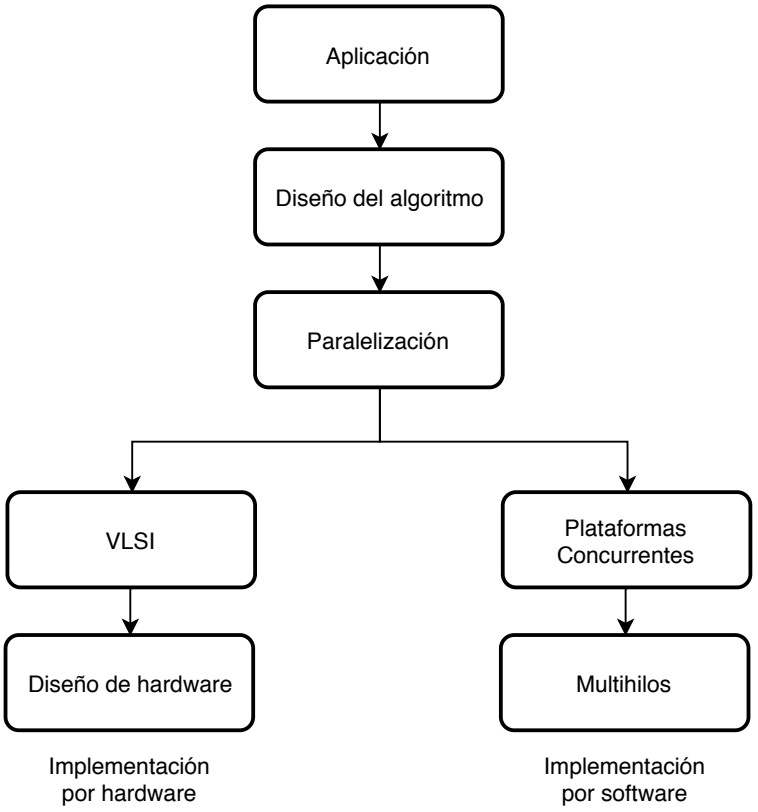


Figura 2.7: Fases de la implementación de una aplicación en software o hardware usando cómputo paralelo [19].

2.4.1. Algoritmo

Un algoritmo es un conjunto pre-escrito de reglas o procesos bien definidos para la solución de un problema en un número finito de pasos [20]. Las tareas o procesos de un algoritmo generalmente son independientes, aunque algunas pueden ejecutarse de forma concurrente para formar un algoritmo en paralelo [19].

Los componentes básicos que definen a un algoritmo son [19]:

1. Las tareas a procesar.
2. La dependencia entre las tareas donde la salida de una tarea es la entrada de otra tarea.
3. El listado de las entradas primarias necesarias para el algoritmo.
4. El listado de las salidas primarias producidas por el algoritmo.

Los algoritmos pueden ser clasificados basados en la dependencia de sus tareas [19]:

Algoritmo Serial (Serial Algorithm - SA)
Un algoritmo serial es aquél en donde las tareas deben ser ejecutadas en serie una detrás de otra debido a la dependencia de sus información entre tareas.

Algoritmo Paralelo (Parallel Algorithm - PA)
Es aquél en donde no hay dependencia en la ejecución de sus tareas, por lo cual cada una puede ejecutarse de forma concurrente.

Algoritmo Serial-Paralelo (Serial-Parallel Algorithms - SPAs)
Este tipo de algoritmo se encuentra con agrupaciones de tareas, las cuales estas pueden ser ejecutadas con concurrencia, pero las agrupaciones deben ser ejecutadas en forma secuencial.

2.4.2. Hilo (Thread)

Un proceso es una abstracción de un programa en ejecución [21], incluyendo los valores actuales del contador del programa, así como sus registros, variables y un espacio de direcciones. Un hilo es un miniproceso dentro de un proceso, que puede compartir el espacio de direcciones y todos sus datos entre ellos [21]. Un sistema operativo puede tener múltiples procesos, o programas ejecutándose de forma concurrente y a su vez, un proceso puede tener uno o más hilos en ejecución.

El término multihilo es una propiedad que permite la ejecución de múltiples hilos en el mismo proceso [21]. En nivel de hardware esta propiedad que un núcleo físico pueda ejecutar múltiples hilos simultáneos.

En una aplicación de ventana, el hilo principal es el encargado de actualizar la interfaz gráfica de usuario, por lo cual si se comienza una tarea que tarde mucho en ejecutarse, la interfaz se bloqueará, hasta que el hilo termine la ejecución y regrese a actualizar la interfaz de usuario. Para evitar este problema, se hace uso de un hilo adicional, para procesar en segundo plano la tarea mientras el hilo principal, sigue actualizando la interfaz de usuario [22].

2.4.3. Algoritmos Paralelos y Arquitecturas Paralelas

Paralelo (software) es una transferencia, ocurrencia o procesamiento simultáneo de las partes individuales de un todo [20]. De esta manera se puede entender que un algoritmo es paralelo cuando dos o más partes del mismo se pueden ejecutar independientemente de forma concurrente en hardware. Un multiprocesador es una arquitectura paralela en la cual puede ser efectuado procesamiento paralelo [20]. Entonces para hablar de un algoritmo paralelo se necesita de una arquitectura paralela para procesar la información de forma eficiente. Se necesita que el programador distribuya el trabajo entre los distintos núcleos de un sistema para mantener al procesador ocupado y trabajando de manera simultánea para resolver un problema.

Un programa en ejecución es llamado proceso. Un proceso es creado por el sistema operativo, por lo tanto recursos como memoria y registros son alojados para un proceso. Un hilo de un proceso comparte recursos que el proceso inició y reservó. Los hilos de un proceso pueden ejecutarse de manera simultánea en uno o varios núcleos/procesadores y trabajar de manera concurrente para ejecutar un programa [23].

El paralelismo puede ser implementado en diferentes niveles de un sistema computacional usando técnicas por hardware y software [19]:

Paralelismo en Nivel de Datos (Data-Level Parallelism - DLP)
Simultáneamente se opera en múltiples bits de datos, compartiendo la memoria entre los procesos en paralelo.

Paralelismo en Nivel de Instrucciones (Instruccion-Level Parallelism - ILP)
Se ejecutan diferentes instrucciones en el procesador de forma concurrente.

Paralelismo en Nivel de Hilos (Thread-Level Parallelism - TLP)
Un hilo (thread) es una porción de un programa que comparte recursos computacionales con otros hilos, en este caso múltiples hilos de proceso son ejecutados de forma paralela en uno o varios procesadores.

Paralelismo en Nivel de Procesos (Process-Level Parallelism - PLP)
Un proceso es un programa que está corriendo en un computador, el cual reserva sus propios recursos computacionales como pueden ser: memoria, registros, entre otros. Este es el caso clásico usado por un sistema operativo, donde varios programas están corriendo de forma simultánea en una o varias computadoras.

2.4.4. Gastos de Comunicación

Para procesos secuenciales o paralelos siempre hay la necesidad de leer o escribir datos a memoria [19]. Para sistemas paralelos existe la necesidad extra de comunicación entre procesos o hilos para el intercambio de información, ya sea para saber qué porcentaje del trabajo se ha realizado, hasta para saber si se ha concluido el mismo.

La comunicación entre procesos o hilos tiene diferentes problemas [19]:

Colisiones de Memoria
Este problema ocurre cuando dos o más procesos o hilos tratan de acceder ya sea para lectura o escritura al mismo espacio de memoria.

Ancho de Banda de la Memoria
Sin importar el tamaño de la memoria, el acceso a ella siempre es de forma secuencial usando un movimiento de una palabra a la vez, en escritura o lectura.

Pared de Memoria
La velocidad de transferencia de la memoria es más lenta que la velocidad de procesamiento.

2.4.5. Mecanismos de Sincronización

Al realizar la comunicación entre hilos de procesamiento se tiene como objetivo minimizar los gastos de comunicación, al igual evitar las condiciones de carrera, en la cual dos o más hilos leen y escriben datos compartidos, los cuales se pueden sobrescribir si no se protege la información de manera adecuada.

Sección Crítica

Para evitar una condición de carrera es necesario una exclusión mutua, para asegurar que si un proceso está utilizando una variable compartida, los demás hilos o procesos se excluirán

de hacer lo mismo [21]. Esta parte del programa en que se accede a la memoria compartida solamente por un hilo de procesamiento a la vez, se le conoce como sección crítica.

En la Figura 2.8, se muestra el comportamiento de una sección crítica para la implementación de una exclusión mutua, necesaria para la comunicación entre hilos de procesamiento, evitando así, la escritura o lectura simultánea en el mismo espacio de memoria. Cuando el hilo A entra en su región crítica, no permite que nadie más entre a la misma, es hasta que el hilo A, sale de la sección, el hilo B puede entrar a su región crítica.

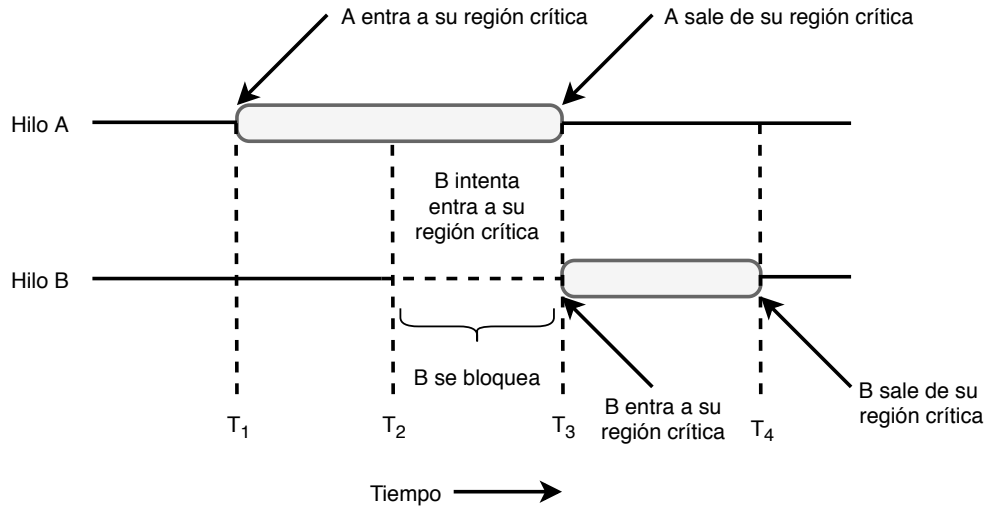


Figura 2.8: Exclusión mutua de hilos de procesamiento mediante el uso de secciones críticas [21].

Semáforo

Un semáforo es un tipo de dato que permite el acceso a recursos compartidos en un entorno paralelo, puede tener un valor de 0 o algún valor positivo, indicando que no se guardaron señales de despertar o si están pendientes una o más señales de despertar respectivamente [21].

