UNIVERSIDAD NACIONAL DE LA PLATA

Facultad de INFORMÁTICA
UNIVERSIDAD NACIONAL DE LA PLATA

Secretaria de Postgrado

ALGORITMOS PARA AGRICULTURA DE PRECISIÓN UTILIZANDO COMPUTACIÓN DE ALTO RENDIMIENTO

Tesis Doctoral – Doctorado en Ciencias Informáticas

Autor	Director	Codirector
Marco Remigio Pusdá Chulde	Armando De Giusti	Iván García

La Plata, Argentina, 2022

Resumen

La agricultura de precisión (AP) automatiza actividades mediante la recopilación y análisis de datos agrícolas para la toma de decisiones con fines de mejorar la producción agrícola. En AP el procesamiento de imágenes digitales agrícolas implica realizar operaciones para obtener sus transformaciones en la sistematización de tareas agrícolas. El procesamiento de imágenes digitales requiere de varios recursos de cómputo, principalmente de tiempo y memoria; por ello, mediante la aplicación de paralelismo en procesadores heterogéneos con memoria compartida se pretende reducir el tiempo de ejecución con nuevos algoritmos. La obtención de imágenes sobre grandes extensiones de cultivos agrícolas es difícil para los agricultores de manera tradicional, aprovechando nuevos recursos tecnológicos como los vehículos aéreos no tripulados UAV-Unmanned Aerial Vehicle) más conocidos como drones es posible capturar fotografías digitales de alta calidad a diferentes alturas de sobrevuelo. Teniendo en cuenta las pérdidas económicas en la agricultura cada vez mayores debido a las malezas en los cultivos de maíz, se propone un algoritmo aplicando paralelismo mediante la metodología de Foster´s capaz de identificar líneas de cultivo y discriminar malezas. El algoritmo se ejecuta de manera paralela en una serie de pasos intermedios, que incluyen 4 particiones (segmentación, detección de líneas de cultivo, exclusión de cultivo, discriminación de malezas), en cada partición se aplica paralelismo de datos utilizando un procesador de 8 núcleos. Se utilizó Matlab como lenguaje de programación y herramientas orientadas al paralelismo local para análisis de imágenes adquiridas mediante un dron DJI Mavic 2 Pro con una resolución de 5472× 3648 a alturas de 5, 10 y 15 metros. Los resultados obtenidos demuestran que se puede identificar como mínimo líneas de cultivo (85%) y máximo de malezas (93.28%) del total de vegetación en la cuarta semana de seguimiento al cultivo a 15 metros de altura. Igualmente, con el algoritmo propuesto los tiempos de procesamiento evaluados en promedio alcanzan un rango entre 4.54 y 5.67 segundos con imágenes que alcanzan una extensión máxima de 160 metros cuadrados.

Palabras Clave

Análisis de imágenes; Líneas de cultivo; Malezas; Vehículos aéreos no tripulados (UAV – drones); Arquitecturas heterogéneas; Algoritmos paralelos; Procesadores multicore; Agricultura de precisión; Computación de alto rendimiento; Visión por computadora.

Derechos de Autor

Dedicatoria

Este trabajo lo dedico a todos quienes apoyaron con su granito de arena hasta la obtención del título, de manera especial a mi querida esposa Johana y mis hijas Vanessa, Emily y Mabel.

A mi hermano Arnulfo Amílcar (+), quien desde la primera estancia me acompañó al aeropuerto y de seguro que desde el más allá derramó plegarias para finalizar con éxito este proyecto.

Agradecimientos

En primer lugar, agradezco a mi esposa Johana y mis hijas, por su apoyo incondicional durante el periodo de tiempo que duró esta travesía profesional.

A mis familiares y amigos por sus buenos deseos y aliento en cada una de las estancias fuera de mi querido Ecuador.

A la facultad de Informática de la Universidad Nacional de La Plata por la oportunidad de formar parte de tan prestigiosa institución a nivel internacional.

A mi director de Tesis Armando De Giusti, por su guía y apoyo profesional durante el tiempo de estudios presencial y virtual; su experiencia sirvió para aprender y avanzar hasta instancias finales.

A mi codirector de Tesis Iván García-Santillán por su apoyo con el tema de investigación desde su planteamiento hasta la finalización.

A la comunidad de la Universidad Técnica del Norte por todo el apoyo en el proceso del doctorado a nivel internacional.

Índice de Contenidos

Índice de Figuras

Índice de Tablas

Capítulo 1. Introducción

1.1 Antecedentes

La investigación realizada por el grupo ISCAR, oficialmente reconocido por la Universidad Complutense de Madrid, en la línea de investigación de Ingeniería del Software e Inteligencia Artificial, a través del Proyecto RHEA (RHEA, 2014), financiado por la Unión Europea dentro del VII Programa Marco de Investigación y Desarrollo tecnológico durante los años 2010-2014. El objetivo del proyecto RHEA fue diseñar tres prototipos de tractores robotizados y coordinados, en conjunción con vehículos aéreos no tripulados (drones). Cada uno de los prototipos robotizados actuaba en tres campos de cultivo: maíz, trigo y olivos. Los tres tractores fueron equipados con distintas tecnologías sensoriales con el objetivo de realizar la navegación autónoma y la localización de las áreas de interés, para la aplicación de tratamientos selectivos, con el fin de eliminar malezas en el cereal y maíz, y combatir plagas en el olivar, minimizando el uso de agroquímicos, a la vez que se reducen los costos de producción, tiempo y energía. El grupo ISCAR se centró específicamente en el tractor dedicado al cultivo de maíz, que fue equipado con un sistema de visión instalado en la parte superior, mediante el cual se llevaron a cabo procesos para la detección de líneas de cultivo, así como también, para la identificación de malezas localizadas fuera de las líneas de cultivo (inter-líneas).

El proyecto RHEA finalizó en 2014 y dejó sentadas las bases para la utilización de nuevos equipos tecnológicos equipados con sensores de última generación, mejores actuadores, algoritmos de control y decisión precisos, aplicables a la agricultura. RHEA fue la oportunidad para reunir y financiar un gran número de grupos de investigación multidisciplinares para la aplicación práctica y generalizada de las técnicas de agricultura de precisión.

El uso de imágenes capturadas por vehículos aéreos no tripulados (UAV) tiene un enorme potencial para tratamientos agrícolas, sobre todo por la cantidad de datos y calidad de información que aportan para la agricultura de precisión (AP), lo que no es posible con imágenes convencionales obtenidas desde la tierra debido a la limitada cobertura de extensión de terreno.

1.2 Objetivos

De acuerdo con el contexto de la investigación propuesta, se plantean objetivos a mediano plazo mediante el perfeccionamiento en la detección de líneas de cultivo y malezas en cultivos de maíz con algoritmos de tratamiento de imágenes adquiridas empleando vehículos aéreos no tripulados mediante lenguajes de programación paralela para mejorar la escalabilidad y el tiempo de respuesta

1.3 General

- Desarrollar algoritmos de visión por computadora utilizando técnicas de programación de alto rendimiento para detección automática de líneas de cultivo y malezas en campos de maíz, estudiando su mejora en cuanto a tiempo de procesamiento, escalabilidad, arquitectura de soporte.

1.4 Específicos

- Describir el estado del arte respecto a las arquitecturas paralelas, técnicas de programación de alto rendimiento y algoritmos de visión por computador para detección automática de líneas de cultivo y malezas.

- Diseñar algoritmos de visión por computador para la detección de líneas de cultivo y malezas en campos de maíz utilizando técnicas de programación de alto rendimiento.

- Implementar algoritmos de visión por computador para la detección de líneas de cultivo y malezas en arquitecturas multicore.

- Evaluar el rendimiento de los métodos propuestos y versiones secuenciales originales utilizando métricas cuantitativas para arquitecturas multiprocesador.

1.5 Motivación

Actualmente, la agricultura de precisión juega un papel importante en el contexto agrícola a través del uso de recursos tecnológicos que permiten mejorar la producción, ajustando el proceso de la fertilización, la aplicación de cantidades y dosis precisas de insumos agrícolas como son los fertilizantes y productos químicos para el control de plagas y enfermedades. La utilización de las TIC en la sistematización de procesos agrícolas ayuda a resolver problemas críticos de

la agricultura tradicional: desperdicio de recursos, altos costos e impacto medioambiental.

Las técnicas de análisis de imágenes y visión por computador son utilizadas ampliamente para respaldar las tareas de la AP, mediante diversas aplicaciones agrícolas que requieren gran esfuerzo computacional para automatizar de manera eficiente (Chillet & Hübner, 2014). Estas técnicas han permitido mejorar la calidad de vida de los agricultores permitiéndoles realizar las tareas de manera rápida y eficiente debido a diferentes factores, entre ellos los siguientes.

- ✓ Reducción de costos de equipos y accesorios
- ✓ Aumento de la eficiencia de cálculo computacional
- ✓ Mejoramiento de calidad de cámaras
- ✓ Creciente interés en sistematización en diferentes tareas agrícolas como la detección automática de líneas de cultivo (García-Santillán, Pusdá, et al., 2017), malezas (Guerrero et al., 2012; Jiménez-Brenes et al., 2019), plagas y enfermedades (Roldán-Serrato et al., 2018), personas, animales y obstáculos en diferentes campos de cultivo (Campos et al., 2016).

Sin embargo, todavía existe una serie de retos a superar respecto a precisión, tiempo de ejecución y especialmente la aplicación en tiempo real (Barbedo, 2016). El cambio tecnológico, fundamentalmente a partir de los procesadores multicore, ofrece la posibilidad de utilizar nuevos paradigmas de hardware y software (técnicas de programación), en los cuales coexisten esquemas de memoria compartida con mensajes (Leibovich et al., 2012), utilización de aceleradores gráficos (GPU, FPGA, Xeon Phi) que presentan una alternativa para alcanzar un alto rendimiento en determinadas aplicaciones informáticas (HajiRassouliha et al., 2018; Weinstock et al., 2016).

Las unidades de procesamiento gráfico (GPU) es un tipo de hardware nacido para mejorar el rendimiento de los juegos por computador. Con el avance tecnológico las GPU han evolucionado para ser utilizados en otras áreas incluida la agricultura de precisión (Murilo, 2014). Por su estructura computacional, las GPU pueden ofrecer un rendimiento superior en comparación con las clásicas

unidades centrales de proceso (CPU) (HajiRassouliha et al., 2018) en aplicaciones que requieran grandes cargas de trabajo computacional. Adicionalmente, lenguajes de programación como CUDA (Compute Unified Device Architecture - Arquitectura Unificada de Dispositivos de Cómputo) (Nvidia, 2019), se pueden adaptar a la programación de algoritmos paralelos que pueden ser utilizados en aplicaciones de análisis de imágenes en campos agrícolas con tiempos de ejecución más cortos. La integración de la visión por computador y la computación de alto rendimiento (HPC) ofrecen un futuro alentador para resolver problemas específicos en agricultura de precisión (Liao et al., 2017).

La aplicación de drones en cultivos está produciendo un cambio en el modo de operar de las empresas y agricultores, principalmente por bajo costo de adquisición, mínimo mantenimiento, facilidad de configuración y capacidad de transmisión de datos. En diferentes países se cuenta con profesionales para la operación de drones para el sector agropecuario adaptándose a cualquier tipo de cultivo sin necesidad de pistas para despegar o aterrizar. Las condiciones meteorológicas actuales son cambiantes y el agricultor debe procurar no acelerar este cambio climático sino contrarrestarlo. Se puede alcanzar una buena productividad cuidando a la vez el medio ambiente aplicando de manera correcta las tecnologías actuales. Las actividades frecuentemente utilizadas por los drones en los diferentes campos de la sociedad se detallan a continuación:

- ✓ Inspección y monitoreo de instalaciones y obras de infraestructura.
- ✓ Investigaciones atmosféricas.
- ✓ Topografía y cartografía temática.
- ✓ Geología y prospección petrolífera y gasífera.
- ✓ Gestión de riesgos y desastres naturales (incendios, inundaciones, etc.).
- ✓ Exploración de lugares de difícil acceso, salvamento y rescate.
- ✓ Control medioambiental.
- ✓ Limnología y oceanografía.
- ✓ Investigaciones sobre conservación de la biodiversidad.
- ✓ Medios de comunicación y entretenimiento.
- ✓ Movilidad, tráfico y logística en general.
- ✓ Actividades agrícolas y pecuarias.

- ✓ Aplicación de productos fitosanitarios.

En grandes extensiones de cultivo es difícil el control de los cultivos por parte de los agricultores. A esto se suman las zonas de difícil acceso que pueda haber y el gran trabajo humano que se requiere en la agricultura convencional para las diferentes labores agrícolas. Con el uso de drones que portan cámaras de diferentes tipos se puede elaborar con más rigor un análisis del campo, revisar datos y tomar decisiones en varios aspectos como los siguientes:

- ✓ Estado fenológico de las plantas.
- ✓ Estado del suelo.
- ✓ Fertilidad del suelo.
- ✓ Estimar rendimiento.
- ✓ Detectar tempranamente estreses en los cultivos.

La detección temprana de plagas y malezas puede evitar una mala cosecha. La detección de enfermedades y plagas puede reducir el estrés al que son sometidas las plantas debido a lo cual afectan el rendimiento de los cultivos. La aplicación de pesticidas de forma manual con mochilas o con equipos pulverizadores terrestres requiere de mucho tiempo, esfuerzo personal y no siempre es efectivo. Un dron comandado remotamente puede recorrer el campo de cultivo en un corto periodo de tiempo para capturar imágenes de buena calidad para diversas actividades agrícolas. Esto presenta una gran ventaja, a la que se le suma un tratamiento preciso y localizado por sistemas de posicionamiento global (GPS). Por tanto, habría menos contaminación y mayor productividad.

En agricultura de precisión, se han realizado diferentes trabajos utilizando análisis de imágenes agrícolas para varias actividades agrícolas como: identificación de calidad del arroz (Zareiforoush et al., 2015), clasificación de granos (Vithu & Moses, 2016), detección de enfermedades en plantas (Barbedo, 2016), detección de enfermedades en cítricos (Andrade et al., 2017), detección de obstáculos en videos agrícolas (Campos et al., 2016), identificación de malezas (García-Santillán, Montalvo, et al., 2017; García-Santillán, Peluffo-Ordoñez, Caranqui, Pusdá-Chulde, et al., 2018; García-Santillán, Pusdá-Chulde, et al., 2017; López-Granados et al., 2019; Torres-Sánchez et al., 2018). Además,

el avance de técnicas y herramientas de inteligencia artificial aplicadas en agricultura de precisión ha tenido interés sobre identificación de plantas utilizando Machine Learning (Shah et al., 2016), producción de alimentos con Deep Learning (Kamilaris & Prenafeta-Boldú, 2018).

1.6 Aporte

La necesidad de automatizar las tareas en la agricultura con la finalidad de mejorar la producción agrícola y disminuir la contaminación ambiental no es exclusiva de los países del primer mundo. Los países en vías de desarrollo (como Ecuador) también requieren aplicar la tecnología en el sector agrícola, que resulte pertinente al contexto y las realidades/características propias de las regiones.

Ecuador es un país que basa principalmente su economía en la explotación del petróleo y la exportación de ciertos recursos marinos (p. ej. camarón, atún) y agrícolas (banano, cacao, flores). Específicamente, la provincia del Imbabura, donde se llevó adelante gran parte de la investigación, depende mayoritariamente de la actividad agropecuaria (SENPLADES, 2017). Los productos agrícolas (p. ej. papa, maíz, haba, quinua, arveja, fréjol, cebolla) y ganaderos (carne, leche) que se producen en esta zona abastecen al consumo interno del norte del Ecuador y parte del sur de Colombia. Las labores agrícolas se han realizado históricamente utilizando técnicas manuales tradicionales generando la baja calidad de los productos, altos costos de producción y contaminación del suelo. Por lo que la incorporación de TIC en el sector agrícola, para automatizar tareas en los cultivos de productos del sector, resulta vital para minimizar los efectos adversos ya mencionados, con la consecuente contribución a la mejora de la calidad de vida de los agricultores y la población de la provincia, así como el aseguramiento de la soberanía alimentaria de la región.

La automatización de los procesos agrícolas utilizando sistemas de visión por computador es una práctica con mayor proyección que tiene el sector agrícola. En cultivos de maíz existen dos aspectos importantes para tener en cuenta durante el análisis de imágenes aplicados a la agricultura de precisión (i) la detección de líneas de cultivo y (ii) la identificación de malezas. En efecto, las

plantas localizadas dentro de los espacios entre las líneas de cultivo pueden ser consideradas como malezas con alta probabilidad, requiriendo la aplicación de tratamientos selectivos (RHEA, 2014). Además, la identificación de las malezas intra-fila (localizadas dentro del mismo surco) es una tarea determinante. Sin embargo, este problema es complejo de tratar, debido a que el cultivo y las malezas intra-fila se encuentran entrelazados y solapados, con un alto grado de similitud en sus formas espectrales.

El aporte general de la propuesta estará enfocó en la investigación y desarrollo de algoritmos de visión por computador utilizando técnicas de programación de alto rendimiento procesando imágenes adquiridas con drones considerando como base el trabajo realizado por (García-Santillán, Guerrero, et al., 2017), implementado en plataformas de hardware paralelas, con la finalidad de determinar la altura mínima/máxima de captura de imágenes con dron para reconocimiento automático de líneas de cultivo/malezas en campos de maíz y comparar el rendimiento de los algoritmos secuenciales con los nuevos algoritmos paralelos.