Generalmente cuentan con dos señales, *down* y *up*. La operación *down* verifica si el valor es mayor a 0, de ser así, disminuye este valor y continua. Si es 0, el proceso se pone a dormir sin completar la operación. Las acciones de comprobar el valor, modificarlo y pasarlo a dormir se realizan de forma indivisible con una acción atómica [21]. La operación *up*, incrementa el valor del semáforo.

De esta forma se tiene una forma de sincronización distinto a la exclusión mutua. Los semáforos vacíos y llenos se necesitan para que ciertas secuencias de eventos ocurran o no, teniendo un mínimo y un máximo de operaciones a realizar.

Mutex

Cuando no es necesaria la habilidad de llevar un contador como lo hace un semáforo, se implementa una versión simplificada del mismo, llamada mutex [21].

Un mutex es una variable que tiene dos estados, desbloqueado o bloqueado. Por lo tanto un mutex puede controlar el acceso a una región crítica y dependiendo de si su estado es abierto o cerrado, permite a un hilo de ejecución entrar o no a la región crítica.

Los mutexes son buenos para administrar regiones mutuamente exclusivas, como una región crítica mostrada en la Figura 2.8. La principal diferencia es que una sección crítica se ejecuta como un objeto de modo usuario, lo cual quiere decir que no puede compartir su información entre procesos. Un mutex puede tener un nombre, por lo tanto además de sincronizar hilos, también puede servir para la sincronización entre procesos de sistema o programas.

2.4.6. Factor de Aceleración

El principal beneficio del cómputo paralelo es medido por el tiempo que tarda en completar una tarea en un proceso simple contra el tiempo que tarda en completar la misma tarea en N procesos paralelos [19].

El factor de aceleración $S(N)$ debido al uso de N procesos paralelos está definido en la ecuación 2.3, donde $T_p(1)$ es el tiempo que tomó al algoritmo terminar la tarea en un solo proceso y $T_p(N)$ es el tiempo de cómputo para la misma tarea en forma paralela.

$$S(N) = \frac{T_p(1)}{T_p(N)} \tag{2.3}$$

En una situación ideal, con un algoritmo totalmente paralelizable y cuando los tiempos de comunicación entre procesos y memoria es nulo, tenemos que $T_p(N) = T_p(1)/N$ con lo cual obtenemos la ecuación 2.4.

$$S(N) = N \tag{2.4}$$

2.4.7. Ley de Amdahl

La ley de Amdahl especifica el factor de aceleración esperado teniendo en cuenta que una fracción del algoritmo es paralizable f y una porción de ese algoritmo se ejecuta de forma secuencial $1 - f$, se obtiene la ecuación 2.5, donde el factor de aceleración es posible gracias a que la parte paralelizable del algoritmo es ahora distribuida en N procesadores [19].

$$S(N) = \frac{1}{(1-f) + f/N} \tag{2.5}$$

En la Figura 2.9 se muestra el factor de aceleración contra la porción paralelizable f para diferentes valores de N, obtenido de la ecuación 2.5.

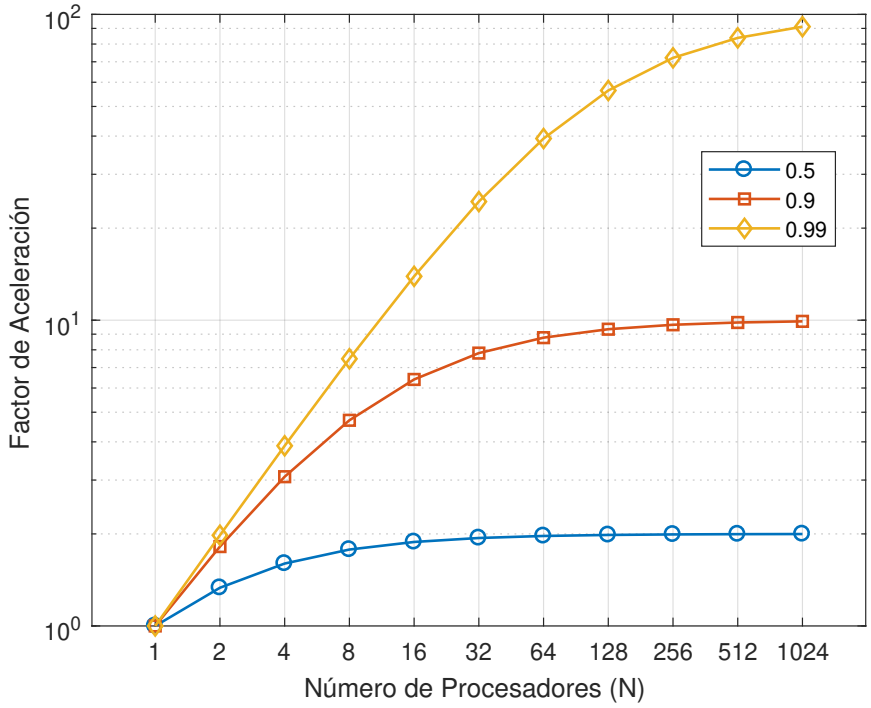


Figura 2.9: Factor de aceleración de acuerdo a la ley de Amdahl, para parámetros de $f = 0.5$, $f = 0.9$, $f = 0.99$.

Para valores grandes de N, el factor de aceleración en la ecuación 2.5 es aproximado como se muestra en la ecuación 2.6.

$$S(N) \approx \frac{1}{1-f} \tag{2.6}$$

Este resultado indica, observando en la Figura 2.9 que después de 10 hilos o procesos,

cualquier incremento en el factor de aceleración está directamente relacionado con descubrir las partes paralelizables del programa y en cómo ejecutar concurrentemente esas partes.

2.4.8. Ley de Gustafson-Barsis

La ley de Gustafson-Barsis hace la observación de que el paralelismo de un algoritmo aumenta cuando el tamaño del problema aumenta [19]. De igual manera se tienen N procesos, dejando como resultado la ecuación 2.7.

$$S(N) = 1 + (N - 1)f \tag{2.7}$$

En la Figura 2.10 se muestra el factor de aceleración de la porción paralelizable f con diferentes valores de N, obtenido de la ecuación 2.7.

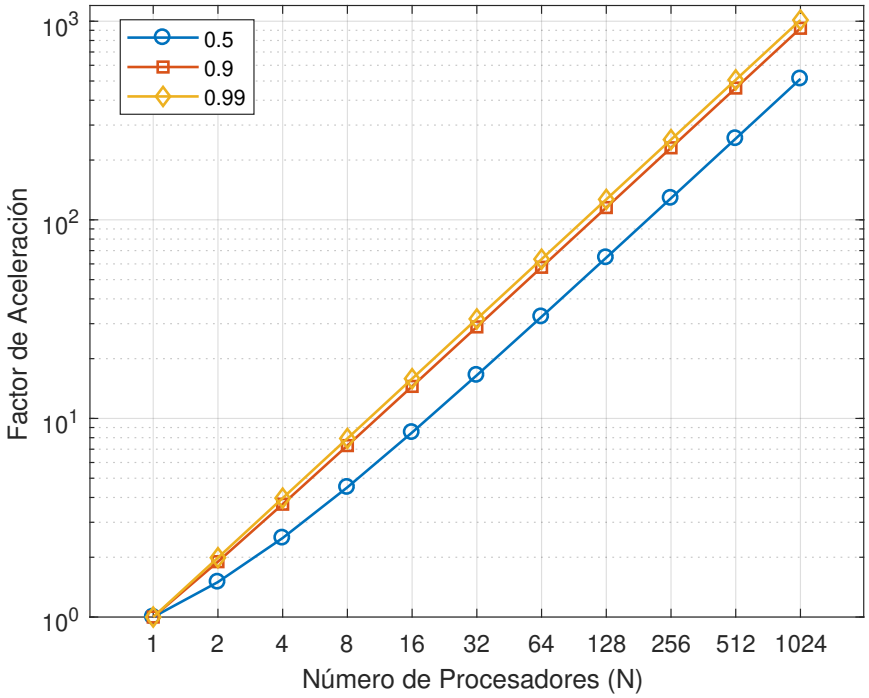


Figura 2.10: Factor de aceleración de acuerdo a la ley de Gustafson-Barsis, para parámetros de $f = 0.5, f = 0.9, f = 0.99$.

Se observa en la Figura 2.10 que se puede obtener un gran factor de aceleración inclusive con valores pequeños de f, cuando N tiende a un número grande.

En este capítulo se establecieron las bases teóricas del procesamiento digital de imágenes, así como una introducción al paradigma de la programación en paralelo, observando los

mecanismos de sincronización y la forma correcta del uso del cómputo concurrente. En el siguiente capítulo se describe la metodología propuesta para el diseño e implementación del presente trabajo.

Capítulo 3

Metodología

En este capítulo se presenta la metodología elegida para la extracción concurrente de características en imágenes digitales, usando programación con llamadas de bajo nivel para algoritmos en paralelo. Se presentan los diagramas de flujo de trabajo general y específicos para cada caso de estudio propuesto en el presente trabajo.

3.1. Metodología Propuesta

El procesamiento digital de imágenes es una tarea compleja que requiere de grandes requisitos computacionales, tanto de software como de hardware. En el capítulo anterior se describió la teoría necesaria para el desarrollo e implementación de un sistema de cómputo paralelo para el óptimo procesamiento para la extracción de características de imágenes digitales usando una convolución en dos dimensiones.

En la Figura 3.1 se muestra un diagrama de flujo general para la carga de trabajo del sistema.

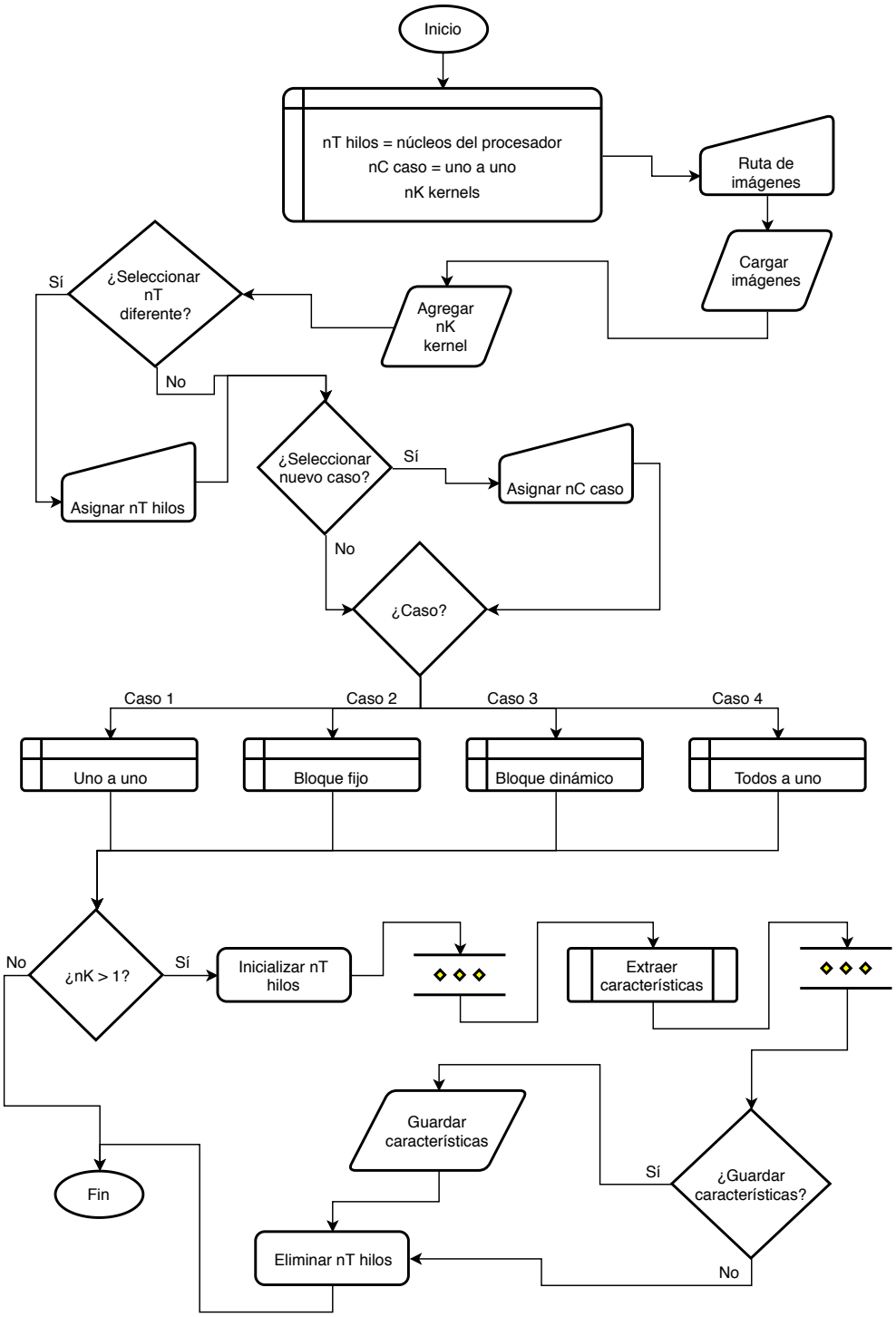


Figura 3.1: Diagrama general propuesto para la extracción concurrente de las características visuales de imágenes digitales.

El proceso inicia con el llamado autónomo del constructor de la clase implementada, el cual define los parámetros por default del proceso. Este constructor define tres variables que el sistema usa para la correcta extracción de características:

nT hilos de procesamiento
: A esta variable se le asignará automáticamente el número máximo de núcleos físicos o lógicos con los que cuente el procesador a ejecutar el algoritmo.

nC caso
: Variable donde se asigna la forma de repartir el trabajo concurrente entre los nT hilos de procesamiento.

nK kernels
: El sistema podrá procesar n convoluciones en dos dimensiones, como sean agregados nK matrices o kernels de convolución, por cada nK matriz de convolución agregada al sistema, se tendrá como resultado una matriz de características resultante.

Una vez definidos los parámetros base del sistema, se le solicita al usuario introducir una ruta donde se encuentren guardadas las imágenes a procesar, con esta ruta el sistema cargará a memoria las imágenes para su posterior procesado, el sistema cargará automáticamente imágenes del tipo JPEG/JPG y PNG que se encuentren en la ruta. Después de la carga a memoria, se solicitará que se agreguen nK matrices de convolución al sistema.

Después de agregar los kernels al sistema, se podrán cambiar las variables predefinidas, para poder cambiar manualmente el caso de procesamiento en caso de ser necesario, así como el número de hilos que se usará durante el proceso concurrente.

Una vez definidos todos los parámetros, además de tener en memoria las imágenes a procesar, se inicializan los nT hilos seleccionados, teniendo en cuenta la variable nC para la correcta inicialización de los hilos, los cuales comenzarán a procesar las imágenes con las matrices de convolución previamente cargadas. Todo este procesamiento se lleva de manera concurrente, siendo invisible para el usuario, el cual una vez terminado el procesamiento podrá elegir si guardar las características obtenidas de dicho procesamiento o terminar la ejecución del programa. Para guardar las características al disco duro de la computadora es necesario asignar una ruta para establecer dónde se guardarán estas.

El mismo programa de forma automática elimina los hilos usados, verificando que los mismos hayan terminado su trabajo, además de borrar correctamente sus manejadores y liberando los recursos usados, evitando así desperdicio de memoria y recursos computacionales.

El diagrama general cuenta con un bloque de procesamiento predefinido, llamado “Extraer características”, este bloque se procesa de forma concurrente con todos los hilos seleccionados en la variable nT, en esta parte es donde se utiliza la variable nC, ya que esta

repartirá el trabajo entre los hilos de ejecución de forma diferente. Para esto se cuenta con cuatro métodos de procesamiento concurrente para su estudio:

- Método 1: Uno a uno.
- Método 2: Bloque fijo.
- Método 3: Bloque dinámico.
- Método 4: Todos a uno.

3.2. Método 1: Uno a uno

El primer Método propuesto es llamado "uno a uno". Este método de estudio separa el flujo de trabajo a los hilos de ejecución de forma singular, esto quiere decir que cada hilo inicializado procesará una imagen digital. Cada que un hilo de trabajo termine de procesar o de extraer las características de una imagen, pedirá a memoria la siguiente, así hasta terminar el total de las imágenes cargadas en memoria.

En la Figura 3.2, se muestra un diagrama de flujo para la implementación del método de estudio "uno a uno". En este se expone el flujo a seguir para la separación del trabajo, que es lo mismo, la forma en que cada hilo procesará las imágenes digitales cargadas en memoria. Se cuenta con una variable general que compartirán todos los hilos de ejecución, además de una variable local para cada hilo, donde mediante una sección crítica se asignará la variable general a la local, además de aumentar el contador. Este contador verificará que no se sobrepase el número máximo de imágenes en memoria.

Después de la verificación del contador, cuando no sobrepase el número de imágenes, cada hilo comenzará a procesar una imagen nK veces, esto es, el número de matrices de convolución existentes en el sistema, guardando en un vector en memoria las características extraídas de cada imagen por matriz convolucional.

Cuando el contador $nImg$ exceda el número de imágenes en memoria, se procederá a salir del hilo, el cual quedará en espera para su correcta liberación.

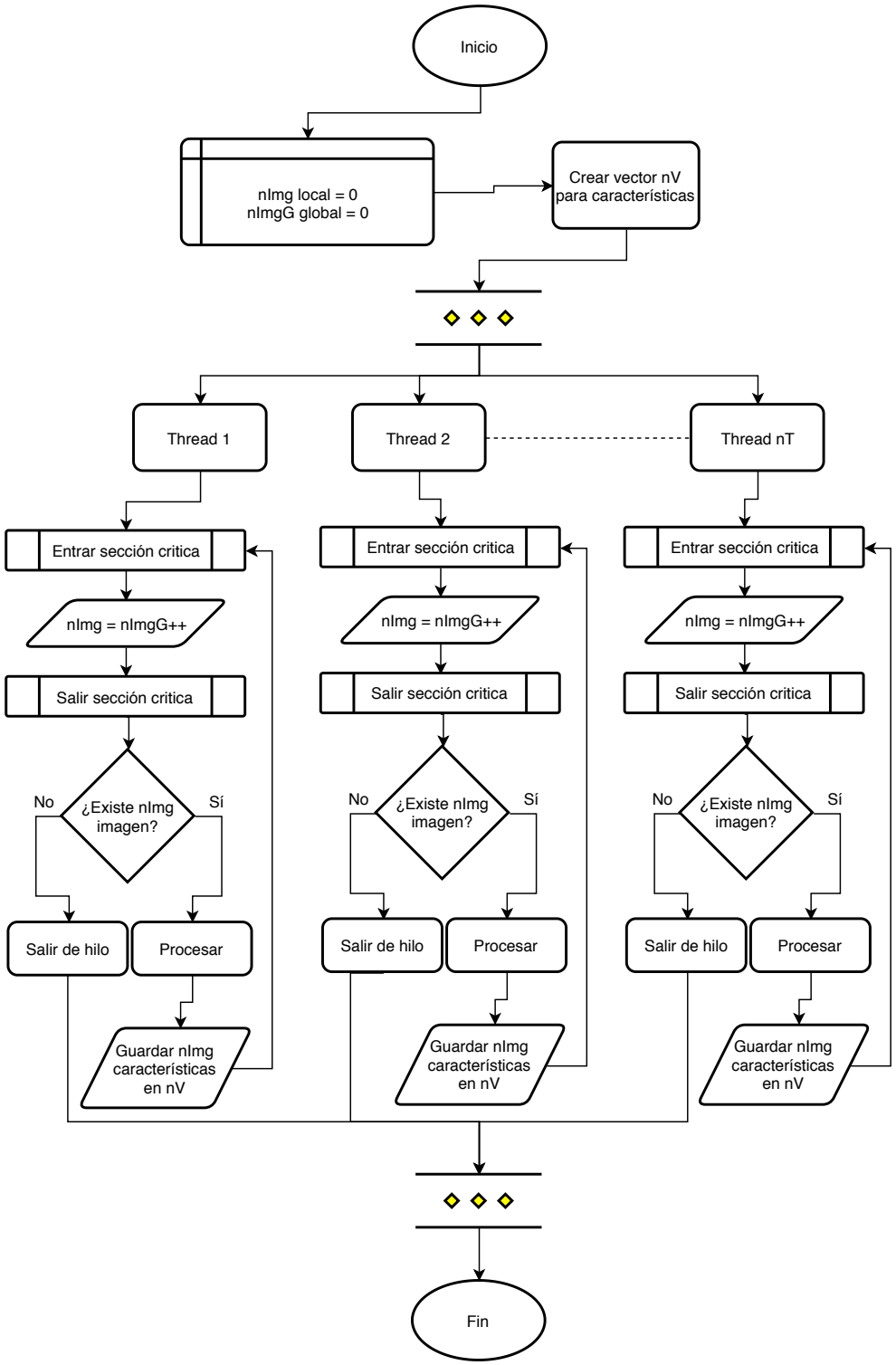


Figura 3.2: Diagrama de flujo para el proceso usando la distribución de trabajo “uno a uno”.

3.3. Método 2: Bloque fijo

El siguiente método de estudio es llamado "bloque fijo". En este procedimiento se separa el trabajo en forma de bloques, de esta manera cada hilo de ejecución extraerá las características a un conjunto fijo predefinido de imágenes. Cuando el hilo de proceso termine con un bloque de imágenes, pedirá el siguiente bloque para procesar, en caso de exceder el número total de imágenes, se calcula el número de imágenes restantes a procesar.