1.7 Publicaciones Propias

1.7.1 Revisión Literatura

Image Analysis Based on Heterogeneous Architectures for Precision Agriculture: A Systematic Literature Review: https://doi.org/10.1007/978-3-030-33614-1_4

La agricultura de precisión (AP) es una estrategia de gestión agrícola que utiliza TIC (Tecnologías de información y comunicación) en la obtención de información desde distintas fuentes con el fin de apoyar en la toma de decisiones, considerando aspectos ambientales y económicos para optimizar las tareas del agricultor y brindar productos de calidad al consumidor. La aplicación de AP en el agro disminuye los tiempos de las actividades manuales, conduce a optimizar el uso de agroquímicos; evitando aumento del costo de producción, deterioro del suelo y la contaminación ambiental. Actualmente, es un área que se encuentra en un proceso de activo crecimiento, aprovechando de los avances tecnológicos, en visión por computador, arquitecturas heterogéneas (Multicore, GPU, FGPA)

y técnicas de inteligencia artificial (Machine Learning, Deep Learning), ha permitido sistematizar variedad de actividades agrícolas, tales como identificación de patologías, identificación de malezas en cultivos, así como la presencia de plagas. En este trabajo se presenta una revisión sistemática de literatura (SRL) sobre técnicas de procesamiento y análisis de imágenes aplicadas en agricultura de precisión utilizando tecnologías heterogéneas. Por lo que, se buscó, analizó y sintetizó 32 artículos científicos de los últimos cinco años provenientes de cuatro bases de datos bibliográficas relevantes (Scopus, ScienceDirect, IEEE Xplore, SpringerLink). Las publicaciones seleccionadas responden a cuatro preguntas de investigación planteadas en este estudio: i) ¿Qué tareas agrícolas se han automatizo utilizando técnicas de análisis de imágenes? ii) ¿Qué técnicas de análisis de imágenes se han utilizado para detección de cultivos y malas hierbas? iii) ¿Qué arquitecturas heterogéneas se han utilizado en AP? iv) ¿Cuáles son los retos y tendencias en AP?. A partir de los resultados obtenidos se identificaron grandes oportunidades para el análisis de imágenes (segmentación), aprendizaje automático y uso de aceleradores gráficos (GPU), las cuales se destacan como técnicas y herramientas prometedoras para el desarrollo de sistemas automáticos eficientes y precisos con miras a su aplicación en tiempo real en diversas tareas agrícolas.

1.7.2 Algoritmos Paralelos

Parallel CPU-Based Processing for Automatic Crop Row Detection in Corn Fields: https://doi.org/10.1007/978-3-030-68080-0_18

Con el avance tecnológico, las aplicaciones informáticas actuales generan grandes volúmenes de datos heterogéneos (texto, imágenes, video, audio), ocasionando un incremento considerable en los tiempos de ejecución de su procesamiento. La utilización de arquitecturas heterogéneas en la sistematización de procesos son una alternativa para obtener mejores tiempos de procesamiento, optimizando recursos computacionales, mediante la interacción entre arquitecturas CPU (Unidad Central de Proceso) y GPU (Unidad de Procesamiento Gráfico) para procesamiento paralelo. Específicamente, en agricultura de precisión (AP), el análisis de imágenes para la detección automática de líneas de cultivo es fundamental para la utilización de sistemas

autónomos aplicados a diferentes procesos agrícolas en tiempo real. En el trabajo citado se propone la utilización de programación paralela para la implementación de un algoritmo existente para la detección de líneas curvas y rectas de cultivos, basado en micro-ROI (Pequeñas regiones de interés), con imágenes capturadas en campos de maíz durante etapas iniciales de crecimiento. Se utilizaron las librerías de Matlab (Parallel Computing ToolBox) para la implementación de forma paralela y secuencial en una CPU multinúcleo con el objetivo de contrastar tiempos de ejecución en un conjunto de 300 imágenes. Los resultados obtenidos se validaron estadísticamente mediante la prueba T-Student en el lenguaje de programación R. Los tiempos evaluados indican aceleraciones del 20% en promedio en el reconocimiento de líneas de cultivo, demostrando una mejora notable del rendimiento del algoritmo paralelo sobre el secuencial.

1.7.3 Imágenes adquiridas con dron

Detection of Crop Lines and Weeds in Corn Fields Based on Images Obtained from a Drone: https://doi.org/10.1007/978-3-030-84825-5_3

La agricultura de precisión (AP) automatiza procesos mediante la recopilación y análisis de datos agrícolas para la toma de decisiones y obtención de producciones agrícolas eficientes. Las malezas son uno de los principales factores que afectan el rendimiento y la calidad de los productos agrícolas. La AP es una de es una de las herramientas claves para la detección y control de malezas en los cultivos de maíz mediante al análisis de imágenes digitales con la finalidad de realizar diferentes acciones que permitan disminuir el riesgo de la producción de manera tradicional o automatizada. En el presente trabajo se propone una alternativa para la detección líneas de cultivo y malezas en cultivos de maíz durante las primeras 4 semanas de crecimiento utilizando imágenes adquiridas mediante un dron (UAV) DJI Mavic 2 Pro con una resolución de 5472× 3648. La implementación del algoritmo se realizó con las librerías de Matlab (Image Processing ToolBox) en 4 fases: detección de vegetación, detección de líneas de cultivo, exclusión del cultivo, detección de malezas. Esto permitió separar las malezas a partir de las líneas de cultivo detectadas en las imágenes capturadas a 5, 10 y 15 m de altura. Los resultados del algoritmo paralelo

mostraron que se puede identificar las líneas de cultivo (85%) y malezas (84.61%) del total de vegetación en la cuarta semana hasta 15 metros de altura, con una altura superior a los 15 metros la identificación se dificulta debido a similares tamaños de las plantas de cultivo y malezas. Los tiempos promedios de procesamiento evaluados con el algoritmo secuencial (9.41 segundos) como con el algoritmo paralelo (4,54 segundos con 8 núcleos) muestran una disminución de tiempo significante (48%), posibilitando un escalamiento importante relacionado con la cobertura de extensión de cultivo (160 metros cuadrados).

Capítulo 2. Marco Teórico

2.1 Agricultura de precisión

La agricultura de precisión (AP) según la ISPA (International Society of Precision Agriculture) “es una estrategia de gestión que recoge, procesa y analiza datos temporales, espaciales e individuales y los combina con otras informaciones para respaldar las decisiones de manejo de acuerdo con la variabilidad estimada, y así mejorar la eficiencia en el uso de recursos, la productividad, la calidad, la rentabilidad y la sostenibilidad de la producción agrícola” (ISPA, 2022). La AP involucra el uso GPS y de otros medios electrónicos para obtener datos del cultivo, aumentando el valor del rendimiento productivo en cuanto a calidad y cantidad en las cosechas (García & Flego, 2016).

Según datos recopilados y analizados por la Organización de las Naciones Unidas para el año 2050 la población alcanzará los 9500 millones de personas (Njoroge et al., 2018a); por lo que será necesario un 70% más de alimentos de lo que se produce actualmente para poder alimentar a esta cantidad de personas. Debido a los limitados recursos en el campo de la agricultura se requiere de la innovación para tener una agricultura sostenible y eficiente (Vasudevan et al., 2016).

El monitoreo de cultivos es un aspecto esencial de la agricultura de precisión que captura información en diferentes etapas de crecimiento del cultivo (Tian et al., 2020) con la finalidad de conocer propiedades relacionadas a la cosecha tales como el crecimiento, rendimiento, sanidad del cultivo, entre otras. Las tecnologías más utilizadas en AP son: GPS, Nanosensores o sensores en general, UAV, entre otros. Mientras que las tecnologías o software más utilizadas están: sistemas de información geográfica (GIS), imágenes multiespectrales, mapas de suelos, entre otros. (Cisternas et al., 2020). En la Figura 1 se muestra el equipamiento moderno y software especializado para AP con el objetivo de mejorar la calidad y la rentabilidad de los cultivos. Entre las técnicas utilizadas en dicho sistema se encuentran los GPS, drones e imágenes satelitales para recopilar datos agrícolas y en base a datos recolectados los agricultores toman decisiones para mejorar la productividad.

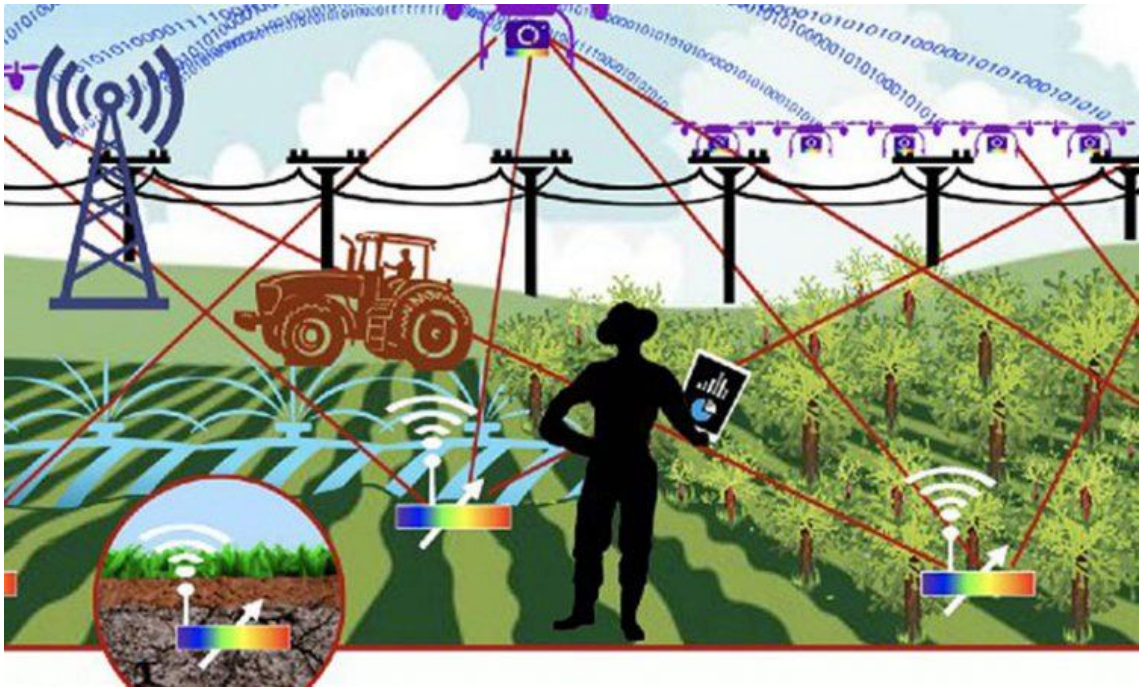


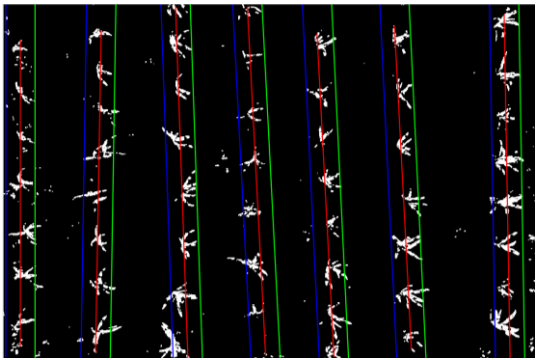
Figura 1: Elementos básicos para agricultura de precisión
(Agricultura de Precisión, 2022)

2.1.1 Visión por computadora

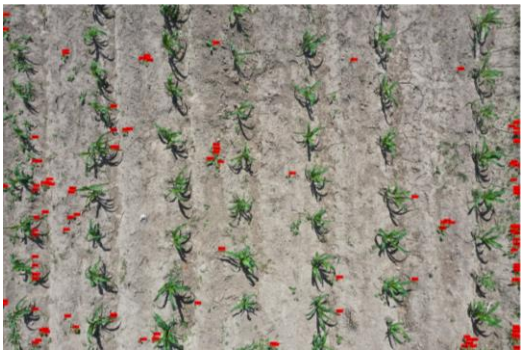
Los sistemas de visión por computadora permiten identificar y comprender automáticamente el mundo visual, simulando de la misma manera que lo hace la visión humana. Los investigadores en visión por computadora desarrollan algoritmos para diversas tareas de percepción visual, incluido la detección, reconocimiento y comprensión de objetos utilizando imágenes digitales. (Feng et al., 2019)

Las técnicas de visión por computadora crecieron significativamente a fines de la década de 1960 y principios de la de 1970 y posteriormente han sido aplicado en diversos campos aprovechando nuevas tecnologías y técnicas para el reconocimiento de patrones, el aprendizaje automático, gráficos por computadora, reconstrucciones en 3D, realidad virtual y realidad aumentada. Hoy en día la visión por computadora permite realizar tareas de alto nivel como reconocimiento de objetos, navegación automática de vehículos, detección de rostros y huellas dactilares, entre otras (Kakani et al., 2020). La Figura 2 muestra un ejemplo de procesamiento de imágenes agrícolas mediante visión por

computador para identificar líneas de cultivo (2a) y etiqueta las malezas que se encuentran fuera del perímetro de las plantas de cultivo (2b).



a) Detección de líneas de cultivo



b) Etiquetación de malezas

Figura 2: Procesamiento de imágenes digitales agrícolas
(Pusdá-Chulde, Robayo, et al., 2021)

2.1.2 Procesamiento de imágenes

El procesamiento digital de imágenes es el conjunto de técnicas que se aplican a las imágenes digitales con el objetivo de mejorar la calidad o facilitar la búsqueda de información dentro de la misma. Con el procesamiento de imágenes se puede extraer, modificar o alterar propiedades de las imágenes, tales como luminosidad, contraste, niveles de color, saturación, entre otros aspectos (García-Santillán, Pusdá-Chulde, et al., 2017; Korohou et al., 2020; Ramírez Osuna, 2012; Sucar & Gómez Giovani, 2015). Para que un sistema por visión de computadora procese las imágenes digitales se debe realizar las siguientes fases:

Adquisición de imágenes: la primera fase consiste en obtención de las imágenes a ser procesadas. Las imágenes pueden ser obtenidas por diferentes medios (cámaras, drones, satélites, entre otros) y almacenadas en formatos estándares para su procesamiento (jpg, jpeg, png, tif) (García-Santillán & Pajares, 2016).

Preprocesamiento: en esta fase se ajusta, modifica o mejora la calidad informativa de la imagen para facilitar al sistema la interpretación de la información y reducir el entorno que no es de interés. El preprocesamiento se aplica a menudo para eliminar el fondo y el ruido debido a variaciones en las condiciones de luz ambiental (Korohou et al., 2020).

Segmentación: esta fase divide los elementos de interés y discriminantes en regiones blancas y negras respectivamente para convertir los datos en un entorno lógico de ceros y unos permitiendo a los sistemas reconocer y extraer los elementos de interés (Njoroge et al., 2018b; Zheng et al., 2020).

Descripción: en esta fase se procede a obtener cada elemento identificado a través de rasgos que representan una característica cuantitativa que pueda entender el sistema. Estos rasgos pueden ser propiedades geométricas como el área y el perímetro de un objeto o topológicos (propiedades relacionadas a la estructura de un objeto). La técnica más utilizada en esta fase es la transformada de Hough (García-Santillán, Montalvo, et al., 2017).

Reconocimiento: en esta fase se clasifica en categorías los objetos presentes en la imagen usando los descriptores del proceso anterior. Los objetos detectados que presenten descriptores semejantes se agrupan automáticamente en una misma clase o categoría o con una mínima intervención humana a través de técnicas y algoritmos computacionales (García-Santillán & Caranqui, 2015).

2.1.3 Técnicas de procesamiento de imágenes

Las imágenes digitales son representaciones bidimensionales que se representan mediante una matriz numérica, frecuentemente binaria. Estas imágenes, generalmente se obtienen al convertir señales continuas en formato digital, las cuales se pueden visualizar en diversos medios como impresoras digitales, monitores y dispositivos de proyección digital (García-Santillán & Caranqui, 2014). Las técnicas más utilizadas son las siguientes:

Escala de grises: proceso para modificar la matriz de información en valores de escala blanco y negro ya que resulta mucho más conveniente a la hora de manipular información contenida en la imagen. Las imágenes con formatos RGB hacen uso de tres canales que describir la cantidad de rojo, verde y azul en una imagen. Mediante el cambio de formato de escala de grises los valores RGB de cada píxel en la matriz se modifican a través de la intensidad del color para transformarlos en tonalidad blanco y negro (García-Santillán, Pusdá-Chulde, et al., 2017).

Binarización: transforma la imagen de escala de grises, a un entrono lógico donde cada píxel de la imagen puede tomar uno de dos valores posibles (0 o 1) mediante un umbral base que puede ser fijo o automático usando el método de Otsu (García-Santillán, Guerrero, et al., 2017; García-Santillán, Montalvo, et al., 2017; García-Santillán, Pusdá-Chulde, et al., 2017; García-Santillán & Pajares, 2018).

Procesos morfológicos: extrae componentes de la imagen que sean útiles en la representación y descripción de regiones, las operaciones morfológicas simplifican las imágenes y conservan las principales características de forma de los objetos (Costa et al., 2020; Pusdá-Chulde et al., 2020; Zheng et al., 2020). Las operaciones más utilizadas son las siguientes:

- ✓ Extraer píxeles de un objeto y hacerlo más pequeño (erosión).
- ✓ Agregar píxeles a un objeto y hacerlo más grande (dilatación).
- ✓ Convertir los ceros en unos y los unos en ceros de una imagen binarizada para invertir los colores de blanco y negro en la imagen (complementación).
- ✓ Detectar la transición de dos regiones con diferentes niveles de gris determinando la frontera de cada elemento encontrado en la imagen (Detección de bordes).
- ✓ Medir las propiedades de las regiones de una imagen (Propiedades de regiones).

2.1.4 Técnicas para la detección de líneas de cultivo

La detección de líneas de cultivo es una actividad importante en la sistematización de tareas agrícolas aplicable a diferentes tipos de cultivos utilizando procesamiento de imágenes.

Transformada de Hough: algoritmo elaborado por Paul V. C. Hough, originalmente inventado para reconocer líneas complejas en fotografías (Hough, 1962). Para la detección de los bordes se usa un plano en 2 dimensiones llamado Hough Space (HS) con un eje horizontal y eje vertical representando la pendiente y la intersección de una línea en la imagen, respectivamente. El principio del Hough Transform (HT) es mapear el plano de la imagen (x; y) al HS definida por q y h, donde q representa la distancia normal desde el origen a la línea recta y h representa el ángulo que el vector forma normales con el eje horizontal de la imagen. El algoritmo de transformación de Hough detecta líneas al encontrar los pares (ρ, θ) que tienen un número de intersecciones mayor a un cierto umbral establecido. Cuando el algoritmo detecta una línea esta es representada con un punto en el ST indicando su ángulo y el valor p en la imagen (Anthony Simon & Min, 2020; Bah et al., 2020; Davies, 2009; García-Santillán, Guerrero, et al., 2017; García-Santillán, Montalvo, et al., 2017; Romeo et al., 2012; Rovira-Más et al., 2005).

Regresión lineal: este enfoque estima la pendiente e intersección de la ecuación de la línea recta y asigna una ecuación a cada una de las líneas de cultivo. El método de mínimos cuadrados es el más común para estimar los parámetros y realizar el ajuste de la línea recta (Costa et al., 2020; L. Wang et al., 2015).

2.1.5 Técnicas de identificación de vegetación

Las diversas técnicas utilizadas en la detección de vegetación por visión en computadoras se pueden determinar de las siguientes clasificaciones (Guijarro et al., 2015):

Índice espectral visible: método que tiene el objetivo de obtener una imagen en escala de grises con alto contraste resaltando la vegetación, es decir, conteniendo píxeles brillantes para las plantas y píxeles oscuros para el suelo y demás restos (piedras, árboles, cielo) (Guerrero et al., 2012; Tellaeche et al., 2011).

Umbralización: método que discrimina el suelo y la vegetación del terreno a través de la aplicación de un umbral en la imagen, segmentando la región de interés en líneas comprensibles para el algoritmo (Burgos-Artizzu et al., 2011; Costa et al., 2020).

Aprendizaje: métodos que utilizan técnicas de Machine Learning para determinar las zonas de interés a través del entrenamiento supervisado o no supervisado (Gašparović et al., 2020; Kakani et al., 2020; Lu & Young, 2020; Shah et al., 2016).

Wavelet: métodos basados en la frecuencia del contenido de la imagen y el análisis de las texturas de las plantas (Guijarro et al., 2015).

Índice de vegetación: métodos para la detección de plantas y vegetación especialmente con imágenes espectrales para aplicaciones de AP. Existen diversos indicadores de vegetación, tales como NDVI, NDRE, GNDVI, SAVI, etc., sin embargo, NDVI es el más usado debido a su facilidad y precisión en la detección de plantas y vegetación (Costa et al., 2020; Marques Ramos et al., 2020; Xie & Yang, 2020; J. Zhang et al., 2018a).

Exceso de verde (ExG): El uso de imágenes multibanda es más costoso debido a los equipos y a la información que estos manejan, mientras que una cámara comercial es más económica pero los datos que emplea están en el rango del espectro visible, no obstante, puede arrojar resultados similares a los obtenidos con equipos multiespectrales (García-Santillán, Peluffo-Ordoñez, Caranqui, Pusdá, et al., 2018).

2.1.6 Herramientas y equipos

Según (Abdul et al., 2016) la agricultura de precisión es un sistema de gestión agrícola que tiene en cuenta la variabilidad espacial de ciertas características físico-químicas y biológicas de las áreas cultivadas. Los equipos necesarios se consideran los siguientes:

Sistema de posicionamiento global (GPS): sistema de posicionamiento el cual consiste en una red de satélites que ayuda a los usuarios a registrar información de ubicación (latitud, longitud y altitud). El sistema permite a los agricultores identificar con precisión la ubicación del campo para que los insumos (semillas, fertilizantes, pesticidas, herbicidas y agua de riego) se pueden aplicar en los campos individuales en función de criterios de rendimiento (Costa et al., 2020; Njoroge et al., 2018a; Pusdá-Chulde et al., 2020; Vasudevan et al., 2016).

Tecnologías de sensores: permiten recopilar datos para medir la temperatura, la humedad, la vegetación, el nivel de nutrientes, el vapor, el aire dando lugar a la teledetección para conocer condiciones de estrés, distinguir especies de cultivos, identificar plagas y malezas, monitorear la sequía, el suelo y las condiciones de las plantas (De Giusti et al., 2020; Ponnusamy & Natarajan, 2021; Pusdá-Chulde et al., 2020; Vasudevan et al., 2016).

Sistema de información geográfica (GIS): mapa computarizado cuya función es utilizar la estadística y métodos espaciales para analizar datos y geografía. Las datos de los GIS agrícolas proporcionan información sobre topografía archivada, tipos de suelo, drenaje superficial, drenaje subterráneo, pruebas de suelo, riego, tasas de aplicación de productos químicos y rendimiento de cultivos (García & Flego, 2016; Radoglou-Grammatikis et al., 2020a).

Tecnología de tasa variable (VRT): Permiten el muestreo del suelo en cuadrícula ocupa los mismos principios de un muestreo del suelo para obtener un mapa detallado acerca de las necesidades de los nutrientes. El mapa es utilizado por los agricultores para controlar con precisión la cantidad de insumos a los cultivos o suelo (Fabiani et al., 2020; Udal'tsov, 2018).

2.1.7 Vehículos aéreos no tripulados (Drones)

Los vehículos aéreos no tripulados o drones, son aeronaves controlados y gestionados de forma remota o mediante programas informáticos autónomos integrados para la captura de imágenes y video (Mishra & Natalizio, 2020). Entre los principales usos de drones en AP están: posicionamiento en tiempo real del cultivo, contorneado de campos, mapeo de suelos, mapeo de producción y monitoreo de cultivos (Radoglou-Grammatikis et al., 2020b)

La accesibilidad a los drones ha sido un gran paso en la implementación de tecnología en el campo de la agricultura conjuntamente con sistemas de visión por computadora en AP. Debido a la dificultad de acceso y costos de equipos para obtener imágenes satelitales, los drones son alternativas posibles para capturar, manipular y almacenar imágenes desde una vista cenital (Bah et al., 2020; Njoroge et al., 2018a; Ponnusamy & Natarajan, 2021; Pusdá-Chulde et al., 2020).

La importancia de la inteligencia artificial y los sistemas de visión de computadora en la agricultura están muy relacionadas en la industria alimentaria. Los algoritmos predictivos, redes neuronales y cálculos espectrales son algunos de los métodos usados para detectar anomalías, predecir el rendimiento en la producción de alimentos, monitorear el cultivo e incluso el empaquetado del producto, el mismo que será comercializado de diferentes formas y presentaciones (Lu & Young, 2020; Vasudevan et al., 2016).

Los drones cumplen múltiples funciones en la agricultura, como el mapeo de campos, monitoreo de los cultivos en lo que se refiere a presencia de plagas y/o enfermedades, la eficiencia de irrigación, y la aplicación de plaguicidas, entre otros. Adicionalmente, traen múltiples beneficios, como la aplicación precisa, localizada y en áreas de difícil acceso, una menor exposición del aplicador, ahorro de agua y tiempo, y el aumento de la productividad del agricultor (Bah et al., 2020; Costa et al., 2020; Mukherjee et al., 2019; Pusdá-Chulde, Robayo, et al., 2021; Vasudevan et al., 2016). En la Figura 3 se presenta el uso de drones para obtener fotografías de los cultivos agrícolas. Las imágenes tomadas desde

la altura han adquirido una importancia fundamental en la agricultura de precisión.

Figura 3: Obtención de imágenes mediante drones
(HEMAV, 2022)

2.2 Arquitecturas paralelas

Las arquitecturas de hardware están basadas en el modelo de Von Neumann con un procesador y memoria donde se guardan datos y programas, es decir, una máquina secuencial que procesa datos escalares. Debido a los problemas de disipación de calor y rendimiento de los uniprocesadores, surgieron nuevas formas de paralelismo dentro de un mismo chip con los procesadores multicores. Los procesadores multicores permiten aprovechar las mejoras de reducción de tamaño de los transistores, manteniendo o disminuyendo la frecuencia de cada CPU reduciendo considerablemente del consumo energético. Con este tipo de arquitecturas, al tener más de una CPU se pueden alcanzar mejores rendimientos en las aplicaciones (Jiménez-Gonzalez, 2010; Pardo, 2002).

En informática, las arquitecturas paralelas dan un enfoque constructivo del conocimiento del lenguaje basado en la computación paralela que es el uso de múltiples recursos computacionales, los cuales requieren de una gran capacidad de procesamiento y de espacio de memoria, debido a complejas operaciones que se deben realizar, esto se distingue de la computación secuencial en que varias operaciones pueden ocurrir simultáneamente (Barlas, 2015b; Hager & Wellein, 2010).

2.2.1 Taxonomía de Flynn

La clasificación de computadores secuenciales y paralelos se basa en multiplicidad de flujo de instrucciones y del flujo de datos del computador. El flujo de instrucciones es la secuencia de instrucciones ejecutadas por el computador y el flujo de datos es la secuencia de datos sobre los cuales operan las instrucciones (Herbert, 2015). Con estas consideraciones, Flynn clasifica los sistemas en cuatro categorías:

SISD (single instruction stream, single data stream): flujo único de instrucciones y flujo único de datos. Es el concepto de arquitectura secuencial, en cualquier momento, sólo se está ejecutando una única instrucción. Las máquinas SISD poseen un registro simple que se llama contador de programa que asegura la ejecución en serie del programa. Conforme se van leyendo las instrucciones de la memoria, el contador de programa se actualiza para que apunte a la siguiente instrucción a procesar en serie (Hager & Wellein, 2010; Jiménez-Gonzalez, 2010). En la Figura 4 se presenta el funcionamiento de una máquina SISD, mediante un CPU ejecuta una instrucción en un momento dado y busca o guarda un dato en un momento dado.

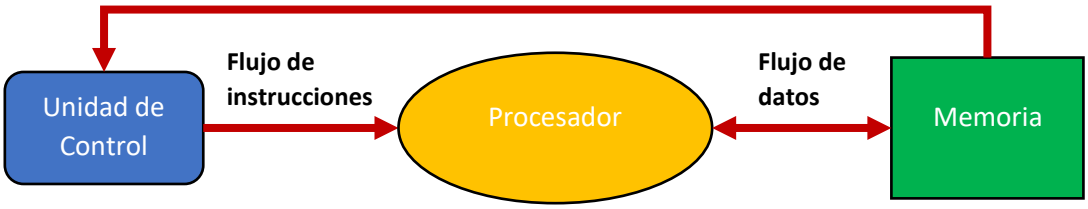


Figura 4:Máquinas SISD
(Herbert, 2015)

SIMD (single instruction stream, multiple data stream): flujo de instrucción simple y flujo de datos múltiple. En este tipo de máquinas hay una sola memoria de programa, desde la cual una unidad de procesamiento especial obtiene y despacha instrucciones. Esto significa que una única instrucción es aplicada sobre diferentes datos al mismo tiempo. En las máquinas de este tipo, diferentes unidades de procesamiento son invocadas por una única unidad de control, adicionalmente soportan procesamiento vectorial (matricial) asignando cada

elemento del vector a una unidad funcional diferente para procesamiento concurrente (Hager & Wellein, 2010; Jiménez-Gonzalez, 2010). En la Figura 5 se presenta el funcionamiento de una máquina SIMD que incluye un conjunto de unidades de procesamiento cada una ejecutando la misma operación sobre distintos datos.

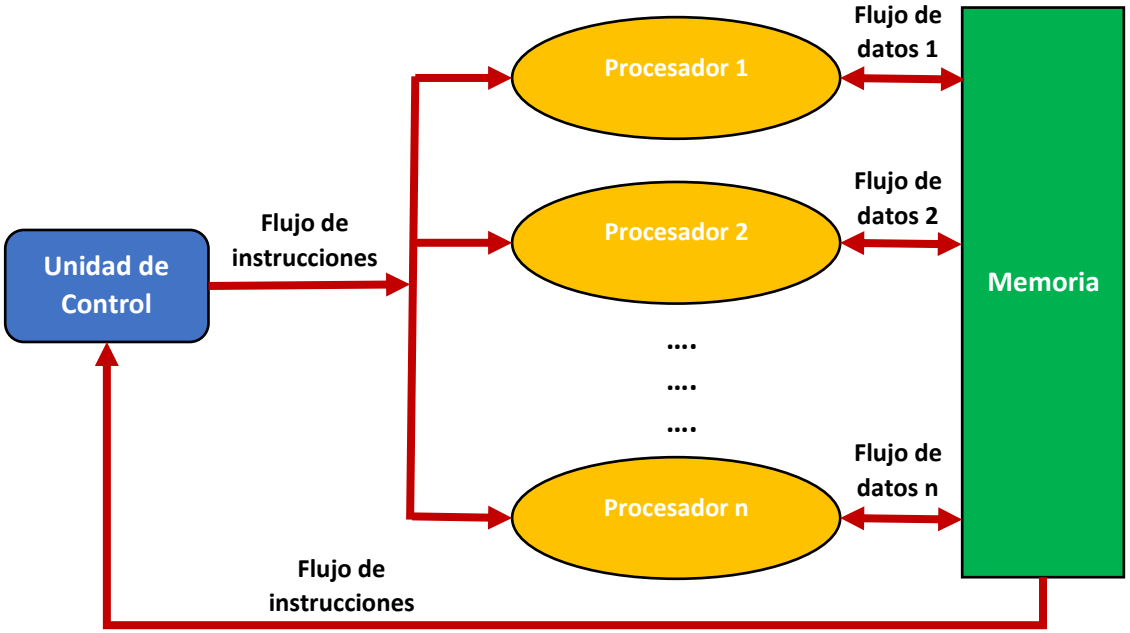


Figura 5: Máquinas SIMD.
(Herbert, 2015)

MISD (multiple instruction stream, single data stream): flujo múltiple de instrucciones y único flujo de datos. En las máquinas MISD existen varias instrucciones actuando sobre el mismo o fracción del flujo de datos o también un mismo flujo de datos fluye a través de varios procesadores. Cada paso, cada elemento de procesamiento obtiene la misma información de la memoria, ejecutadas en paralelo, siendo un modelo muy restrictivo y no se ha usado en ningún computador de tipo comercial (Hager & Wellein, 2010; Jiménez-Gonzalez, 2010). En la Figura 6 se presenta el funcionamiento de una máquina MISD en donde existe varios procesadores, cada uno recibe una instrucción diferente y opera sobre el mismo flujo de datos. El flujo de datos 1 pasa entre varias unidades de procesamiento. En el flujo de datos 2 cada unidad de procesamiento recibe una copia del mismo flujo de datos.

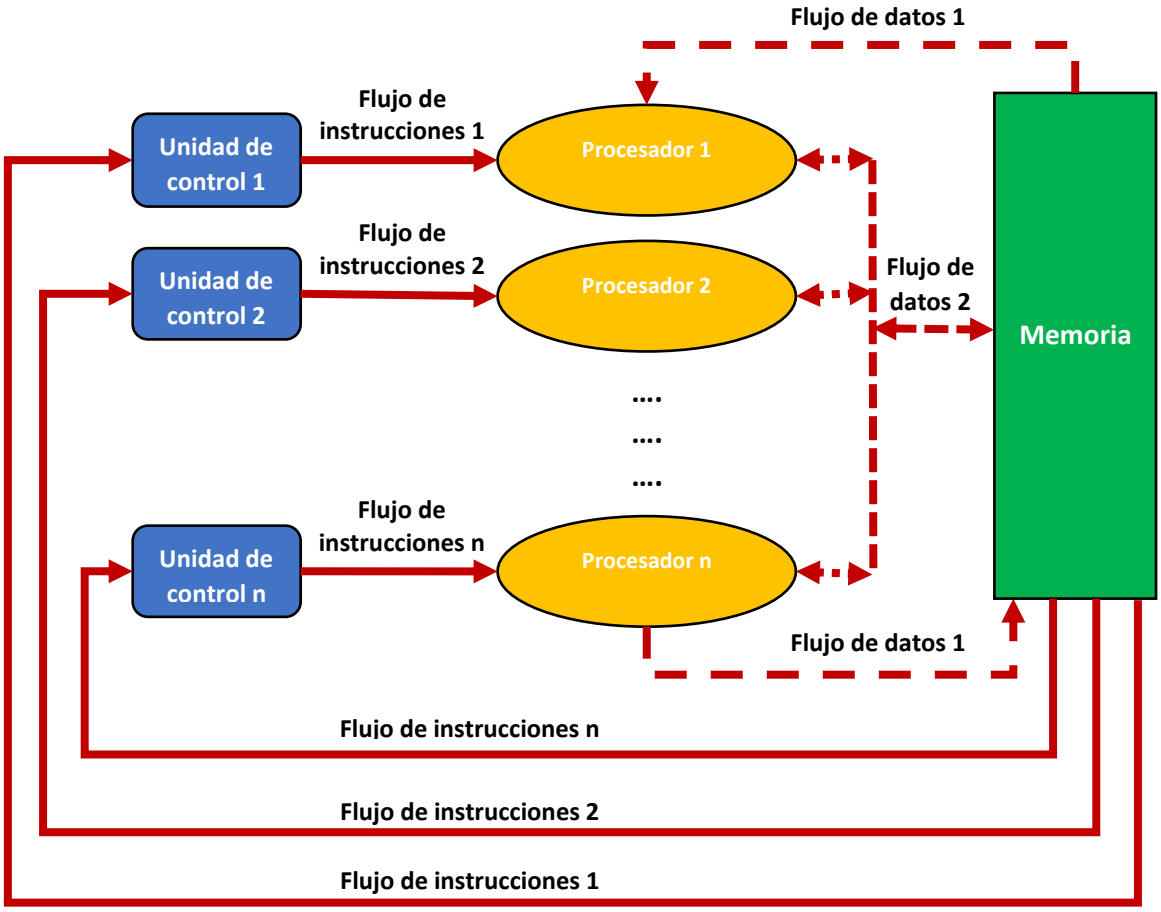


Figura 6: Máquinas MISD.