En la Figura 3.3 se muestra el proceso que lleva a la extracción de las imágenes digitales usando por este caso de estudio. Se cuenta con dos variables a usar en una sección crítica para controlar que imágenes debe procesar cada hilo de ejecución, además de una variable fija para el segmento a procesar.

De igual manera que en el caso anterior, cada hilo procesará el bloque fijo de imágenes, extrayendo nK características de cada imagen contenida en el bloque fijo asignado a cada hilo de procesamiento concurrente.

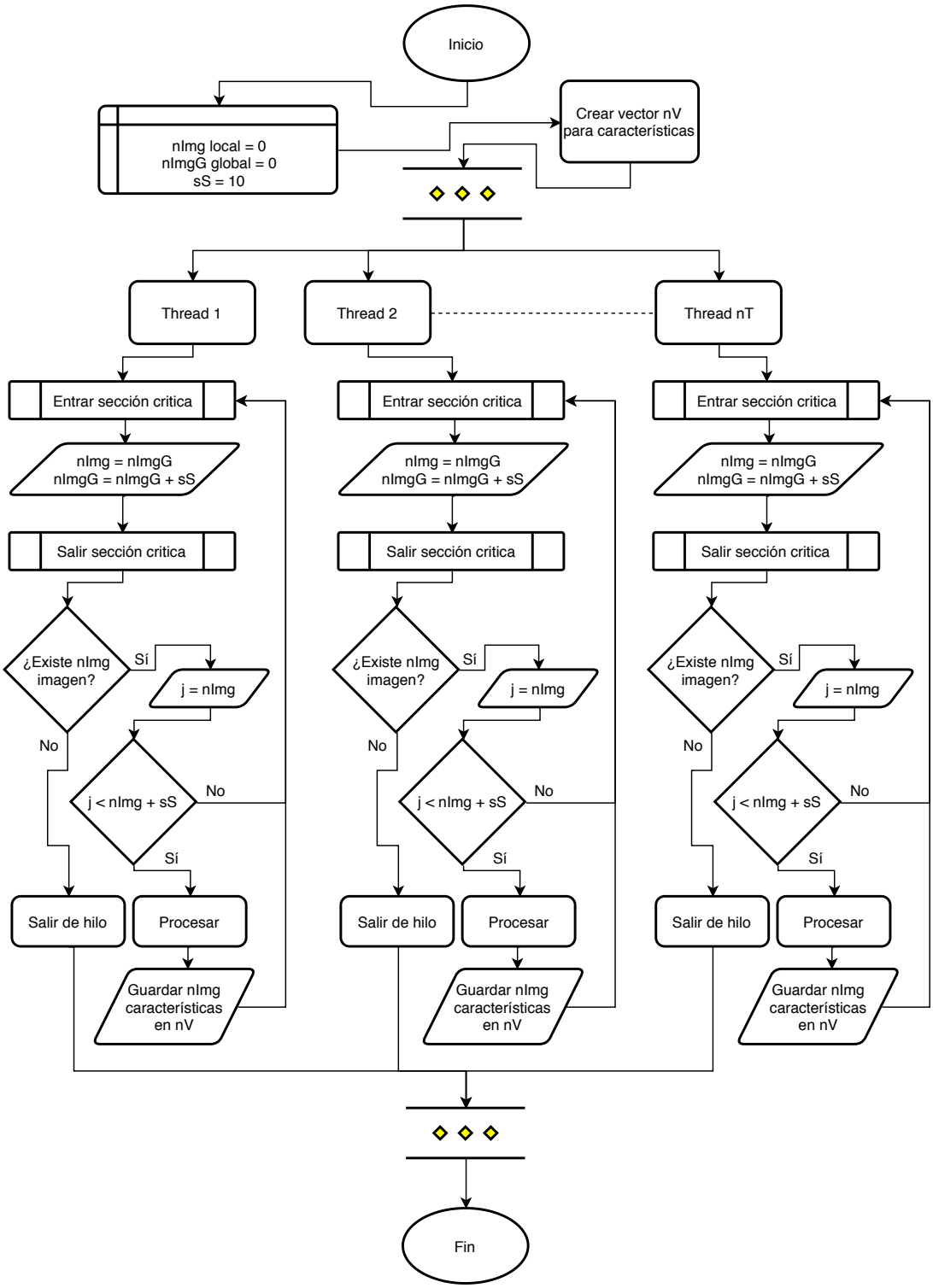


Figura 3.3: Diagrama de flujo para el proceso usando la distribución de trabajo “bloque fijo”.

3.4. Método 3: Bloque dinámico

El método de estudio "bloque dinámico" es muy parecido al caso anterior, divide el flujo de trabajo en bloques, los cuales serán procesados por nT hilos de procesamiento. La principal diferencia es que los bloques ahora serán dinámicos, lo que conlleva a teóricamente una menor colisión en lecturas de memoria, lo que en teoría permite alcanzar un mejor desempeño computacional al extraer las características. Para lograr una separación uniforme del trabajo, se obtiene el número total de imágenes a procesar, esto entre el número de hilos nT, da como resultado una división equilibrada de trabajo entre todos los hilos de procesamiento como se muestra en la ecuación 3.1.

$$Carga\ de\ trabajo = \frac{nImg}{nT} \tag{3.1}$$

A partir de 3.1 se obtiene el flujo de trabajo que seguirá cada hilo de procesamiento. Si se cuentan con 100 imágenes y 4 hilos de procesamiento, la carga de trabajo para cada hilo será de 25 imágenes, lo que quiere decir, que cada hilo de procesamiento extraerá las características de 25 imágenes, además de nK, matrices de convolución para cada imagen, todo de forma concurrente.

En la Figura 3.4 se puede observar el flujo de trabajo adoptado por este caso de estudio, observando que la extracción de las características se hace de forma concurrente con todos los hilos de ejecución disponibles, limitado al hardware utilizado en el programa.

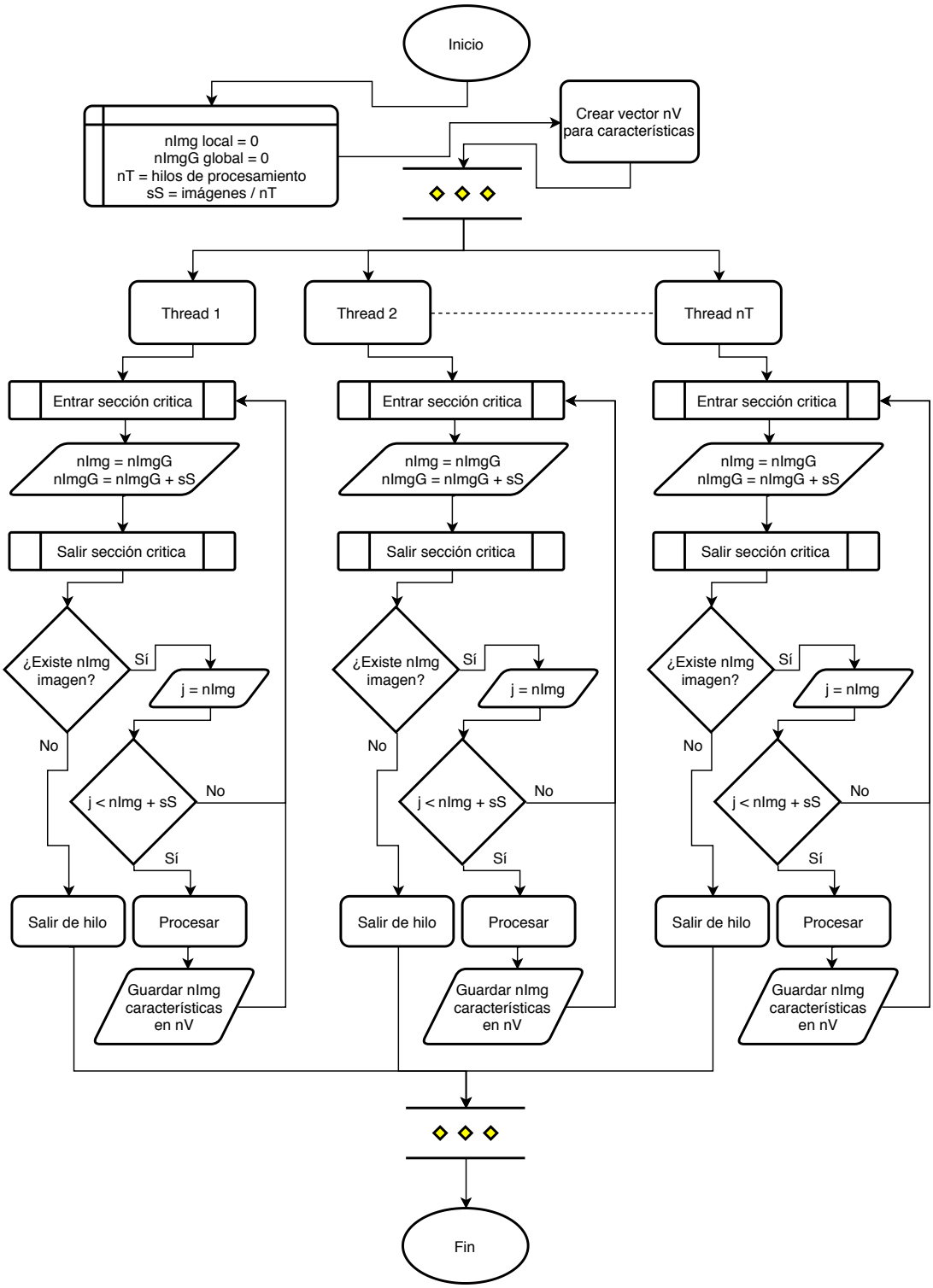


Figura 3.4: Diagrama de flujo para el proceso usando la distribución de trabajo "bloque dinámico".

3.5. Método 4: Todos a uno

El último método de estudio propuesto es llamado "todos a uno". En este caso la división de trabajo será por imagen, obteniendo la altura de esta y dividiendo esa misma altura entre el número de hilos para encontrar la sección correspondiente de la imagen que cada hilo procesará de forma concurrente.

Para obtener el área de trabajo de cada hilo es necesario contar con variables definidas, las cuales serán el número de hilos total inicializados, además del número de hilo actualmente procesando. Una variable delta, para calcular la parte a procesar como se ve en la Figura 3.5. Donde cT es el hilo de proceso actual, nT el número total de hilos de procesamiento, H la altura de la imagen a procesar. De esta manera se encuentra el inicio y el final de la sección que cada hilo procesará de la imagen, obteniendo así la totalidad de la imagen a procesar en partes iguales por cada hilo de procesamiento.