(Herbert, 2015)

MIMD (multiple instruction stream, multiple data stream): flujo de instrucciones múltiple y flujo de datos múltiple. Este tipo de máquinas poseen múltiples unidades de procesamiento en las cuales se puede realizar múltiples instrucciones sobre datos diferentes de forma simultánea (Jiménez-Gonzalez, 2010). Los clústeres son ejemplos del modelo MIMD, los cuales tienen un múltiple flujo de instrucciones y un múltiple flujo de datos, las unidades de instrucciones son independientes y cada una maneja su propio flujo de datos (Czech, 2017). En la Figura 7 se presenta el funcionamiento de una máquina MIMD, la cual dispone de un procesador independiente y cada uno puede ejecutar un programa diferente sobre sus propios datos.

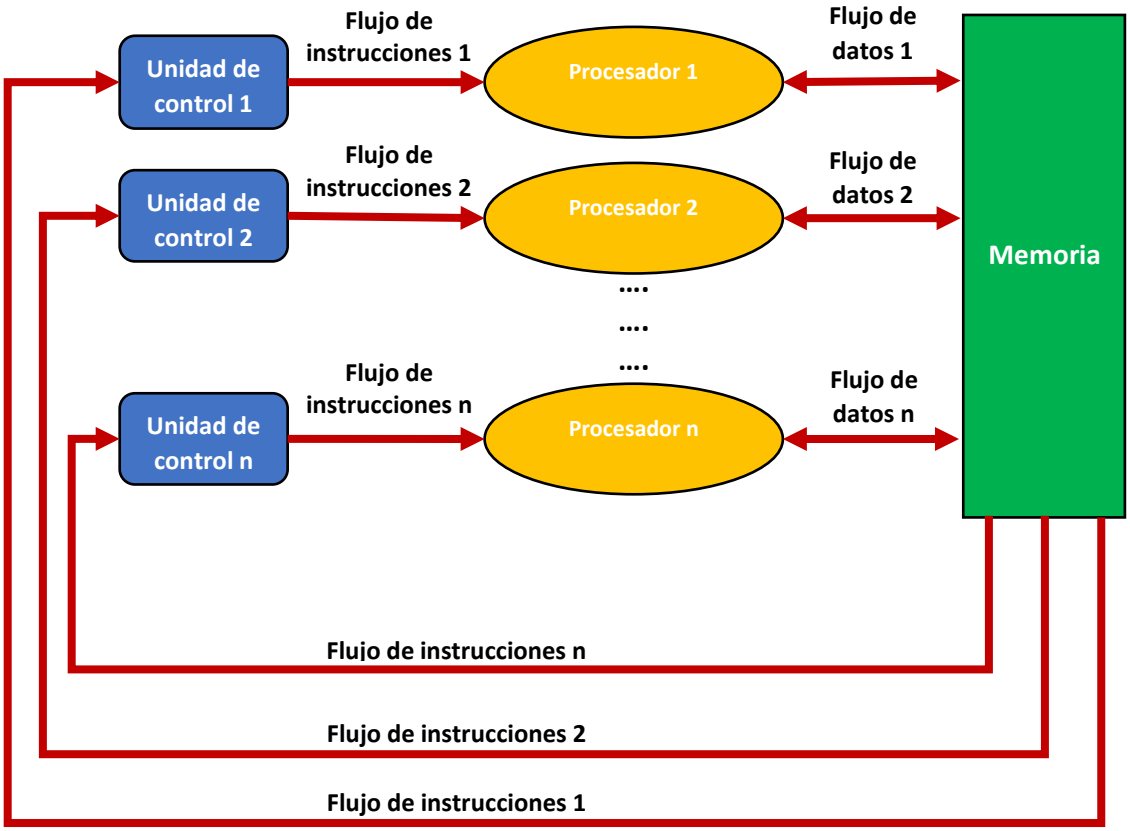


Figura 7: Procedimiento de MIMD

(Herbert, 2015)

2.2.2 Multiprocesadores

Los multiprocesadores son computadores paralelos compuestos por varios procesadores interconectados que pueden compartir un mismo sistema de memoria. Aunque son transparentes para los programadores en lo que a funcionalidad se refiere, existen dos tipos de sistema compartida en relación al rendimiento y en términos de acceso a memoria principal (Hager & Wellein, 2010). Dependiendo de la forma en que los procesadores comparten la memoria, los multiprocesadores se dividen en las siguientes categorías:

UMA (Uniform memory access): son multiprocesadores con sistemas de acceso uniforme a memoria, tienen varios procesadores (CPU) interconectados a través de un mecanismo de buses a una memoria compartida centralizada. La latencia y el ancho de banda son los mismos para todos los procesadores y todas las posiciones de memoria. La forma más común de este tipo de sistemas es el

denominado multiprocesador simétrico (SMP). En un SMP, varios procesadores comparten una única memoria mediante un bus compartido u otro tipo de mecanismo de interconexión. Una característica distintiva de estos sistemas es que el tiempo de acceso a memoria principal es aproximadamente el mismo para cualquier procesador (Herbert, 2015). En la Figura 8 se presenta la arquitectura UMA, la cual dispone de un mecanismo común de buses interconectados a memoria compartida.

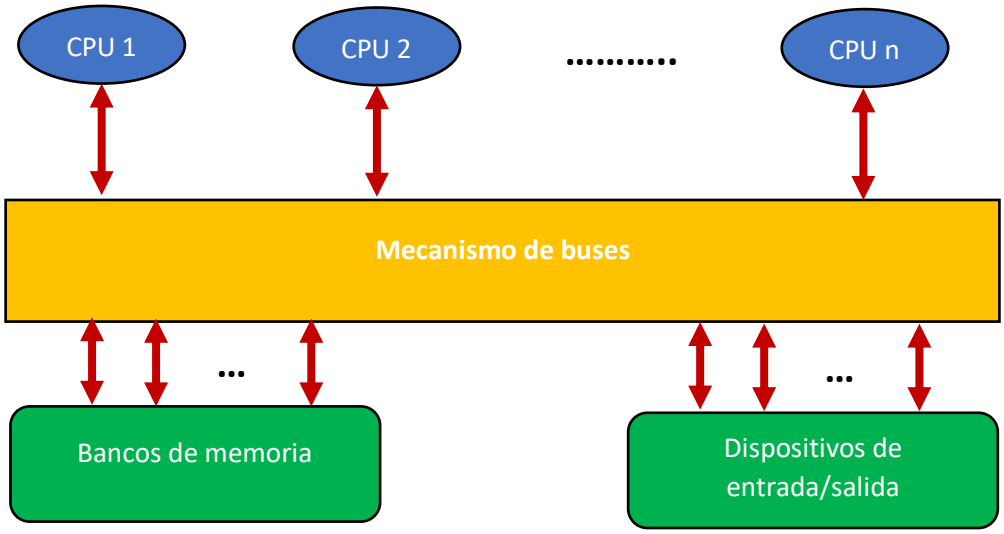


Figura 8: Arquitectura UMA
(Herbert, 2015)

La implementación más simple de un multiprocesador UMA es un procesador de doble núcleo, en el que dos CPU en un mismo chip comparten una única ruta a la memoria. Es muy común en computación de alto rendimiento usar más de un procesador en un nodo de cómputo, ya sea de un solo núcleo o de varios núcleos. En la Figura 9 se presenta la arquitectura de un computador UMA, dos procesadores (de un solo núcleo), cada uno se comunica y accede a la memoria a través de un bus común.

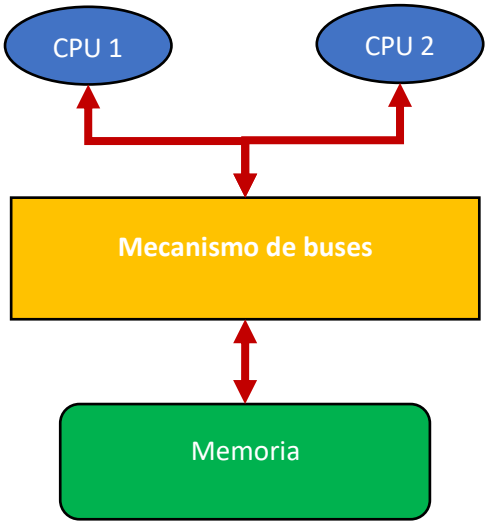


Figura 9: Arquitectura UMA con dos procesadores de un núcleo

(Hager & Wellein, 2010)

En la Figura 10 se presenta la arquitectura de un multiprocesador UMA con dos procesadores de doble núcleo que se conectan por separado hacia el mecanismo de buses y luego por separado hacia la memoria principal.

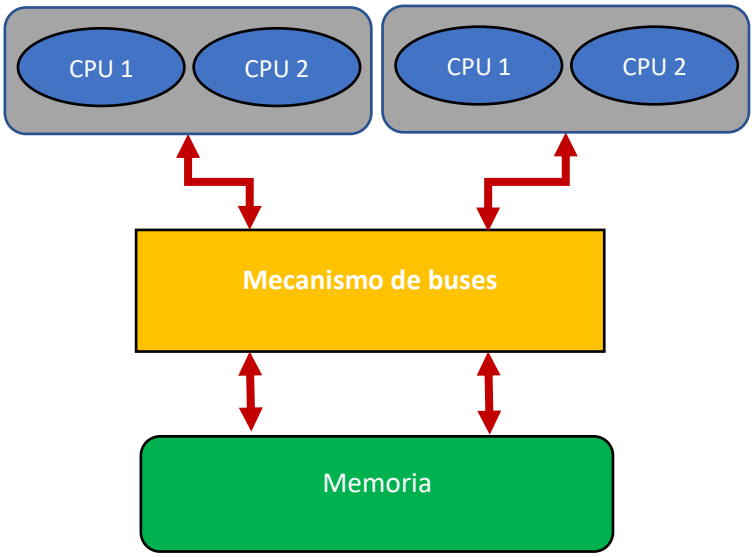


Figura 10: Arquitectura UMA con dos procesadores de un núcleo

(Hager & Wellein, 2010)

NUMA (Non-Uniform Memory Access): son computadores de acceso no uniforme a memoria. En este tipo de multiprocesadores la memoria está físicamente distribuida pero lógicamente compartida. Debido a la naturaleza distribuida, el rendimiento del acceso a la memoria varía en función del procesador que accede a sectores de la memoria (Hager & Wellein, 2010). En la Figura 11 se presenta la arquitectura NUMA en la cual el espacio de direccionamiento es compartido a pesar de que la memoria es distribuida. El tiempo de acceso a memoria depende del acceso local al procesador.

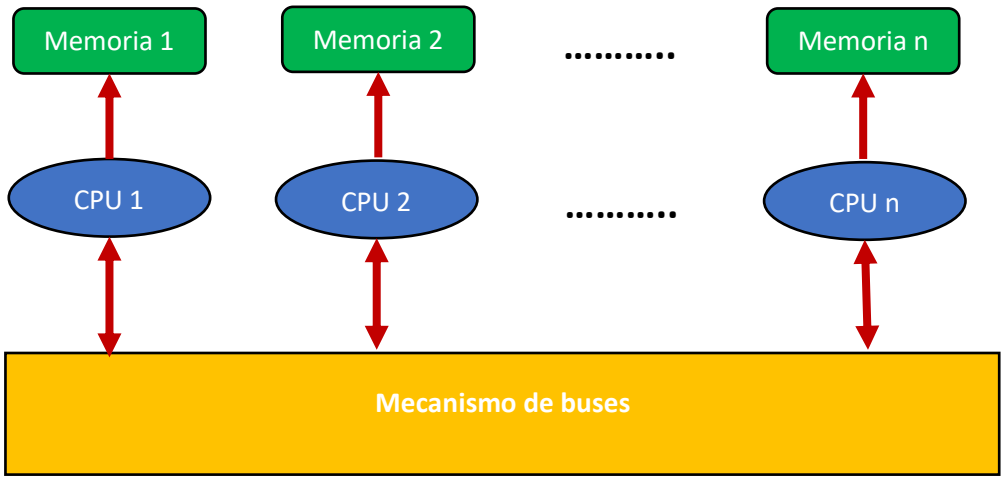


Figura 11: Arquitectura NUMA
(Herbert, 2015)

2.2.3 Procesadores vectoriales lineales

El pipelining (tubería) es una técnica bien conocida para utilizar varios elementos de hardware al mismo tiempo. Por lo tanto, el rendimiento del sistema y la capacidad de rendimiento general del sistema aumentan al aumentar el número de instrucciones que se pueden completar por segundo. La intención de esta técnica es subdividir una tarea computacional en varias subtareas, cada una puede ser ejecutada por una etapa de hardware separada. Los cálculos que se realizan en el proceso de programación deben sincronizarse en un tiempo determinado para evitar los tramos más recargados que se detectan entre dos registros.

Segmentar los cálculos, permite mejorar la frecuencia de trabajo. Este tipo de flujo de datos implica que la salida de una fase es una entrada de otra. Así, los diversos tramos o fases se encadenan a la manera de una tubería, logrando agilizar el flujo a través de este pipeline (School of Science and Technology, 2016).

En la Figura 12 se presenta la implementación de pipelining con cuatro fases, en las que cada fase opera simultáneamente con las otras en una de estas subtareas de varias instrucciones. Este proceso continúa hasta que se ejecutan todas las instrucciones.

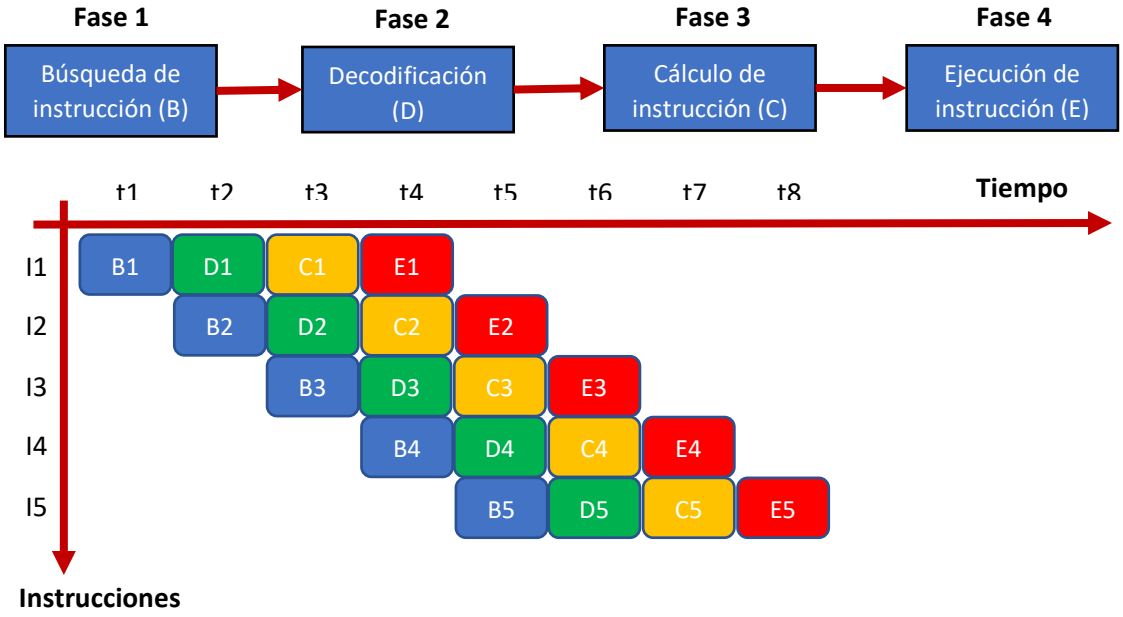


Figura 12: Proceso de pipelining
(School of Science and Technology, 2016)

2.2.4 Arreglo de procesadores

Una versión de computadoras paralelas son los arreglos de procesadores. Un arreglo de procesadores es de tipo SIMD y consta de un número de procesadores idénticos y relativamente elementales conectados juntos para formar una matriz. Cada procesador activo (algunos pueden estar inactivos) obedece la misma instrucción, emitida por una unidad de control, pero en diferentes datos locales. Como hay una sola unidad de control, hay un solo programa que opera en todos los datos a la vez. Por lo tanto, no hay dificultades

con la sincronización de los procesadores. procesadores de matriz son particularmente adecuados para cálculos que involucran manipulaciones de matrices (School of Science and Technology, 2016).

En la Figura 13 se presenta un arreglo de 4 procesadores, cada procesador tiene memoria local y una unidad de control compartida. Cada elemento de procesamiento tiene registros internos y memoria local para almacenar datos.

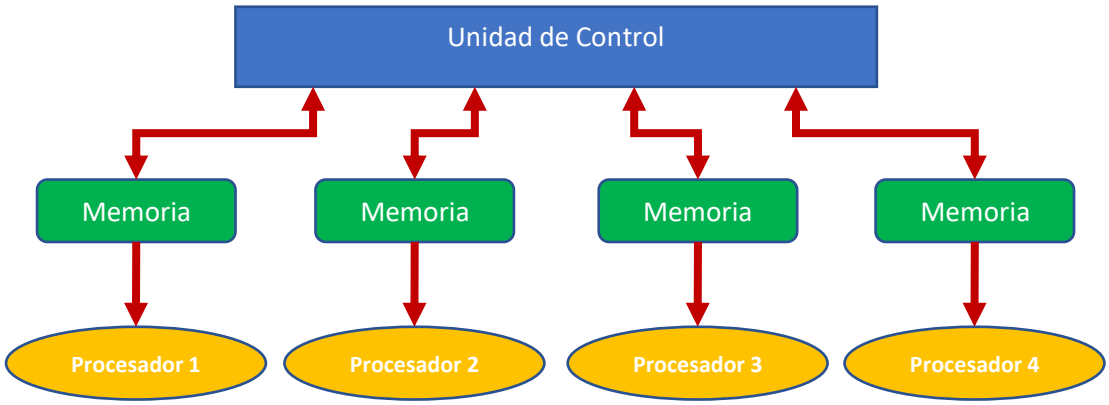


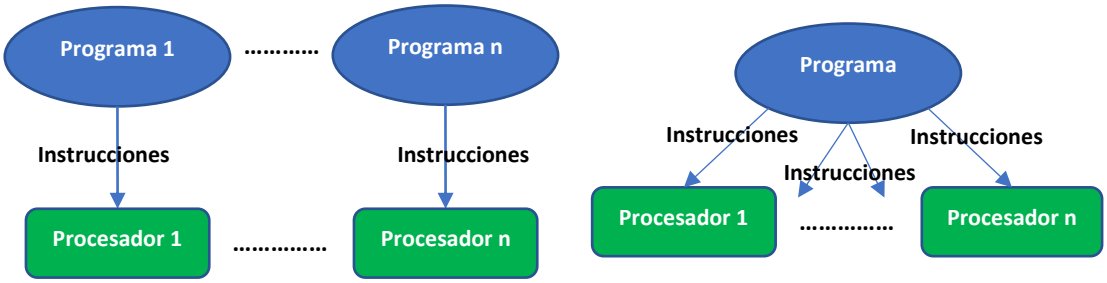
Figura 13: Ejemplo de un arreglo de procesadores.
(School of Science and Technology, 2016).