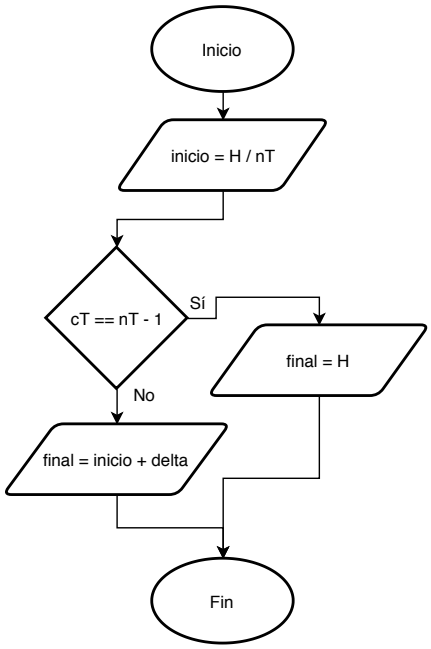


Figura 3.5: Obtención de la distribución delta de trabajo.

En la Figura 3.6 se muestra el flujo de trabajo para realizar la extracción de características usando el caso de estudio "todos a uno". Se puede ver la forma de repartir el trabajo entre todos los hilos de ejecución, que en este caso, todos trabajarán al mismo tiempo sobre la misma imagen.

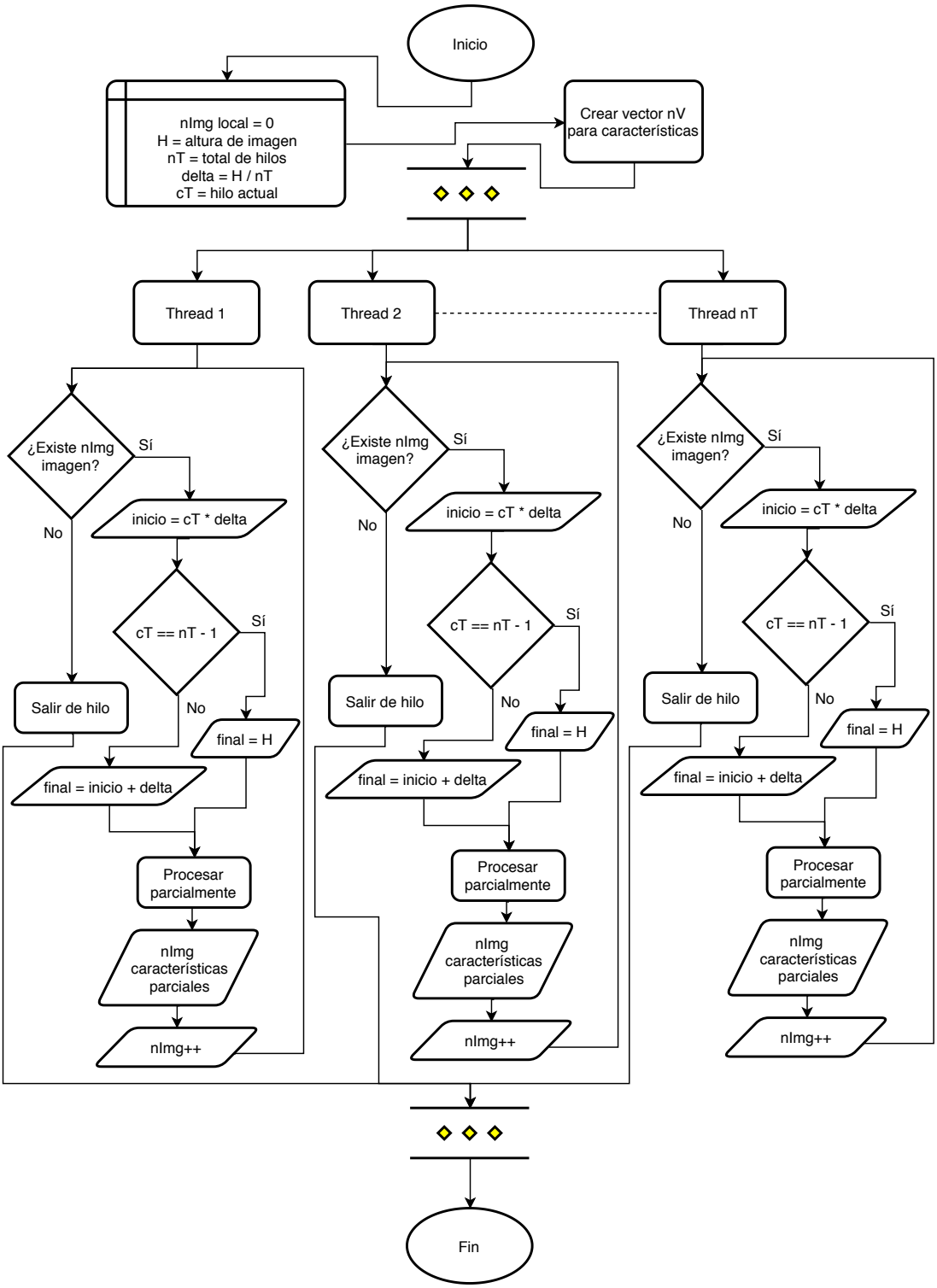


Figura 3.6: Diagrama de flujo para el proceso usando la distribución de trabajo “todos a uno”.

3.6. Desarrollo de algoritmos

Para el desarrollo de la metodología propuesta en el capítulo, se eligió el lenguaje de programación C++, haciendo uso del estándar ISO/IEC 14882:2017 [24] para el manejo de ficheros y archivos y el estándar ISO/IEC 14882:2011 [25] para el manejo de hilos de procesamiento. Se eligieron estos estándares ya que proveen las herramientas necesarias para desarrollar los algoritmos utilizados en el presente trabajo. Un algoritmo permite desarrollar un reloj para medir el costo computacional de los algoritmos, siendo este de alta precisión. Además permite desarrollar un algoritmo para leer todos los ficheros de un directorio, necesario para la carga a memoria de las imágenes digitales.

Este estándar ISO/IEC 14882:2011 [25] provee también de las herramientas para desarrollar multihilos a bajo nivel, siendo este último y todos los algoritmos portables a nivel de sistema operativo, dejando como opción el poder compilar en diferentes sistemas operativos sin necesidad de cambiar gran parte del código.

En primer lugar se desarrolla una función dentro de un espacio de nombres que se muestra en Código 3.1 para obtener los núcleos físicos o lógicos del procesador donde se usará el sistema. Esta función regresa el valor deseado.

Código 3.1: Función estática para saber el número de núcleos físicos o lógicos de un procesador.

```
namespace MTLibrary
{
        // HW
        static size_t getNumberOfProcessors();
}
```

Para obtener el desempeño computacional de la extracción de características, se implementó una clase como se muestra en Código 3.2 para un reloj de alta resolución, el cual permite obtener el tiempo en milisegundos o segundos usado por un algoritmo.

Código 3.2: Clase para medir el tiempo de cómputo usado por un algoritmo.

```
namespace MTLibrary
{
        class Timer
        {
        public:
                Timer();
                ~Timer();
                void start();
                double getMilliSeconds();
                double getSeconds();
        private:
                using Clock = std::conditional_t<
                        std::chrono::high_resolution_clock::is_steady,
                        std::chrono::high_resolution_clock,
```

```
                        std::chrono::steady_clock>;
                Clock::time_point startC;
        };
}
```

Se desarrolló también una función estática dentro del espacio de nombres *MTLibrary*, la cual devuelve la ruta de todas las imágenes en formato PNG, JPEG y JPG, del directorio que recibe como argumento la función de tipo std::string. Esta función se puede observar en Código 3.3.

Código 3.3: Función para obtener los ficheros de un directorio.

```
namespace MTLibrary
{
        // Files
        static std::vector<std::string>
                getDirectoryImages(std::string _dir);
}
```

Para el trabajo principal se desarrolla una clase la cual se muestra en Código 3.4. Esta consta de funciones privadas para el uso interno del sistema y funciones públicas que permiten la interacción con el usuario. La función loadImages, permite cargar las imágenes a memoria de un directorio, internamente usa el Código 3.3.

Se tiene un constructor, el cual inicializa las variables privadas a un determinado estado, usando el máximo número de hilos, como se observa en Código 3.1, además del caso de separación del trabajo entre los hilos de procesamiento.

En el destructor se agrega el código necesario que detecta cuando los hilos de procesamiento terminan, por lo cual procede a eliminarlos correctamente haciendo uso de la función privada deleteThreadPool, evitando desperdicios de recursos computacionales.

Se cuenta con una función para guardar en el disco las características extraídas de las imágenes digitales, dando como opción la ruta dónde guardarlas. Además de una función para reportar el progreso del trabajo de la librería al usuario.

Se tiene una función para una convolución en dos dimensiones, implementada a partir de la ecuación 2.1. Esta función recibe la imagen original y otra como referencia para guardar las características extraídas, también recibe un conjunto de matrices convolucionales y un parámetro opcional para normalizar o no estas matrices. Esta misma función consta de una sobrecarga para aceptar dos parámetros adicionales, los cuales permiten procesar sólo una región de la imagen.

La clase contiene dos propiedades, que permiten modificar el caso de procesamiento además de los hilos usados por la librería. Y tiene una función, con la cual permitirá al usuario procesar las imágenes en memoria.

Se tienen 4 funciones privadas y responsables de dividir el trabajo entre los hilos de procesamiento. Todas las funciones C1, C2, C3 y C4 reciben un vector con las imágenes originales, además de otro vector donde se guardarán en memoria las características extraídas de las imágenes. Reciben de igual manera un vector con las matrices convolucionales a usar, junto a una variable para medir el tiempo computacional usado.

La función C1 hace uso del método "uno a uno", el cual se puede observar su flujo de trabajo en la Figura 3.2. Además recibe una variable del tipo atomic, para el aumento del contador general, el cual comunica el índice de la imagen a procesar entre los hilos inicializados.

En la función C2, se recibe además una sección crítica para proteger el acceso a la variable global al aumentar el segmento de imágenes a procesar. Este flujo de trabajo es llamado "bloque fijo" y se puede ver su operación en la Figura 3.3.

La función C3 opera bajo el flujo de trabajo mostrado en la Figura 3.4, la cual recibe como parámetros adicionales el inicio y el final del segmento que procesarán, este segmento se calcula usando la ecuación 3.1, donde es necesario conocer el total de imágenes a procesar y el total de hilos usados, para determinar el "bloque dinámico" que procesará cada hilo de trabajo.

Al final se muestra la función C4, basada en la Figura 3.6. Esta división de trabajo llamada "todos a uno" permite procesar una imagen con el total de los hilos inicializados. Para determinar esa carga de trabajo es necesario conocer el total de hilos usados, más el hilo que actualmente procesa una parte, junto a la altura de la imagen a procesar. De esta manera se determina qué parte de la imagen procesará cada hilo, pudiendo así, procesar cada imagen con todos los hilos disponibles de manera concurrente.

Código 3.4: Clase para la extracción de características de una imagen digital usando programación en paralelo.

```
namespace MTLibrary
{
        class ConvLayer
        {
        public:
                ConvLayer();
                ~ConvLayer();
                std::vector<float> printProcessingInfo();
                bool loadImages(const std::string & _path);
                bool saveFeatures(const std::string & _path);
                void addKernel(std::vector<float> _k);
                void addKernel(size_t _kernelSize, size_t _kernelType);
                void cleanKernelList();
                static void printProgressBar(size_t _iter, size_t _max);
                static void convolution(cv::Mat _img,
                        cv::Mat & _conv,
                        std::vector<float> _kernel,
                        bool _normalizedKernel = true);
```

```
            static void convolution(cv::Mat _img,
                    cv::Mat & _conv,
                    std::vector<float> _kernel,
                    size_t _first,
                    size_t _last,
                    bool _normalizedKernel = true);
            void _setCase(size_t _value);
            size_t _getCase();
            __declspec(property(get = _getCase, put = _setCase))
                    size_t processingCase;
            void _setThreads(size_t _value);
            size_t _getThreads();
            __declspec(property(get = _getThreads, put = _setThreads))
                    size_t processingThreads;
            void process();
    private:
            void deleteThreadPool();
            static void C1(
                    std::vector<cv::Mat> & _images,
                    std::vector<std::vector<cv::Mat> > & _convImages,
                    std::vector<std::vector<float> > & _kernels,
                    std::atomic<size_t> & _nImg,
                    float & _time
            );
            static void C2(
                    std::vector<cv::Mat> & _images,
                    std::vector<std::vector<cv::Mat> > & _convImages,
                    std::vector<std::vector<float> > & _kernels,
                    std::atomic<size_t> & _nImg,
                    std::mutex & _mx,
                    float & _time
            );
            static void C3(
                    std::vector<cv::Mat> & _images,
                    std::vector<std::vector<cv::Mat> > & _convImages,
                    std::vector<std::vector<float> > & _kernels,
                    float & _time,
                    size_t _first,
                    size_t _last
            );
            static void C4(
                    std::vector<cv::Mat> & _images,
                    std::vector<std::vector<cv::Mat> > & _convImages,
                    std::vector<std::vector<float> > & _kernels,
                    std::mutex & _mx,
                    size_t _nT,
                    size_t _cT,
                    float & _time
            );
            std::vector<std::string> imagesPaths;
            std::vector<cv::Mat> images;
            std::vector<std::vector<cv::Mat> > convImages;
            std::vector<std::vector<float> > kernels;
            std::vector<std::thread> threads;
```

```
                MTLibrary::Timer timer;
                size_t caseP;
                size_t threadsP;
                std::atomic<size_t> nImg;
                std::mutex mx;
                float time;
                size_t kSize;
        };
}
```

En este capítulo se mostró el desarrollo de los métodos propuestos, desde el flujo de trabajo general, hasta el flujo individual de casa caso propuesto. Además se muestra el módulo implementado, que servirá para la extracción de características en forma concurrente. En el siguiente capítulo, se describe el equipo usado para las pruebas de rendimiento, comparando el módulo propio, contra librerías existentes como OpenCV para las características extraídas y OpenCL, OpenMP para comparar el rendimiento del procesamiento.

Capítulo 4

Pruebas y Resultados

En este capítulo se presentarán las pruebas y resultados obtenidos del presente trabajo. Se dividen las pruebas en dos partes, la primera consta de la visualización y comparación de las características extraídas. La segunda muestra la comparación en tiempo y factor de aceleración de los métodos propuestos contra librerías existentes actualmente.

4.1. Extracción de Características

Para la extracción de características se compara el algoritmo propuesto, implementando la ecuación 2.1, usando un kernel de convolución Laplaciano de tamaño 3×3. El proceso de convolución convierte los datos de la imagen en información útil, o lo que es lo mismo, las características visuales que ofrece la imagen, en este caso, el realce de los bordes. A su vez, se compara usando la misma matriz de convolución pero ahora usando el algoritmo convolutivo de la librería OpenCV.

En la Figura 4.1 se muestran las características visuales extraídas usando el método propuesto y el algoritmo de OpenCV. Además de la extracción se aplica un algoritmo de negativo para una mejor apreciación de las características en una hoja blanca. En esta misma Figura 4.1 se observa una mejor detección cualitativa de los bordes de la imagen original usando el algoritmo propuesto en este trabajo de tesis.

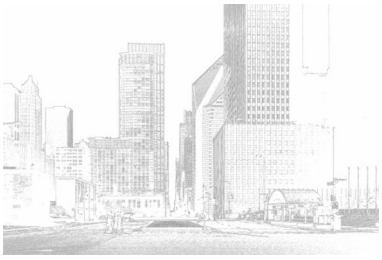
(a) Imagen original [26]

(b) Características extraídas usando OpenCV

(c) Características extraídas usando el método propuesto

(d) Imagen original [27]

(e) Características extraídas usando OpenCV



(f) Características extraídas usando el método propuesto

(g) Imagen original [28]

(h) Características extraídas usando OpenCV

(i) Características extraídas usando el método propuesto

(j) Imagen original [29]

(k) Características extraídas usando OpenCV

(l) Características extraídas usando el método propuesto

(m) Imagen original [30]

(n) Características extraídas usando OpenCV

(ñ) Características extraídas usando el método propuesto

Figura 4.1: Características visuales extraídas usando el algoritmo propuesto y usando el algoritmo de OpenCV.

En la Tabla 4.1 se muestra el índice de similitud estructural (SSIM [31]). El SSIM puede tomar un valor desde -1 a 1. Cuando las dos imágenes comparadas son iguales, el valor de SSIM es 1, y cuando las imágenes son totalmente diferentes toma el valor de -1. Se puede observar que el valor SSIM calculado de las características obtenidas por el algoritmo propuesto para una convolución en dos dimensiones, obtiene un valor SSIM generalmente más alto que el valor obtenido por las características obtenidas de OpenCV, indicando una mejor detección del algoritmo propuesto comparado con el usado por OpenCV.

Tabla 4.1: Índice de similitud estructural de la imagen original respecto a sus características extraídas en OpenCV y con el método propuesto.

Figura	OpenCV	Método propuesto
	SSIM	
4.1a	0.2982	0.4056
4.1d	0.2045	0.4127
4.1g	0.6160	0.6401
4.1j	0.2247	0.2130
4.1m	0.2613	0.3235

En la Tabla 4.2 se muestra el SSIM obtenido de comparar las características obtenidas usando OpenCV y aquéllas obtenidas usando el método propuesto. Se observa un valor más alto que en la Tabla 4.1, ya que como esta es una comparación entre características, se espera que sean más parecidas entre éstas, que las características y su imagen original. El valor alto indica que ambas características tienen información parecida, validando la extracción de características de ambos algoritmos.

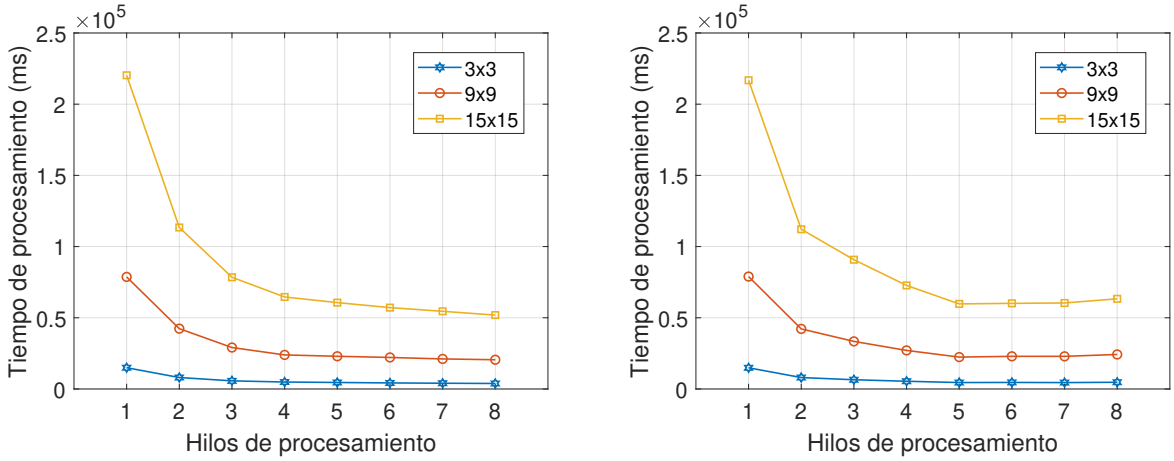
Tabla 4.2: Índice de similitud estructural de las características obtenidas por OpenCV y el método propuesto.

OpenCV	Método propuesto	SSIM
Figura		
4.1b	4.1c	0.6028
4.1e	4.1f	0.4971
4.1h	4.1i	0.8308
4.1k	4.1l	0.5840
4.1n	4.1ñ	0.7525

La métrica SSIM permite validar la obtención de características de la metodología propuesta. En la siguiente sección se muestra el análisis sobre los cuatro métodos propuestos y sus comparativas con librerías existentes.

4.2. Desempeño Computacional de los Métodos Propuestos

Para las pruebas de desempeño, se comparan los resultados en tiempo y el factor de aceleración de los métodos propuestos en este trabajo de tesis, comparando además contra OpenMP y OpenCL (CPU (Central Processing Unit) y GPU). Para todas las pruebas se utiliza un banco de 100 imágenes con resolución 4K (3840×2160 píxeles), además se usa como matriz convolucional un filtro suavizante de promedio. Los resultados fueron obtenidos usando de 1 a 8 hilos de procesamiento en un procesador Intel i7 7820HQ a 2.9 GHz, corriendo Windows pro (1809), además de memoria RAM DDR4 a 2400 MHz y un GPU NVIDIA QUADRO M1200 usando CUDA 10.1. Los algoritmos implementados permiten el uso de ventanas de convolución impares de tamaño $n \times n$. Además permiten el uso de imágenes de tamaño $M \times N$, para su procesado y extracción de características.



(a) Método 1

(b) Método 2

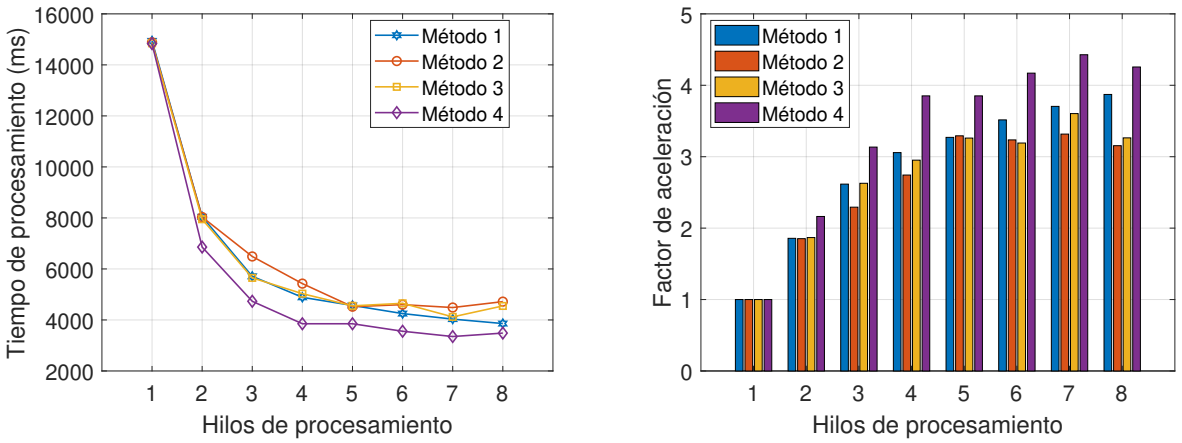
(c) Método 3

(d) Método 4

Figura 4.2: Tiempo de procesamiento de los métodos propuestos para diferentes tamaños de kernel.