2.3 Modelos de paralelismo

2.3.1 **Paso** de **mensajes**

El modelo de programación de paso de mensajes está basado en la abstracción de un sistema paralelo con un espacio de direcciones distribuido, donde cada procesador tiene una memoria local, que es únicamente accesible desde ese procesador. El intercambio de datos se debe realizar a través del paso de mensajes. Para transferir datos de la memoria local de un procesador a otro, el primero debe enviar los datos al segundo y el segundo recibe los datos que serán almacenados en su memoria local. Para garantizar la portabilidad de los programas se asume que todos los procesadores pueden enviar mensajes a todos los procesadores de forma directa (Fernández, 2012).

Un programa basado en paso de mensajes ejecuta varios procesos, cada uno de los procesos tiene sus propios datos locales. Normalmente, un proceso se ejecuta en un núcleo (core) de la máquina. El número de procesos a ejecutar se especifica al iniciar el programa, cada proceso puede acceder a sus datos locales y puede intercambiar sus datos locales con otros procesos por medio del envío y recepción de mensajes. En teoría, cada uno de los procesos puede ejecutar un programa diferente (MPMD, multiple program multiple data), pero por facilidad de programación, se asume que cada uno de los procesos ejecuta el mismo programa (SPMD, single program, multiple data). En la práctica, esto no es realmente una restricción, ya que cada proceso puede ejecutar diferentes partes del programa, dependiendo, por ejemplo, del rango del proceso. Los procesos que usan el modelo de paso de mensajes pueden intercambiar datos usando las operaciones de comunicación, como, por ejemplo, operaciones de transferencia punto-a-punto o de comunicación global como broadcast (Hager & Wellein, 2010). En la Figura 14 se presenta un ejemplo del esquema que podría tener una arquitectura de paso de mensajes.



a) Arquitectura MPMD b) Arquitectura SPMD

Figura 14: Arquitecturas basadas en paso de mensajes
(Weiss, 2014)

2.3.2 Memoria compartida

El paradigma de memoria compartida significa que todos los procesos comparten un espacio único de direcciones de memoria. Una forma natural de programar en este tipo de arquitectura es con un modelo basado en threads en el que todos tienen acceso a variables compartidas. Estas variables serán usadas como medio de intercambio de información entre threads. Para coordinar

el acceso a las variables compartidas se utilizan mecanismos de sincronización con el objetivo de evitar condiciones de carrera en caso de accesos concurrentes. Los paradigmas de programación en memoria compartida pueden variar de unos a otros con respecto a los modelos de concurrencia, las variables compartidas y el soporte para sincronización (Hager & Wellein, 2010; Naiouf et al., 2012). En la Figura 15 se presenta el esquema de funcionalidad del paralelismo con memoria compartida.

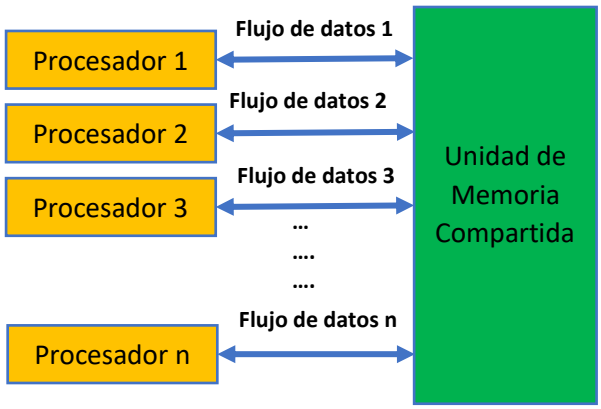


Figura 15: Esquema de paralelismo mediante memoria compartida (Hager & Wellein, 2010)

2.3.3 Paralelismo de Hilos

Un hilo es un flujo de instrucciones dentro del programa que pueden ser ejecutadas al mismo tiempo. El modelo de programación basado en hilos ofrece ventajas importantes sobre el modelo inspirado en el paso de mensajes, pero como en todo también existen desventajas (Hager & Wellein, 2010). Algunas de las características más importantes:

- Portabilidad de software: las aplicaciones basadas en threads pueden ser desarrolladas en máquinas secuenciales y ser ejecutadas en máquinas paralelas sin necesidad de hacer ningún cambio. La habilidad para migrar programas entre diversas plataformas es una ventaja muy importante en favor de las API's basadas en threads, debido a que la capacidad de acceso a un supercomputador puede ser limitado y costoso, y no se puede estar malgastando en tiempo de desarrollo.

- Ocultación de latencia: uno de los mayores costes que tienen los programas (tantos secuenciales como paralelos) es la latencia de acceso, tanto a memoria como a dispositivos I/O. Puesto que se ejecutan múltiples hilos en el mismo procesador, las aplicaciones basadas en hilos permiten ocultar esta latencia. De hecho, mientras un hilo está esperando a que termine una operación de comunicación, los otros hilos pueden utilizar la CPU, ocultando de esta manera la espera.

- Planificación y balanceo de carga: cuando se están escribiendo programas usando memoria compartida, el programador debe expresar la concurrencia de manera que el coste de sincronización sea mínimo y que los procesadores se mantengan ocupados la mayoría del tiempo

En varias aplicaciones con datos estructurados la tarea de asignar la misma cantidad de trabajo a los procesos es sencilla, pero en aplicaciones con datos sin estructura y de carácter dinámico esta tarea resulta mucho más complicada. Las API (Aplication Programming Interfaces - Interfaces de programación de Aplicaciones) basadas en hilos permiten al programador especificar un gran número de tareas concurrentes y también soportan la asignación dinámica de tareas a procesadores de forma que se minimiza el tiempo que un núcleo (core) pasa desocupado. Esta capa de abstracción a nivel de sistema evita que el programador tenga que preocuparse de forma explícita de una planificación y un balanceo de carga (Hager & Wellein, 2010; Slabaugh et al., 2010). En la Figura 16 se presenta el funcionamiento del paralelismo mediante hilos. Con el paralelismo de hilos, es posible aprovechar la capacidad de los procesadores modernos para ejecutar múltiples tareas en paralelo, lo que puede mejorar significativamente el rendimiento de las aplicaciones. Sin embargo, la programación multihilo también puede presentar desafíos, como la necesidad de sincronizar los hilos para evitar conflictos en los datos compartidos y la posibilidad de errores de concurrencia

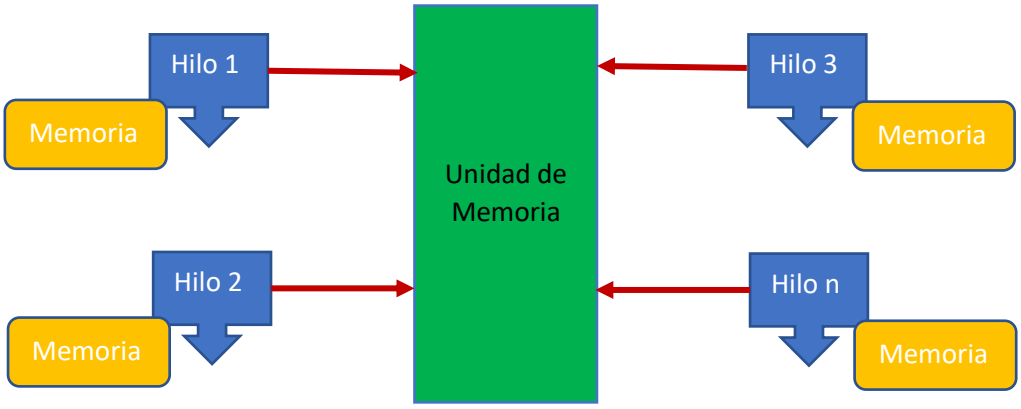


Figura 16: Esquema de paralelismo mediante hilos (Hager & Wellein, 2010).

2.3.4 Paralelismo de datos

Diversas aplicaciones de software que procesan grandes cantidades de datos y que, por lo tanto, tienen un tiempo de ejecución muy grande se dedican a modelar fenómenos del mundo real. Por ejemplo, las imágenes y los fotogramas de vídeo son “capturas” de un instante donde las diferentes partes de la fotografía capturan eventos físicos independientes y simultáneos. Se puede diseñar un algoritmo de manera que la misma operación se aplique de forma concurrente sobre un conjunto de elementos de una estructura de datos. La concurrencia está en los datos; es decir, una operación se puede realizar de forma simultánea en varios elementos de las estructuras de datos, con la seguridad de que no se alterará el resultado final (Herbert, 2015; Naiouf et al., 2020). En la Figura 17 se presenta un esquema de funcionalidad del paralelismo de datos. Los datos se dividen en múltiples fragmentos y se procesan de manera simultánea mediante varios procesadores disponibles compartiendo memoria mientras realizan las tareas encomendadas.

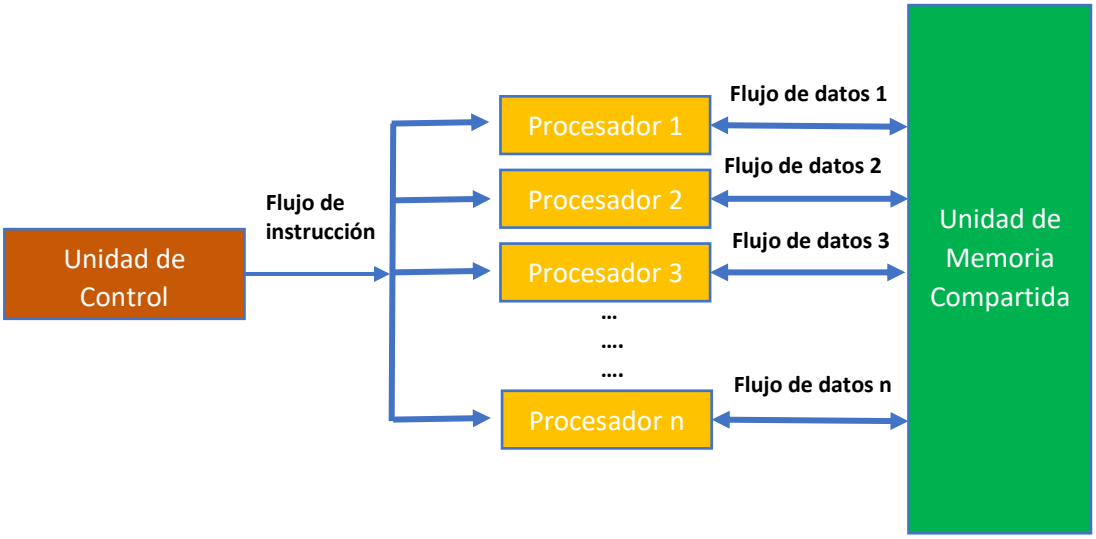
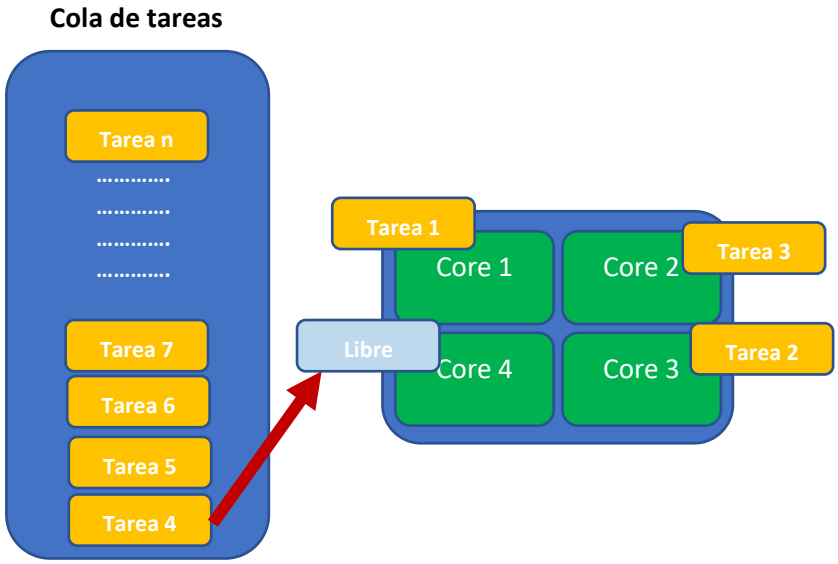


Figura 17: Esquema de paralelismo de datos

(Czech, 2017)

2.3.5 Paralelismo de tareas

En un modelo de programación basado el paralelismo de tareas, el programador debe definir y manipular tareas concurrentes. Los problemas se descomponen en tareas que pueden ser ejecutadas de forma concurrente, y entonces asignarlas a hilos (threads) para que se ejecuten en paralelo. Esto resulta muy sencillo cuando las tareas no dependen entre ellas, pero en este modelo de programación también se usa cuando las tareas comparten datos entre sí. Existen dos elementos a ser considerados: coherencia de datos y balanceo de carga. Una categoría especialmente importante del paralelismo de tareas está



dedicado a mantener las dependencias de datos. Debido a que la ejecución de un conjunto de tareas finaliza cuando la última ha terminado se debe tener en cuenta que las tareas pueden tener requisitos significativamente diferentes, por lo que su distribución para que todas lleguen a su fin al mismo tiempo es un proceso difícil. Este es el problema del balanceo de carga (Fang et al., 2020; Slabaugh et al., 2010). En la Figura 18 se presenta el funcionamiento del paralelismo mediante tareas, las tareas se ejecutan de manera simultánea en diferentes núcleos del procesador, lo que permite mejorar significativamente el rendimiento y la eficiencia de los algoritmos informáticos.

Figura 18: Esquema paralelismo mediante tareas
(Hager & Wellein, 2010)

2.4 Medidas de rendimiento paralelo

Los algoritmos de manera general son conjuntos ordenados y finitos de pasos para solucionar problemas computacionales considerando entradas de datos, proceso y salida de resultados. El diseño de algoritmos paralelos se basa en identificar tareas que sean susceptibles a ser ejecutadas concurrentemente, y posteriormente asignar dichas tareas a diferentes hilos o procesos del sistema operativo (Alonso & Lvarruiz, 2018).

Un algoritmo paralelo describe cómo se puede resolver el problema en una arquitectura paralela dada, dividiendo el problema en subproblemas, comunicándose entre procesadores y uniendo las soluciones parciales para producir el resultado final. Un buen algoritmo paralelo resulta de buscar el paralelismo que podría ser inherente a un algoritmo secuencial para un problema dado (Leibovich et al., 2012; Naiouf et al., 2020; Universidad Europea de Madrid, 2017). Hay dos enfoques en el diseño de algoritmos paralelos con respecto al número de procesadores disponibles. El primero es diseñar un algoritmo en el que el número de procesadores utilizados por el algoritmo es un parámetro de entrada, lo que significa que el número de procesadores no depende del tamaño de entrada del problema. El segundo enfoque es permitir que la cantidad de procesadores utilizados por el algoritmo paralelo crezca con el tamaño de la entrada, lo que significa que la cantidad de procesadores no es un parámetro de

entrada sino una función del tamaño de entrada (Universidad Europea de Madrid, 2017).

La computación paralela intenta aprovechar la concurrencia disponible en las arquitecturas de computadoras actuales, las cuales permiten ejecutar varias operaciones de manera simultánea, procesar varios hilos de ejecución concurrentemente, o incluso ejecutar al mismo tiempo diferentes aplicaciones en varios procesadores (Bombieri et al., 2012; Ravi et al., 2012). Las arquitecturas paralelas actualmente están presentes en laptops, servidores e incluso en teléfonos móviles permitiendo realizar aplicaciones mediante algoritmos implementados en lenguajes de programación paralela, cuyo objetivo principal es acelerar el cálculo computacional obteniendo respuesta en tiempos menores a los obtenidos por los algoritmos secuenciales (L. Wang et al., 2015). En la Figura 19 se presenta el esquema funcional básico del paralelismo, las tareas se dividen en unidades más pequeñas que se ejecutan simultáneamente en múltiples núcleos de procesamiento para realizar tareas en forma paralela, lo que puede mejorar significativamente el rendimiento y la eficiencia de los sistemas informáticos.