En la Figura 4.2 se aprecian los tiempos de cómputo que tardan los métodos propuestos para procesar el banco de imágenes provistos para las pruebas. Se usan tamaños de ventana de 3×3, 9×9 y 15×15. Se observa también, que el tiempo computacional crece de manera significativa conforme la ventana de convolución aumenta. Debido a que el kernel de tamaño 3×3 es el que mejor resultados de desempeño arroja, en este se basarán las siguientes pruebas.

En la Figura 4.3 se muestra el desempeño computacional de todos los métodos propuestos, usando un kernel de tamaño 3×3. Se observan los tiempos de cómputo 4.3a, además del factor de aceleramiento 4.3b. De los métodos propuestos en el presente trabajo, el método 4 descrito en la sección 3.5, es el que obtiene un menor tiempo de procesamiento, también obtiene un mejor factor de aceleramiento que los demás métodos.

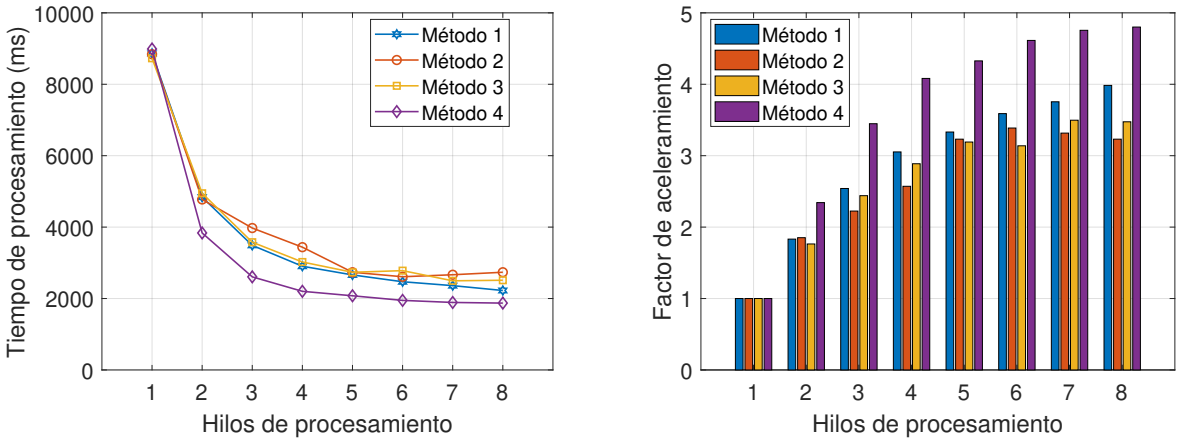


(a) Tiempo computacional para todos los casos (b) Factor de aceleración para todos los casos

Figura 4.3: Tiempo computacional y factor de aceleración para todos los métodos usando un kernel de tamaño 3×3.

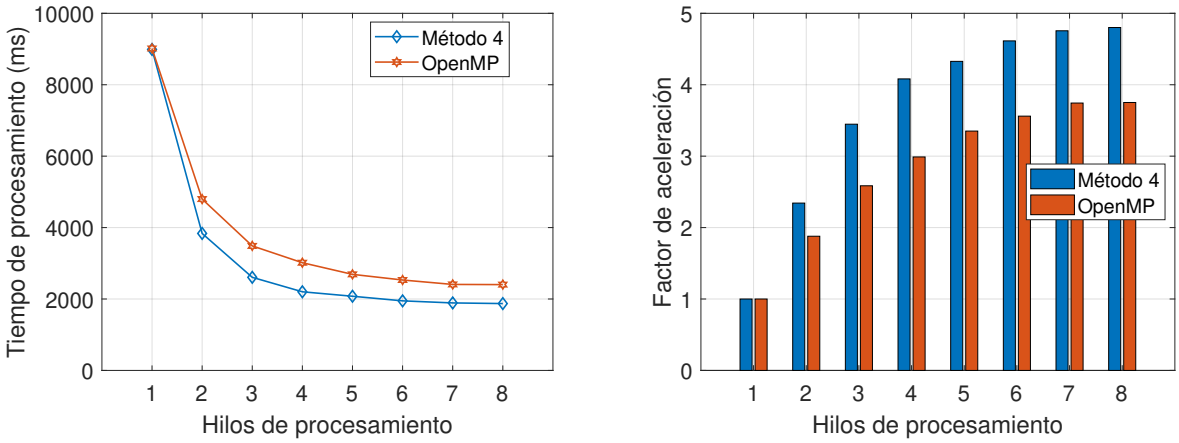
Teniendo en cuenta que la condición de aceptar ventanas de convolución de tamaño dinámico afecta el desempeño final del procesamiento, se aplica una técnica de *Loop Unrolling* o desenroscado de ciclos, la cual busca optimizar ciclos para mejorar la velocidad de ejecución de un programa. Se aplica la técnica en el caso de ventanas de tamaño 3×3 para obtener un mejor rendimiento. En la Figura 4.4 se muestra el desempeño obtenido de todos los métodos ahora usando una optimización en el manejo de los bucles. Se observa que el tiempo de procesamiento fue reducido casi a la mitad, además de un ligero aumento en el factor de aceleración. De igual forma, el comportamiento de los métodos propuestos es el mismo, el método 4 sigue siendo el más rápido de todos los métodos.

Para las siguientes comparativas se usará una ventana convolucional de 3×3 además de una optimización de ciclos para todos los casos. En la Figura 4.5 se muestran los resultados de la comparación del mejor método propuesto contra OpenMP. El método propuesto obtiene una mejora del 20 % en el tiempo de procesamiento respecto a OpenMP, además de un 30 % de mejora en el factor de aceleración.



(a) Tiempo computacional para todos los casos (b) Factor de aceleración para todos los casos

Figura 4.4: Tiempo computacional y factor de aceleración para todos los métodos usando un kernel de tamaño 3×3 usando la técnica de loop unrolling.



(a) Tiempo computacional para el mejor método propuesto (4) contra OpenMP (b) Factor de aceleración para el mejor método propuesto (4) contra OpenMP

Figura 4.5: Tiempo computacional y factor de aceleración para el mejor de los métodos propuestos contra OpenMP para una ventana 3×3 usando loop unrolling.

Una vez obtenida la comparación con OpenMP, se procedió a comparar los métodos propuestos contra OpenCL, que permite usar CPU y GPU para el procesamiento de las imágenes. Se puede observar en la Figura 4.6 que el método propuesto 4 "todos a uno" fue el que obtuvo el mejor desempeño computacional de todos los casos probados usando el CPU como unidad principal de procesamiento. En la Tabla 4.3 se muestran los tiempos específicos de todos los métodos propuestos y de los métodos usando librerías existentes, estos obtenidos usando el 100 % de la capacidad del dispositivo elegido como unidad principal de procesamiento.

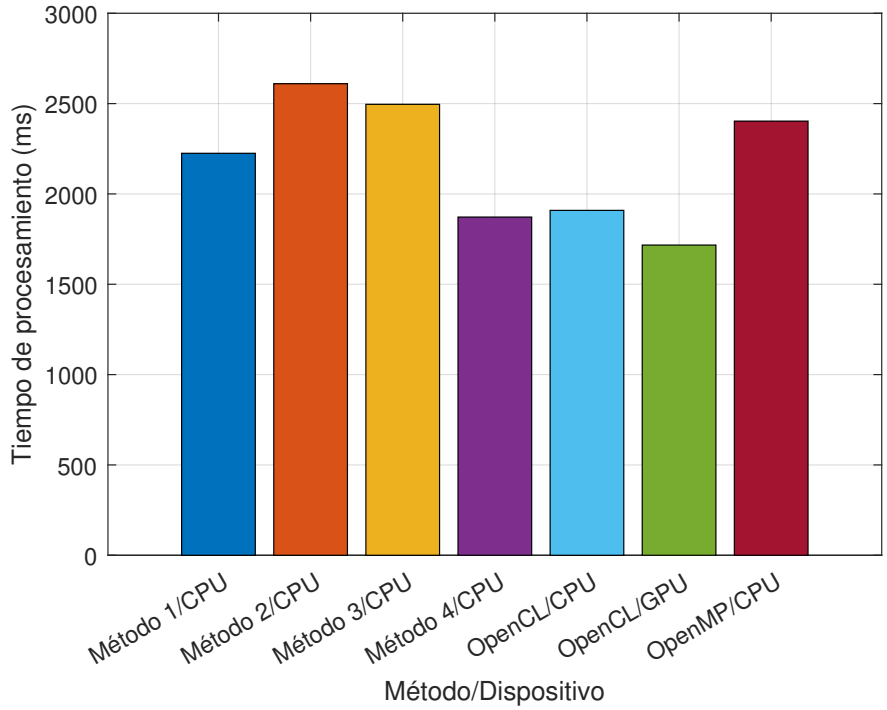


Figura 4.6: Comparación de todos los métodos propuestos respecto a OpenMP y OpenCL.

Tabla 4.3: Comparación de todos los métodos propuestos respecto a OpenMP y OpenCL.

Método/Dispositivo	Tiempo de procesamiento (ms)
Método 1 (uno a uno)	2225
Método 2 (bloque fijo)	2610
Método 3 (bloque dinámico)	2496
Método 4 (todos a uno)	1872
OpenCL/CPU	1909
OpenCL/GPU	1717
OpenMP	2403

Productos Obtenidos del Trabajo de Tesis

- Librería multiplataforma para la extracción concurrente de características visuales de imágenes de tamaño $M \times N$, mediante un proceso convolutivo para K ventanas de tamaño $n \times n$ impar.
- Articulo aceptado para presentación oral en la conferencia *8th International Conference on Software Process Improvement*, CIMPS 2019 (Congreso Internacional de Mejora de Procesos de Software), titulado "Multithreading Programming for Feature Extraction in Digital Images", los días 23-25 de octubre del 2019.

Capítulo 5

Conclusión

En este trabajo se proponen 4 métodos diferentes para dividir la carga de trabajo entre varios hilos de procesamiento, usando como dispositivo principal un CPU, con el fin de aplicar una convolución de dos dimensiones para obtener las características visuales de una imagen digital usando la programación en paralelo.