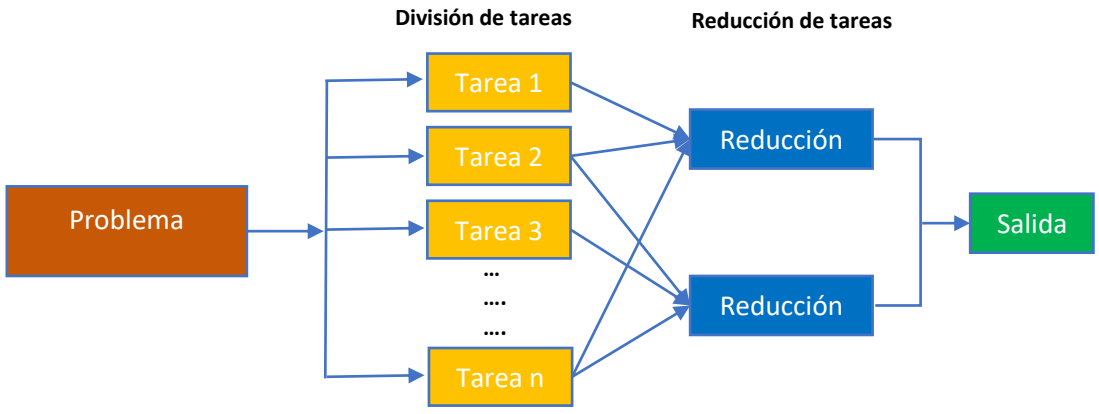


Figura 19: Modelo básico de paralelismo
(Czech, 2017)

Existen una serie de restricciones en el diseño de algoritmos paralelos que no están presentes en el diseño de algoritmos secuenciales. Estas limitaciones tienen que ser destacadas junto con el desarrollo de varias medidas de desempeño para algoritmos paralelos. En algoritmos con paralelismo la ley de

Amdahl facilita encontrar la mejora máxima esperada en una arquitectura de computadora o sistema informático cuando solo se mejoran ciertas partes del sistema mediante el aumento teórico de velocidad máxima que se logra utilizando múltiples procesadores (Herbert, 2015). En la Figura 20 se presenta la aceleración de un algoritmo con referencia al número de núcleos utilizados en la ejecución de un algoritmo. Todo algoritmo no es 100% paralelizable por características propias del diseño y funcionalidad, algunos algoritmos tienen dependencias entre las tareas que los componen. Ciertos lenguajes de programación incluyen sentencias de paralelismo especialmente para ser utilizados en operaciones con ciclos.

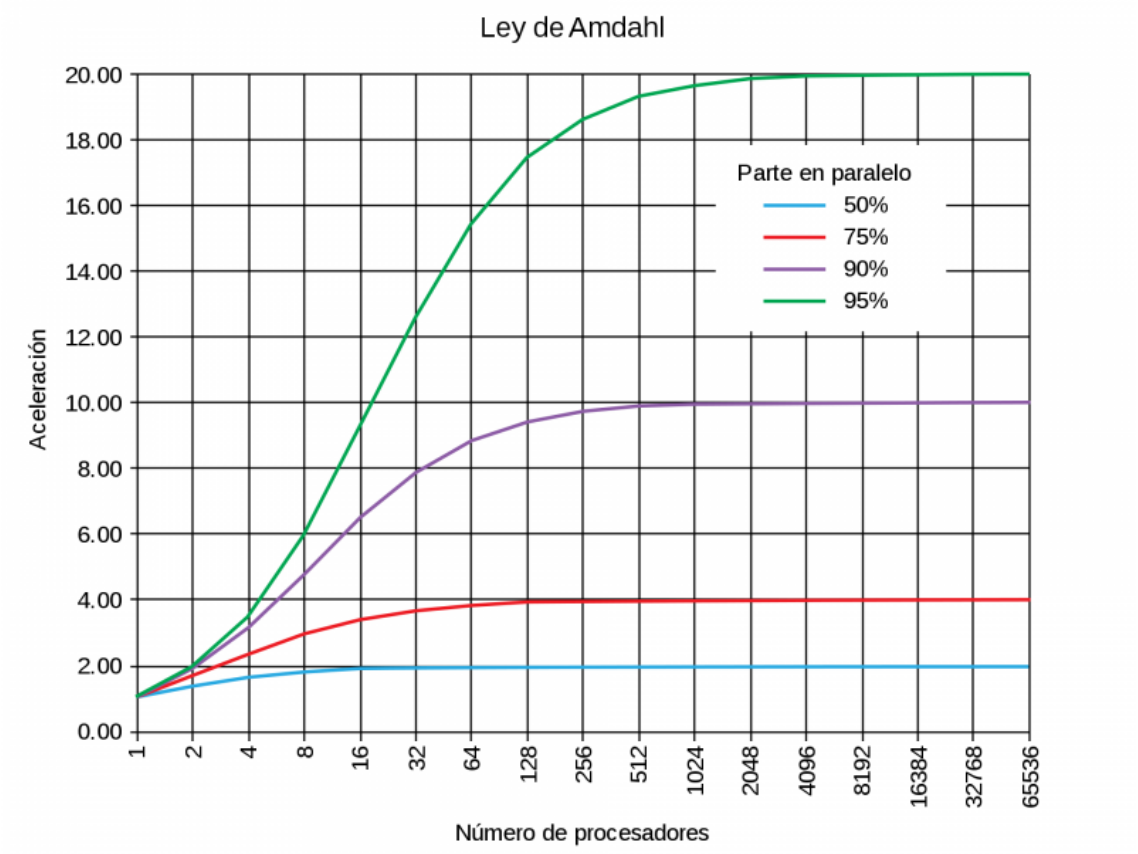


Figura 20: Tiempos de ejecución algoritmos con paralelismo (Herbert, 2015)

2.4.1 Limitaciones del paralelismo

Las limitaciones de un algoritmo están asociadas al grado de paralelismo. No todos los algoritmos permiten ser paralelizados pues el paralelismo implica la ejecución simultánea de tareas, cuya factibilidad depende de las dependencias existentes entre la información y los resultados intermedios. Así, un algoritmo inherentemente secuencial impone una restricción insalvable para su realización paralela. Por otra parte, cada algoritmo tiene sus propias características que hacen que su paralelización sea más o menos beneficiosa (Zaripov et al., 2018).

Existen limitaciones de implementación cuando un procesador no puede dedicar tiempo a realizar cómputo provechoso. Estas penalidades surgen cuando el procesador ejecuta una operación de comunicación o cuando queda ocioso (Herbert, 2015).

- ✓ Necesidad de comunicación: esta es una penalidad propia al modelo paralelo si se considera que los procesadores tienen necesidad de comunicarse para resolver un problema cooperativamente.
- ✓ Tiempo ocioso: ocurre cuando un procesador no puede realizar cómputo productivo, ya sea porque no tiene más tareas para ejecutar, porque su tarea actual se encuentra suspendida esperando una sincronización, o porque los datos que necesita para continuar con su operación no han arribado. La estrategia de utilización plena de todos los procesadores se denomina balanceo de carga.

Si bien estas limitaciones no pueden ser eliminadas en su totalidad, diversas estrategias tales como el correcto balanceo de carga de los procesadores o el uso de protocolos no bloqueantes para las comunicaciones, permiten mejorarlas para ofrecer un mayor desempeño del sistema paralelo (Hager & Wellein, 2010).

2.4.2 Tiempos de ejecución

Cuando se aborda la solución de un problema computacional se tiene la disyuntiva de seleccionar entre varios algoritmos tomando en cuenta fundamentalmente los siguientes aspectos: el algoritmo fácil de entender, codificar, depurar versus el algoritmo que se ejecute con la mayor rapidez posible, siempre con el objetivo de hacer un uso eficiente de los recursos de la máquina (Azzini et al., 2018; Pusdá-Chulde, De Giusti, et al., 2021; Salza & Ferrucci, 2019). La razón principal del diseño de algoritmos paralelos es usarlo de tal manera que una tarea computacional pueda ser completada rápidamente. El tiempo de ejecución es un buen indicador de la funcionalidad de un algoritmo paralelo. La medición del tiempo de ejecución utilizada está representada por el conteo de las unidades de segundos comprendidas entre el momento de inicio y termino de ejecución (Díaz, 2018).

Considerando que un algoritmo puede ser implementado para correr en forma secuencial/paralela y asumiendo que el mismo tipo de procesador es utilizado para los dos tipos de implementación, debería esperarse que el tiempo requerido para resolver el problema computacional decrezca a medida que se agregan más procesadores. De esta manera, el tiempo de cómputo de un algoritmo que se ejecuta en una computadora con un único procesador puede reducirse hasta N veces (en el caso ideal) cuando este esfuerzo es distribuido en un ambiente de N procesadores. Si N es suficientemente grande, la implementación paralela de un algoritmo podría resolver problemas en mínimas fracciones de tiempo considerando los límites dados por la fracción secuencial del mismo (Universidad Europea de Madrid, 2017). En la Figura 21 se presenta el tiempo paralelo, el cual puede denominarse tiempo moderado si fuese directamente proporcional el tiempo de acuerdo con el número de procesadores, en comparación con el tiempo real de cómputo asociado a un problema típico.

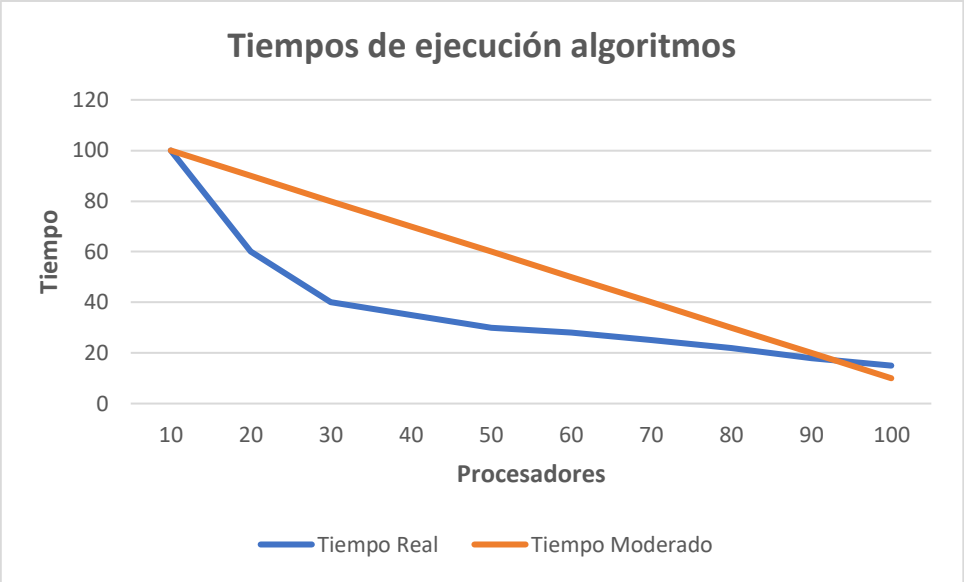


Figura 21: Tiempos de ejecución algoritmos con paralelismo (Pusdá-Chulde, De Giusti, et al., 2021)

2.4.3 Aceleración (SpeedUP)

La aceleración de un algoritmo en implementación paralelo con respecto a la implementación secuencial se denomina SpeedUP (Cantidad de mejora global del rendimiento de un sistema) y es una magnitud adimensional. Las ecuaciones 1, 2, 3 muestran el cálculo de la aceleración, donde U son las unidades de cómputo, N el número de elementos de proceso por unidad, R(N) el rendimiento máximo teórico, S(N) SpeedUP o aceleración, TS tiempo secuencial (Tiempo desde el inicio hasta que el último elemento de procesamiento finaliza su ejecución), TP(N) tiempo paralelo para N procesadores. La aceleración real de un algoritmo se realiza mediante la ley de Amdahl (Universidad Europea de Madrid, 2017), la cual explica cómo se calcula el tiempo en paralelo de un algoritmo y en base a esto predice el comportamiento real de la aceleración del mismo (Salza & Ferrucci, 2019; Zaripov et al., 2018).

$$R(N) = UxN \quad (1)$$

$$S(N): \frac{TS}{TP(N)} \quad (2)$$

$$E(N) = \frac{S(N)}{N} = \frac{TS}{NxTP(N)} \quad (3)$$

En la ecuación 4 se presenta la ley de Amdahl, en donde f es la parte secuencial del código, reemplazando la ecuación (4) en (2) se obtiene la ecuación (5) y haciendo un límite sobre esta, teniendo unidades de proceso a infinito se obtiene la ecuación (6). La ecuación 6 significa que la aceleración real SR(N) de un algoritmo tiende a un valor constante, esto se debe a que a medida que incrementan unidades de cómputo, el tiempo de procesamiento disminuye, pero los tiempos de transferencia y comunicación entre las unidades aumenta.

$$TP(N) = f\ x\ T + (1 - f)\ x\ \frac{TS}{N} \qquad (4)$$

$$S(N) = \frac{N}{f(N-1)+1} \qquad (5)$$

$$SR(N) = \lim_{n\to\infty} \frac{N}{f(N-1)+1} = \frac{1}{f} \qquad (6)$$

2.4.4 Eficiencia

En computación se considera eficiencia (E) a la Fracción de tiempo en que los recursos computacionales están siendo utilizados de forma productiva (López-Granados et al., 2019; Osio et al., 2017). La medición de la eficiencia está determinada por la ecuación 7.

$$E(N) = \frac{S(N)}{N} = \frac{TS}{NxTP(N)} \qquad (7)$$

Donde S(N) es la rapidez previamente calculada del algoritmo y N es el número de procesadores utilizados en la ejecución del algoritmo.

En algoritmos mediante paralelismo se obtiene la mejor eficiencia considerando el caso homogéneo, es decir un conjunto de procesadores idénticos. En condiciones habituales el speedup máximo que puede lograrse con varios procesadores es **(N)** y se denomina speedup lineal. Si la implementación paralela alcanza un valor superior a **(N)** se puede estar en un caso donde no se ha testeado el algoritmo paralelo frente al mejor secuencial o bien existe alguna diferencia en el desempeño de los procesadores. Sin embargo, en algunas

arquitecturas paralelas pueden suceder que aun haciendo una medición correcta se supere el valor de **(N),** en este caso se considera como speedup superlineal. La situación más común en que puede darse es cuando se aprovecha la mayor cantidad de memoria disponible del sistema multiprocesador (Hager & Wellein, 2010; Herbert, 2015).

2.4.5 Costo computacional

El costo computacional C(N) de un algoritmo paralelo es el límite sobre un total de operaciones ejecutadas colectivamente por los N procesadores (Weiss, 2014). La medición del costo está determinada por la ecuación 8.

$$C(N) = \frac{TS}{E(N)} \qquad (8)$$

Donde TS es el tiempo de ejecución del algoritmo secuencial y E(N) es la eficiencia obtenida dependiendo del número de núcleos.

2.4.6 Escalabilidad

La escalabilidad es la medida que evalúa el comportamiento de un programa paralelo, cuando incrementa el tamaño del problema proporcionalmente al número de procesadores, o incrementa el número de procesadores sin variar el tamaño del problema a tratar.

En la Figura 22 se presenta dos medidas de escalabilidad que se pueden hacer, según el punto de vista del tamaño total del problema a tratar (línea naranja), y el tamaño del problema a tratar por procesador (línea roja). Así, la **Escalabilidad Débil** es una medida que mantiene fijo el tamaño que le toca a cada procesador, por consiguiente, el tamaño aumenta con el número de procesadores. Por el contrario, la **Escalabilidad Fuerte** es una medida que mantiene el tamaño fijo del problema a tratar para cualquier número de procesadores a tratar.

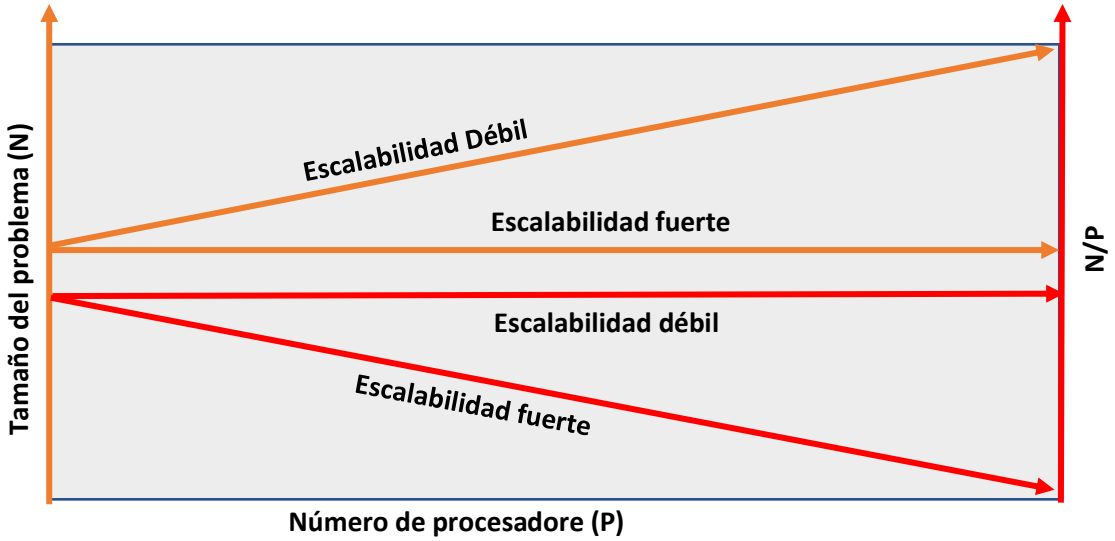


Figura 22: Medidas de escalabilidad paralela

(Gonzáles Itúrbide, 2016)

2.5 Arquitecturas heterogéneas

Las arquitecturas de hardware con núcleos heterogéneos actualmente son una parte integral de los sistemas informáticos modernos que van desde computadoras normales hasta supercomputadoras. Si bien el diseño heterogéneo de muchos núcleos ofrece el potencial para un alto rendimiento energéticamente eficiente, dicho potencial solo puede desbloquearse si los programas son adecuadamente paralelizados en todos los niveles y pueden adaptarse a la plataforma heterogénea subyacente (De Giusti et al., 2020; Fang et al., 2020; Libutti et al., 2020).

Uno de los cambios de mayor impacto ha sido el uso de manera masiva de procesadores con más de un núcleo (Multicore), produciendo plataformas distribuidas híbridas (memoria compartida y distribuida) generando la necesidad de desarrollar sistemas operativos, lenguajes y algoritmos que las usen adecuadamente. También incrementó la incorporación de placas aceleradoras a los sistemas multicore constituyendo plataformas paralelas de memoria compartida con paradigma de programación propio asociado como pueden ser las unidades de procesamiento gráfico (GPU, Graphic Processing Unit) de NVIDIA y AMD, los coprocesadores Xeon Phi de Intel o los aceleradores basados en circuitos integrados reconfigurables (FPGAs, Field Programmable Gate Array) (Milla & Rucci, 2022; Yang et al., 2011). En la Figura 23 se presenta algunos

modelos de arquitecturas programables que utilizan múltiples procesadores en el mismo circuito integrado, para dar un mejor procesamiento a diferentes aplicaciones informáticas (renderizado en 3D, cálculos matemáticos intensivos con conjuntos de datos muy grandes).

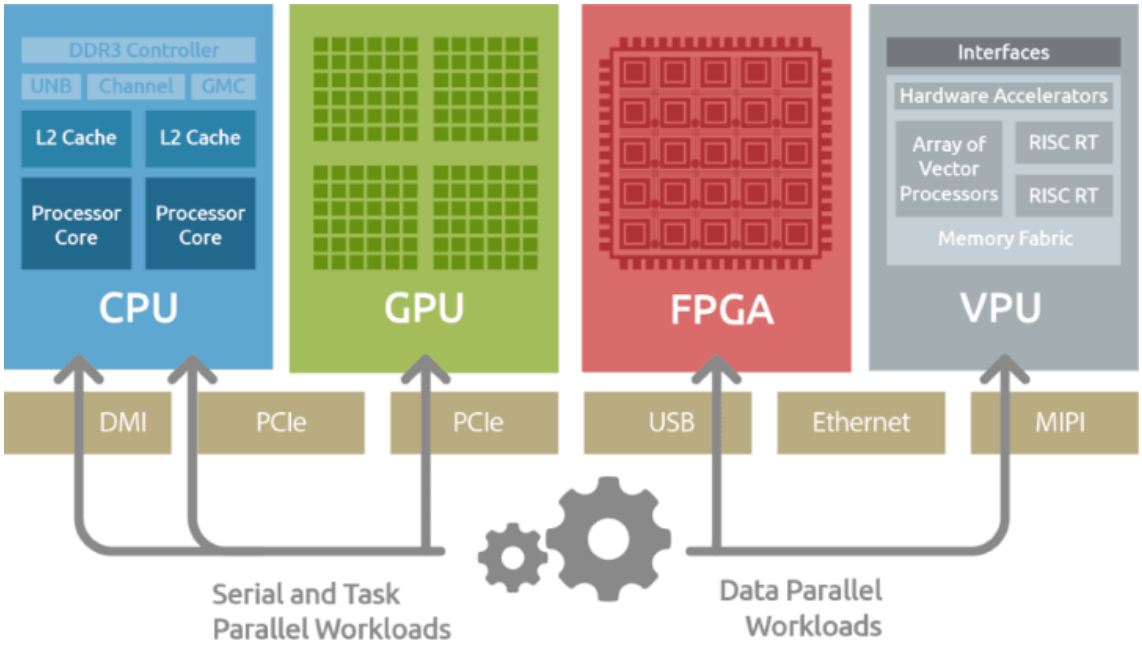


Figura 23: Arquitecturas heterogéneas programables (Fang et al., 2020).

2.5.1 Multicore

Esta arquitectura combina dos o más procesadores (núcleos o cores), en un mismo chip. Estos procesadores son utilizados en el desarrollo de aplicaciones con procesamiento paralelo. Las aplicaciones paralelas están compuestas por múltiples hilos independientes, de forma que es posible la concurrencia. Es decir, los hilos se pueden ejecutar al mismo tiempo de manera individual o de forma paralela. Como consecuencia el rendimiento de las aplicaciones paralelas puede teóricamente escalar linealmente con el número de procesadores (De Giusti et al., 2020; Dufrechou, 2018; Fang et al., 2020; Hager & Wellein, 2010; Naiouf et al., 2020; Pusdá-Chulde, De Giusti, et al., 2021). En la práctica existen factores que lo impiden, como los desbordamientos (overheads) por creación/eliminación de hilos, las comunicaciones entre las memorias de los procesadores, y el posible desbalanceo en las aplicaciones del volumen de cómputo por hilo (hilos esperando a que otros hilos finalicen. Como inconveniente, las aplicaciones

deben ser correctamente paralelizadas para aprovechar todo el potencial de los procesadores multinúcleo (Barlas, 2015a). En la Figura 24 se presenta el modelo de procesador con varios núcleos que comparten memoria local y adicionalmente comparten memoria principal. Los procesadores multicore dividen las tareas en subtareas que se procesan en paralelo en los diferentes núcleos. Cada núcleo tiene su propia memoria caché y comparte memoria principal para garantizar la coherencia de los datos. El resultado de las subtareas se combina para completar las tareas.

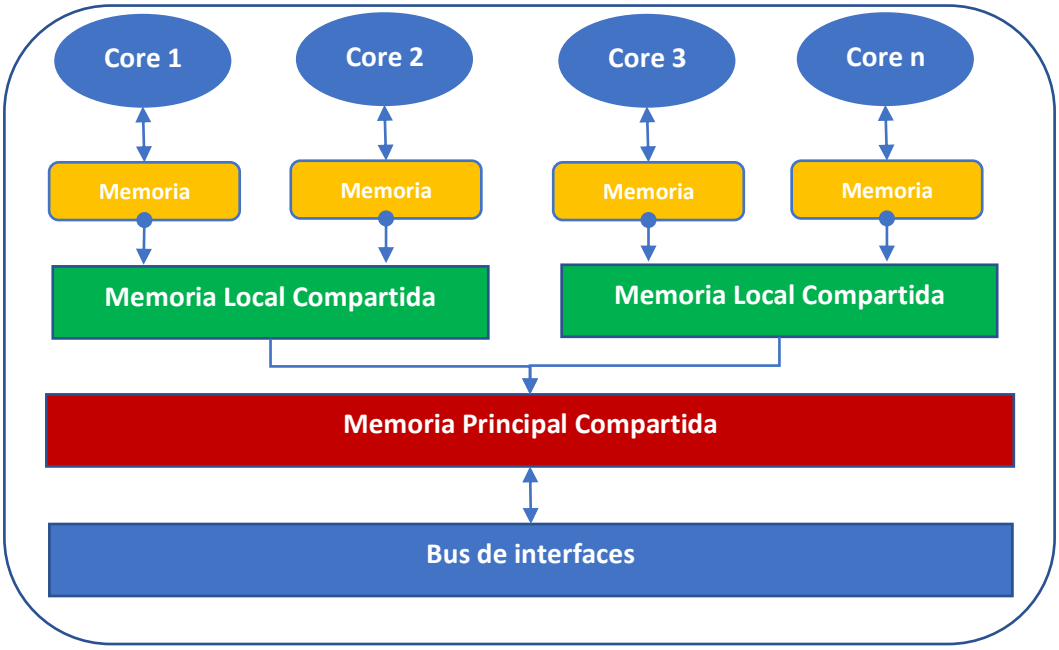


Figura 24: Diagrama genérico de un procesador multicore.
(Hager & Wellein, 2010)

2.5.2 GPU

Las unidades de procesamiento de gráficos (GPU), también conocidas como tarjetas de aceleración de gráficos, se han desarrollado como un medio para procesar una gran cantidad de datos gráficos muy rápidamente, mejorando el rendimiento con relación a las CPU convencionales. Las CPU emplean grandes cachés de memoria en el chip (y a veces múltiples), pocas unidades de procesamiento lógico y aritmético (ALU) complejas (por ejemplo, en cadena) y hardware de predicción y decodificación de instrucciones complejas para evitar el estancamiento mientras se espera que los datos lleguen desde la memoria

principal (Fang et al., 2020; Naiouf et al., 2020; Tabares Soto, 2016). En la Figura 25 se presenta el modelo de una GPU programable con varios procesadores que administran memoria y unidad de control local facilitando el trabajo de los núcleos de procesamiento en paralelo para acelerar el procesamiento de grandes cantidades de datos.

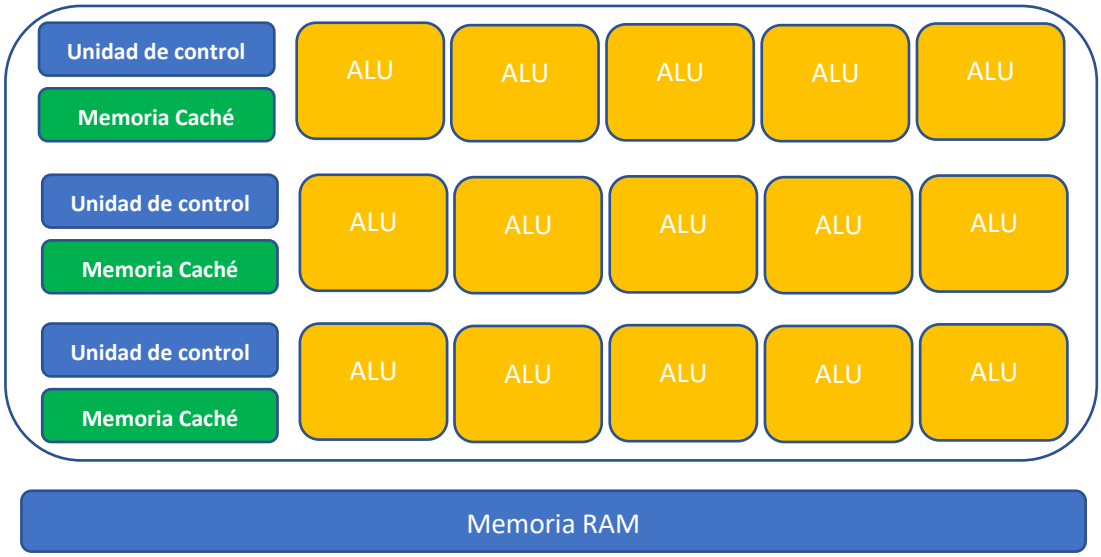


Figura 25: Diagrama genérico de una GPU
(Tabares Soto, 2016)

Actualmente existen diferentes modelos y marcas de GPU, estas siempre se componen par los mismos elementos en mayor o menor número, como los que se presentan a continuación (Barlas, 2015a):

- Procesador de comandos: lee la lista de pantalla o de instrucciones que le envía la CPU, tanto para generar gráficos como para hacer cálculos complejos.
- Unidad de Rasterizado: realiza la transformación del espacio tridimensional basado en vértices a uno bidimensional basado en píxeles.
- Unidad de Texturizado: encargada de aplicar una imagen sobre una superficie para simular textura o color en la vida real.

- Raster Output: encargada de dibujar los píxeles finales sobre el búfer de imagen. Es junto a la Cache de último nivel la única pieza que tiene permisos de escritura sobre la VRAM.

- Unidad Shader: dispone de un núcleo capaz de ejecutar programas pensados para manipular primitivas gráficas a tiempo real.

- Unidad de intersección: calcula la intersección de los rayos de la escena con los objetos. Es esencial para el Ray Tracing.

- Unidad de teselación: subdivide los vértices de los objetos para darles un aspecto más redondeada y pulido.

- CODEC de Vídeo: procesador independiente que descodifica vídeo en varios formatos multimedia y los reproduce, así como se encarga de generar vídeo e incluso pasar de un formato a otro.

- Interfaz de memoria: permite a la GPU leer de su memoria RAM, conocida como VRAM.

- DMA: permite a la GPU leer de la RAM principal del sistema.

A medida que va pasando el tiempo se agregan más elementos a una GPU como en las gráficas RTX de NVIDIA donde se han añadido elementos específicos para el procesamiento del trazado de rayos (RT Cores). Igualmente, también se han añadido en los últimos modelos de NVIDIA los Tensor Cores, que son los responsables de posibilitar el Machine Learning en tiempo real en los videojuegos que utilizan tecnología DLSS (Deep Learning Super Sampling – Super Muestreo de Aprendizaje Profundo), (Dufrechou, 2018; Fang et al., 2020). En la Figura 26 se presenta un esquema de computadores que incluyen interconectividad entre un CPU y GPU.

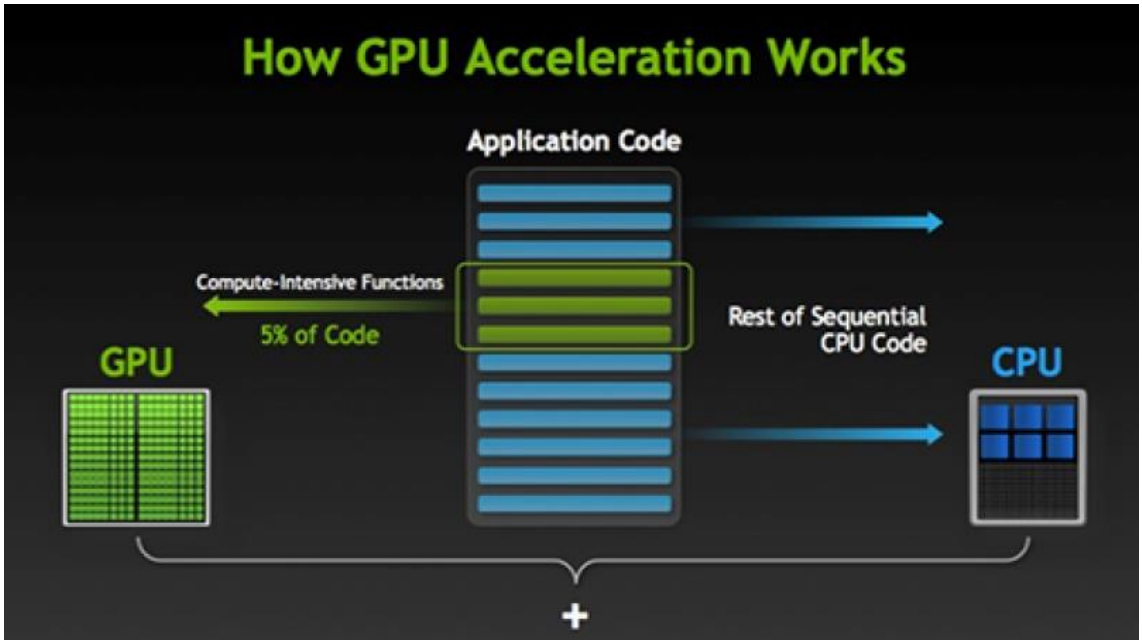


Figura 26: Arquitectura computadores con CPU y GPU
(Dufrechou, 2018)

2.5.3 Xeon Phi

Los procesadores Intel Xeon Phi se desarrollaron a partir de aquello que Intel había aprendido sobre aplicaciones multi núcleo durante el tiempo que estuvieron desarrollando Larrabee, su fallido intento de crear una tarjeta gráfica que estaba basada en la arquitectura interna x86. De hecho, fueron los primeros procesadores Xeon Phi los que introdujeron el concepto de Many Integrated Core (MIC), por el que los núcleos del procesador, en realidad, eran otros procesadores multinúcleo. Por ejemplo, los procesadores Intel Xeon Phi de la serie Knights Landing, tienen, en su interior, 72 procesadores Intel Atom de cuatro núcleos cada uno (INTEL, 2021). En la Figura 27 se presenta la arquitectura del procesador Intel Xeon Phi con 8 procesadores.

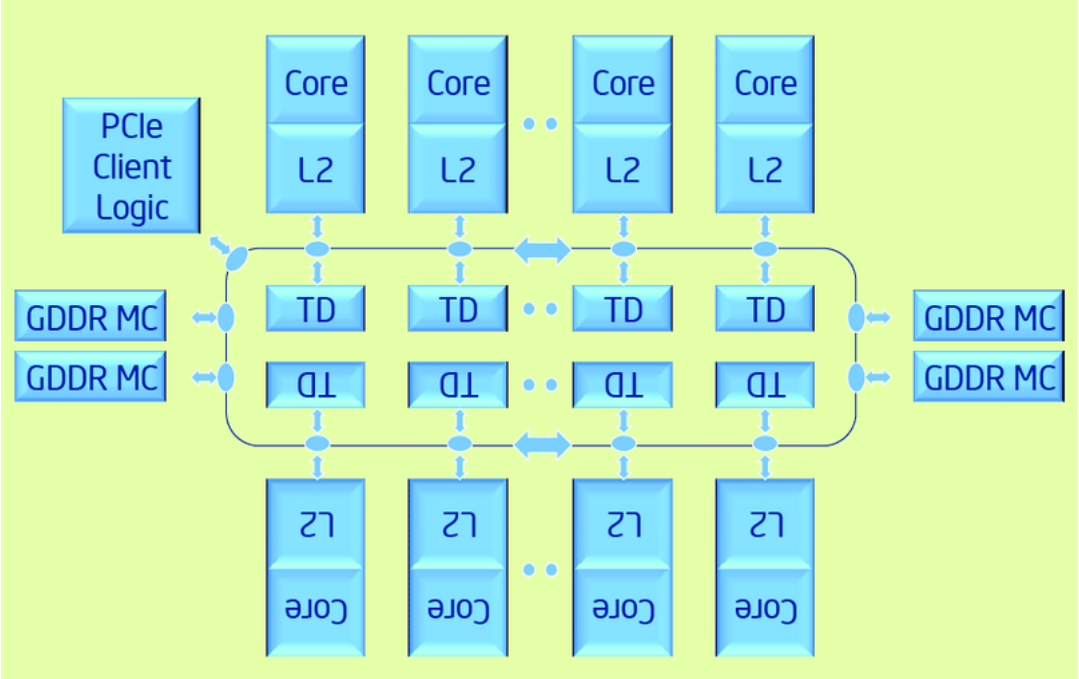


Figura 27:Diagrama de la Arquitectura Xeon Phi
(Atanassov et al., 2017)

2.5.4 Raspberry

Las Raspberry son computadoras de bajo costo, se trata de una placa base con funciones de mini PC, y que puedes adquirir a precios muy bajos. Raspberry incluye un sistema operativo Linux capaz de permitirle a las personas de todas las edades explorar la computación y aprender a programar lenguajes como Scratch y Python. Es capaz de hacer la mayoría de las tareas típicas de un computador de escritorio, desde navegar en internet, reproducir videos en alta resolución, manipular documentos de ofimática, hasta reproducir juegos. Además la Raspberry Pi tiene la habilidad de interactuar con el mundo exterior, puede ser usada en una amplia variedad de proyectos digitales, desde reproductores de música y video, detectores de patrones, estaciones meteorológicas hasta cajas de aves con cámaras infrarrojas (Garcia et al., 2018; Libutti et al., 2020). En la Figura 28 se presenta el diagrama estructural de un RaspBerry Pi con todos los componentes internos y puertos para dispositivos de entrada/salida.