Este algoritmo de convolución trabaja de forma diferente según el método seleccionado previamente. La primera parte de las pruebas muestran el índice de similitud estructural, comparando los métodos propuestos con OpenCV, obteniendo mejores resultados las características obtenidas por el método propuesto.

Los resultados en tiempo muestran que a mayor número de hilos de procesamiento, las características se obtendrán en un menor tiempo. El tiempo que tarda el algoritmo en extraer las características de una imagen digital se reduce de manera importante al usar hasta el 75 % de los hilos de procesamiento de un procesador, a partir de ahí, la mejora del tiempo de procesamiento no es tan significativa, esto debido a la carga en el procesador al usar un número alto de hilos concurrentes. De los 4 métodos propuestos, el método "todos a uno" fue el que obtuvo mejor resultado tanto en tiempo de procesamiento como en el factor de aceleración, esto por la forma en que se manejan las secciones críticas para el acceso a memoria, lo cual resulta en un número menor de intentos de acceso a su sección crítica, evitando colisiones de memoria y a la vez terminar el procesamiento más rápido. Así que fue comparado directamente con OpenMP, obteniendo el método propuesto un mejor resultado en tiempo de procesamiento de 20 % y un mejor factor de aceleración de 30 %.

La segunda parte de las pruebas de desempeño, se realizan con OpenCL, usando como dispositivo principal de procesamiento el CPU y GPU. Frente a OpenCL/CPU se obtiene una mejora del 2 % en tiempo de procesamiento. OpenCL/GPU obtiene un rendimiento 9 % superior. A pesar que el GPU es un dispositivo altamente paralelizable, no se obtiene un aumento considerable en el rendimiento del procesamiento, debido a factores como la copia de información entre la memoria RAM del sistema a la memoria RAM de la tarjeta gráfica

y viceversa.

Los resultados muestran que es importante la forma de separar la carga de trabajo al recurrir a plataformas paralelas o concurrentes, ya que de esta división se tendrán mejores o peores resultados en tiempo de procesamiento.

Índice de figuras

1.1. Evolución de la frecuencia de reloj y núcleos de los procesadores. 4

1.2. Representación de un perceptron, donde x_1 y x_2 son las entradas, w_1 y w_2 los pesos de cada entrada, $\sum$ la sumatoria de los pesos con su respectiva entrada y $f(x)$ la función de activación del perceptron. 5

1.3. Representación de un perceptron multicapa. 6

2.1. Componentes de un sistema general de procesamiento de imágenes [11]. . . . 10

2.2. Ejemplo de una imagen con diferente tamaño. 11

2.3. Ejemplo de una imagen a color, sus componentes y en escala de grises. . . . 12

2.4. Ejemplo de una imagen aplicando una matriz convolucional. 13

2.5. Aplicación de una matriz convolucional. 14

2.6. Diagrama general de una red neuronal convolucional. 15

2.7. Fases de la implementación de una aplicación en software o hardware usando cómputo paralelo [19]. 16

2.8. Exclusión mutua de hilos de procesamiento mediante el uso de secciones críticas [21]. 20

2.9. Factor de aceleración de acuerdo a la ley de Amdahl, para parámetros de $f = 0.5, f = 0.9, f = 0.99$. 22

2.10. Factor de aceleración de acuerdo a la ley de Gustafson-Barsis, para parámetros de $f = 0.5, f = 0.9, f = 0.99$. 23

3.1. Diagrama general propuesto para la extracción concurrente de las características visuales de imágenes digitales. 26

3.2. Diagrama de flujo para el proceso usando la distribución de trabajo "uno a uno". 29

3.3. Diagrama de flujo para el proceso usando la distribución de trabajo "bloque fijo". 31

3.4. Diagrama de flujo para el proceso usando la distribución de trabajo "bloque dinámico". 33

3.5. Obtención de la distribución delta de trabajo. 34

3.6. Diagrama de flujo para el proceso usando la distribución de trabajo "todos a uno". 35

4.1. Características visuales extraídas usando el algoritmo propuesto y usando el algoritmo de OpenCV. 42

4.2. Tiempo de procesamiento de los métodos propuestos para diferentes tamaños de kernel. 44

4.3. Tiempo computacional y factor de aceleración para todos los métodos usando un kernel de tamaño 3×3. 45

4.4. Tiempo computacional y factor de aceleración para todos los métodos usando un kernel de tamaño 3×3 usando la técnica de loop unrolling. 46

4.5. Tiempo computacional y factor de aceleración para el mejor de los métodos propuestos contra OpenMP para una ventana 3×3 usando loop unrolling. . 46

4.6. Comparación de todos los métodos propuestos respecto a OpenMP y OpenCL. 47

Índice de tablas

4.1. Índice de similitud estructural de la imagen original respecto a sus características extraídas en OpenCV y con el método propuesto. 43

4.2. Índice de similitud estructural de las características obtenidas por OpenCV y el método propuesto. 43

4.3. Comparación de todos los métodos propuestos respecto a OpenMP y OpenCL. 47

Índice de códigos

3.1. Función estática para saber el número de núcleos físicos o lógicos de un procesador. 36
3.2. Clase para medir el tiempo de cómputo usado por un algoritmo. 36
3.3. Función para obtener los ficheros de un directorio. 37
3.4. Clase para la extracción de características de una imagen digital usando programación en paralelo. 38

Bibliografía

[1] John C. Russ. *The Image Processing Handbook*. CRC Press, sixth edition, 2016.

[2] Fabio A. Spanhol, Luiz S. Oliveira, Caroline Petitjean, and Laurent Heutte. *Breast cancer histopathological image classification using Convolutional Neural Networks*. In *International Joint Conference on Neural Networks (IJCNN)*, pages 2560–2567, July 2016.

[3] Frank Rosenblatt. *Perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review*, 65(6):386–408, 1958.

[4] Kunihiko Fukushima. *Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Biological Cybernetics*, 36(4):193–202, apr 1980.

[5] David Rumelhart, Geoffrey Hinton, and Ronald Williams. *Learning Internal Representations by Error Propagation. Parallel distributed processing: explorations in the microstructure of cognition*, 1:318–362, 1985.

[6] J. Weng, N. Ahuja, and T.S. Huang. *Cresceptron: a self-organizing neural network which grows adaptively*. In *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*, volume 1, pages 576–581. IEEE.

[7] Murugan Sankaradas, Venkata Jakkula, Srihari Cadambi, Srimat Chakradhar, Igor Durdanovic, Eric Cosatto, and Hans Peter Graf. *A Massively Parallel Coprocessor for Convolutional Neural Ne tworks. Proceedings of the International Conference on Application-Specific Systems, Architectures and Processors*, pages 53–60, 2009.

[8] Dan C. Cireşan, Ueli Meier, Jonathan Masci, Luca M. Gambardella, and Jurgen Schmidhuber. *Flexible, High Performance Convolutional Neural Network for Image Classification. Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence Flexible*, pages 1237–1242, 2011.

[9] Cheong Ghil Kim, Jeom Goo Kim, and Do Hyeon Lee. *Optimizing image processing on multi-core CPUs with Intel parallel programming technologies. Multimedia Tools and Applications*, 68(2):237–251, 2014.

[10] Ashkan Tousimojarad, Wim Vanderbauwhede, and W. Paul Cockshott. *2D Image Convolution using Three Parallel Programming Models on the Xeon Phi*. pages 1–8, nov 2017.

[11] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. Prentice Hall, second edition, 2002.

[12] John Miano. *Compressed Image File Formats*. Addison Wesley Longman, 1999.

[13] Dhyamis Kleber. *Person Showing Left Eye*. `https://www.pexels.com/photo/beautiful-blue-eyes-close-up-dhyamis-kleber-609549/`, 2019. [Web; accedido el 27-12-2018].

[14] Steven W. Smith. *Digital Signal Processing*. California Technical Publishing, second edition, 1999.

[15] Mehdi Khosrow-Pour (ed.). *Encyclopedia of Information Science and Technology*. IGI Global, 3 edition, 2014.

[16] Philipp Fischer, Alexey Dosovitskiy, and Thomas Brox. *Descriptor Matching with Convolutional Neural Networks: a Comparison to SIFT*. pages 1–10, 2014.

[17] Tianyi Liu, Shuangsang Fang, Yuehui Zhao, Peng Wang, and Jun Zhang. *Implementation of Training Convolutional Neural Networks*. 2015.

[18] Norm Matloff. *Programming on Parallel Machines*. 2014.

[19] Fayez Gebali. *Algorithms and Parallel Computing*. John Wiley & Sons, 2011.

[20] Radatz Jane and IEEE Standards Board. *The IEEE Standard Dictionary of Electrical and Electronics Terms*. IEEE, sixth edition, 1996.

[21] Andrew S. Tanenbaum. *Modern Operating Systems*. Prentice Hall, third edition, 2009.

[22] Sergio Ledesma. *Multithread*. `http://sintesis.ugto.mx/WintemplaWeb/02Wintempla/41Multithread/00Introduction/index.htm`, 2019. [Web; accedido el 23-05-2019].

[23] Ashkan Tousimojarad. *GPRM: A High Performance Programming Framework For Manycore Processors*. PhD thesis, University of Glasgow, 2015.

[24] International Organization for Standardization. *ISO/IEC 14882:2017, Programming languages – C++*. `https://www.iso.org/standard/68564.html`, 2017. [Web; accedido el 15-05-2019].

[25] International Organization for Standardization. *ISO/IEC 14882:2011, Programming languages – C++*. `https://www.iso.org/standard/50372.html`, 2011. [Web; accedido el 15-05-2019].

[26] Kaique Rocha. *Panoramic Photography of Road Between Grasses and Body of Waters*. https://www.pexels.com/photo/asphalt-countryside-empty-grass-105234/, 2019. [Web; accedido el 11-07-2019].

[27] Ricardo Esquivel. *Photo Of City*. https://www.pexels.com/photo/photo-of-city-1604141/, 2019. [Web; accedido el 11-07-2019].

[28] Tae Fuller. *Golden Gate Bridge, San Francisco, California*. https://www.pexels.com/photo/golden-gate-bridge-san-francisco-california-1141853/, 2019. [Web; accedido el 11-07-2019].

[29] Kelly Lacy. *Motor Bike Running Close-up Photography*. https://www.pexels.com/photo/motor-bike-running-close-up-photography-2519374/, 2019. [Web; accedido el 11-07-2019].

[30] Helena Lopes. *Suv Traveling On Road Near Flatiron Building*. https://www.pexels.com/photo/suv-traveling-on-road-near-flatiron-building-1388069/, 2019. [Web; accedido el 11-07-2019].

[31] Zhou Wang, Alan C. Bovik, Hamid Sheikh, and Eero P. Simoncelli. *Image quality assessment: from error visibility to structural similarity. IEEE Transactions on Image Processing*, 13(4):600–612, April 2004.