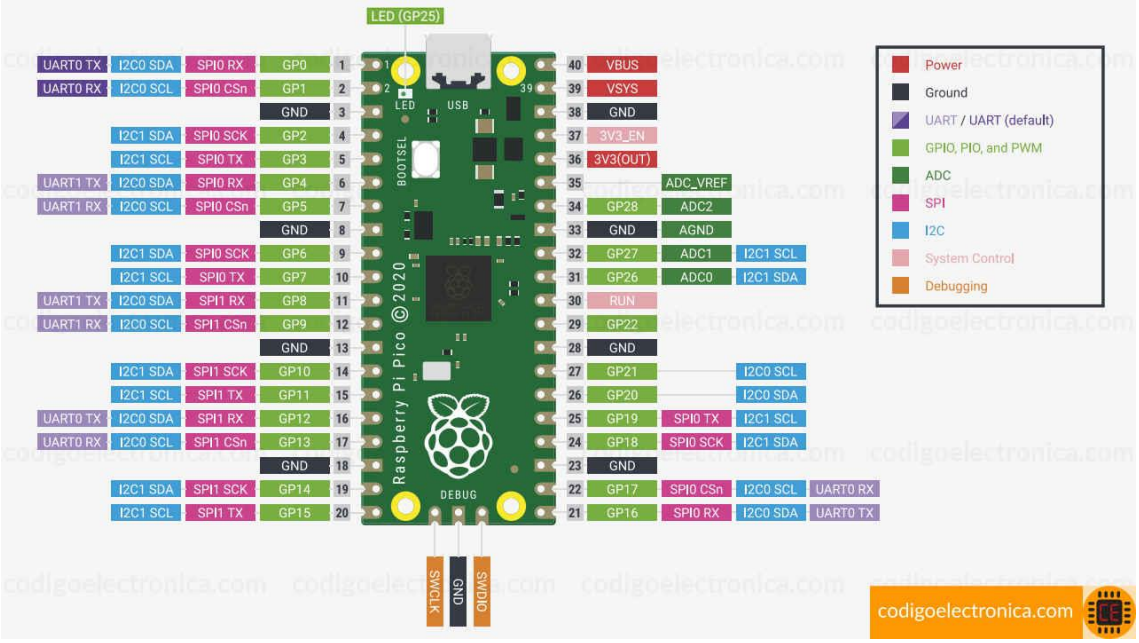


Figura 28: Arquitectura Raspberry Pi
(Raspberry, 2018)

2.5.5 Jetson Nano

La Jetson NANO de NVIDIA es una plataforma de desarrollo para implementar sistemas de inteligencia artificial cómodamente. Incluye todos los periféricos necesarios para desarrollar un sistema embebido que utilice visión artificial, redes neuronales y más. Cuenta con un sistema operativo Linux base con el SDK de NVIDIA para desarrollar sistemas de IA en poco tiempo permitiendo reducir el tiempo total de desarrollo, ya que las empresas pueden actualizar el rendimiento y las capacidades incluso después de que se haya implementado una aplicación (Álvarez Pastor, 2017; Libutti et al., 2020; NVIDIA, 2021). En la Figura 29 se presenta los componentes arquitectónicos de una tarjeta gráfica Nvidia Jetson Nano.

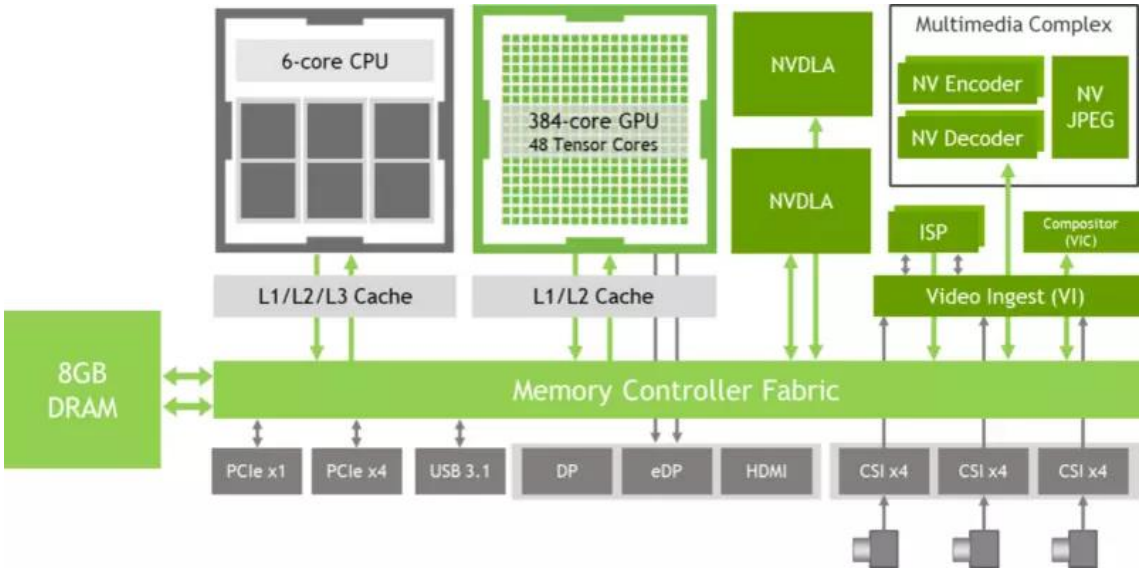


Figura 29: Arquitectura de Nvidia Jetson Nano

(NVIDIA, 2021)

2.6 Lenguajes de programación paralela

El proceso de paralización consiste en transformar un programa secuencial en una nueva versión concurrente semánticamente equivalente. El proceso puede varias de diferentes formas, una de ellas es la siguiente:

- ✓ De manera automática por el compilador, intentando obtener tareas u operaciones que puedan realizarse de forma paralela.
- ✓ Manualmente programando el algoritmo en un lenguaje de programación paralelo que permita especificar operaciones o tareas de mara concurrente.

En un programa secuencial el paralelismo puede desarrollarse en 3 niveles:

- ✓ Nivel de procedimientos: en este caso se ejecutan varias llamadas a procedimientos de forma simultánea.
- ✓ Nivel de bucles: en este caso se ejecutan varias iteraciones y/o sentencias del bucle en paralelo.
- ✓ Nivel de bloque básico: en este caso se ejecutan en paralelo varias operaciones de un bloque de sentencias de asignación.

En relación con los bucles en forma paralela se pueden identificar dos tipos que debido a características diferentes las técnicas de paralelización son distintas:

- ✓ Bucles fijos: el rango de iteración (valor inicia, valor final e incremento de la variable de control del bucle) es conocido en tiempo de compilación.
- ✓ Bucles no fijos: el rango de iteración se determina durante la ejecución del bucle y por lo tanto el tiempo de compilación es desconocido.

En general, la aceptación de la programación paralela ha sido facilitado por dos aspectos principales:

- ✓ Procesadores masivamente paralelos.
- ✓ Uso generalizado de la computación distribuida.

2.6.1 MPI

MPI es una librería estándar que se usa en programas que aprovechan arquitecturas de múltiples procesadores. MPI se desarrolló en 1993 como un estándar abierto por un gran número de organizaciones, principalmente desarrollada por el Centro de investigación en Computación paralela de Williamsbrug, el cual contenía una librería de paso por mensajes diseñada para sistemas de computadores intercomunicados con conexiones en paralelo denominada PVM V3 (Parallel Virtual machine). Enseguida, en mayo 1994 se ajustaron algunos complementos de la librería de PVM y nace la primera versión de MPI 1.0 (Message Passing Interface). Una característica importante de MPI es que permite trabajar con grupos de procesadores definidos según el programador lo disponga mediante objetos virtuales denominados comunicadores, que permiten distribuir los recursos según la tarea a realizar (Alonso & Lvarruiz, 2018; Czech, 2017; Yang et al., 2011).

Con la librería MPI, sólo se puede declarar una única vez el ambiente en paralelo (sección comprendida entre las funciones MPI_Init y MPI_Finalize) y que todo el código que este dentro de la zona se ejecutará en simultáneo por todos los procesos. Sus lenguajes de especificación son C, C++ y Fortran. Existen implementaciones para Python, Ocalm, Java, .NET y PHP. Cualquier librería, MPI cuenta con una serie de funciones que permiten llevar a cabo tareas

específicas dentro del ambiente en paralelo (Hager & Wellein, 2010; Rahmat et al., 2019). En la Figura 30 se presenta la estructura de un programa en MPI utilizando paralelismo.

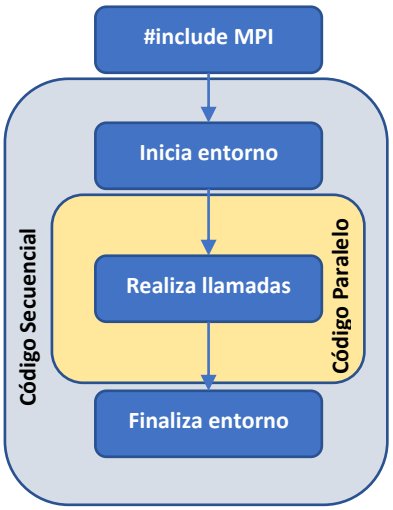


Figura 30: Estructura de un programa en MPI
(Herbert, 2015)

Dentro de los grupos de funciones para MPI se destacan algunas como:

- ✓ **Funciones de control de flujo***:* permiten crear y establecer parámetros de la sección en paralelo como número de procesos a usar, el comunicador, los ID de los procesos de la aplicación, etc.
- ✓ **Funciones para el manejo de grupos y comunicadores:** facilitan la conformación de los grupos de procesadores.
- ✓ **Funciones de administración de recurso:** gestiona los recursos disponibles del procesador y del sistema operativo.

- ✓ **Funciones de comunicación:** permiten la interacción (enviar y recibir información) entre diferentes procesos. Según el número de procesos presentes en la comunicación ésta se clasifica en punto a punto y multipunto.

- ✓ **Funciones para comunicación punto a punto:** implican la interacción de dos procesos exclusivamente (maestro y esclavo), que según el tipo de petición para establecer la conexión se dividen en método bloqueante y no bloqueante.

- ✓ **Funciones para comunicación multipunto***:* interactúan con múltiples procesos simultáneamente, el uso de ellas requiere que el desarrollador tenga claro el recurso con el que cuenta.

2.6.2 OpenMP

OpenMP (Open Multi-Processing) es una API para programación multiproceso de memoria compartida en múltiples plataformas, denominada como una interfaz portable para computadores o redes que soportan SMM, adoptada como un estándar informal en 1997 por científicos que buscaban generalizar los programas basados en SMM (Hager & Wellein, 2010; OpenMP, 2022). Su modelo de ejecución se basa en memoria compartida multihilos. Se considera el sucesor más sofisticado de Posix Threads (Portable operating system interface for Linux, es una librería para sistemas operativos). Dispone de directivas que apoyan al programador para convertir algoritmos secuenciales a paralelos de forma eficiente, introduciendo el código necesario para lanzar al mismo tiempo múltiples hilos. La gran portabilidad es una característica de OpenMP debido a que está soportado para C, C++ y Fortran, disponible para sistemas operativos como Solaris, AIX, HP-UX, GNU/Linux, MAC OS, y Windows (De Giusti et al., 2017; Fang et al., 2020; Grama et al., 2003). En la Figura 31 presenta la estructura de un programa básico en OpenMP utilizando paralelismo a través de directivas de compilador especiales, que indican al compilador cómo dividir y ejecutar el código en paralelo.

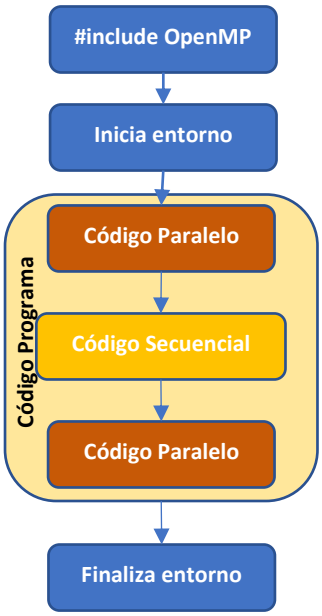


Figura 31: Lanzamiento de múltiples hijos en regiones paralelas (OpenMP, 2022)

Los compiladores y herramientas de desarrollo Intel C/C++ ofrecen muchas extensiones basadas en lenguajes que se pueden utilizar para simplificar el proceso de desarrollo de programas paralelos de alto rendimiento. OpenMP es un estándar bien conocido para la programación paralela de memoria compartida de alto nivel. OpenMP en una extensión de compilador C/C ++ y Fortran que le da al programador la capacidad de agregar paralelismo en un código fuente sin la necesidad de reescribir el código de manera significativa (Printista et al., 2011; S. Zhang et al., 2017). Con el fin de optimizar la utilización de la CPU, las directivas del compilador se utilizan para especificar la ejecución paralela de partes seleccionadas de programas informáticos (es decir, bucles y secciones de código). Las directivas también se utilizan para proporcionar construcciones de sincronización explícitas. Dichas directivas pueden contener cláusulas que definan el intercambio de datos entre subprocesos. Además, OpenMP consta de un pequeño conjunto de rutinas de biblioteca y variables de entorno que influyen en el comportamiento del tiempo de ejecución. Las versiones más recientes del estándar definen construcciones de programación más avanzadas, como tareas y compatibilidad con SIMD (OpenMP, 2022; Slabaugh et al., 2010).

Además, OpenMP soporta la interacción con el modelo de paso por mensajes (MPM), permitiendo la integración de la librería MPI en las aplicaciones, lo que

amplía las alternativas de programación (Hager & Wellein, 2010). En la Figura 32 se presenta un algoritmo básico de un programa desarrollado en OpenMP. Para utilizar OpenMP, se debe incluir una biblioteca de encabezados en el código fuente y agregar directivas de compilador específicas que indiquen cómo se deben paralelizar las secciones críticas del código. Estas directivas indican al compilador cómo dividir el trabajo entre los hilos y cómo manejar la sincronización y los datos compartidos.

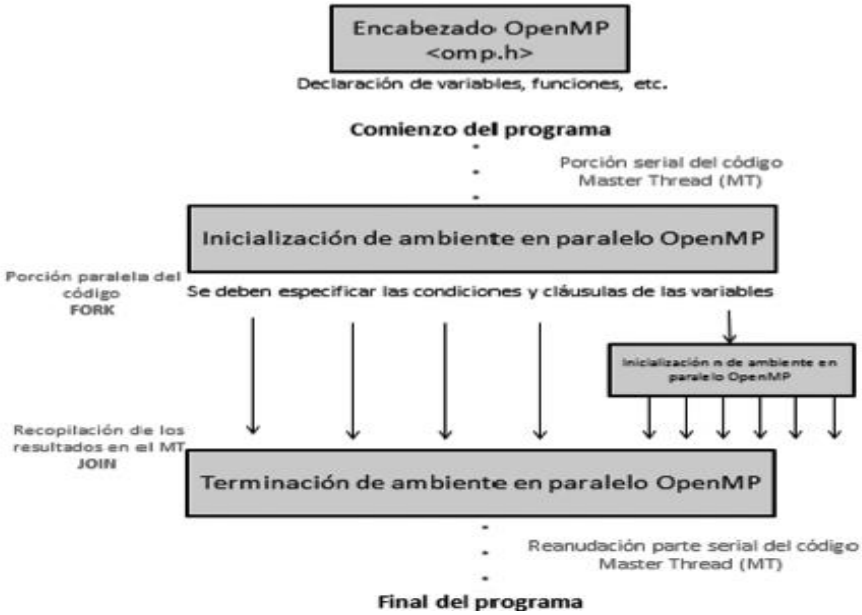


Figura 32: Estructura general de un programa en OpenMP
(OpenMP, 2022)

Una importancia de OpenMP es que permite administrar el recurso en rutinas que contienen cálculos iterativos, mediante la distribución de iteraciones de los ciclos en los diferentes threads (hilos) que maneja la aplicación mediante funciones especiales denominadas constructores. Los constructores se combinan con una cláusula especial llamada Schedule que contiene parámetros que definen la forma de distribución de las iteraciones. OpenMP, consiste en crear nuevas secciones paralelas dentro de los mismos threads formando una especie de ramificación que facilitan el procesamiento de grandes volúmenes de información (Alonso & Lvarruiz, 2018; Czech, 2017; OpenMP, 2022; Tabares Soto, 2016).

Al igual que MPI, OpenMP también cuenta con rutinas especiales para atención y detección de errores, que son de gran ayuda para comprender el modo de trabajo. Con respecto a la tecnología multinúcleo, se ha realizado trabajos para evaluar el rendimiento de diferentes lenguajes de programación paralelos, construcciones, herramientas y bibliotecas. Sin embargo, apenas se está conociendo el rendimiento de un código paralelo con diferentes tipos de programación OpenMP y tamaños de fragmentos variados en un procesador multinúcleo de memoria compartida. La falta de pautas sobre la asignación del tipo de programación y el tamaño del fragmento adecuados podría hacer que un usuario de OpenMP deje la configuración como predeterminada. Dado que el tipo de programación decide cómo se divide la carga de trabajo entre los subprocesos, mientras que el tamaño del fragmento define la granularidad del trabajo, es necesario determinar el tipo de programación junto con el tamaño de fragmento apropiado de un código paralelo. No obstante, aún no se ha identificado cuánta contribución dan estas directivas en términos de mejora del rendimiento de diferentes procesadores multinúcleo (Czech, 2017; OpenMP, 2022).

2.6.3 CUDA

Es una plataforma de desarrollo para las GPGPU o Computación de propósito general en unidades de procesamiento gráfico creada por la Corporación NVIDIA enfocada en la computación y programación paralela (Dávila-Guzmán et al., 2014; NVIDIA, 2015). En la Figura 33 se presenta el modelo de programación en CUDA, se basa en la creación de programas que se ejecutan tanto en la CPU como en la GPU. Para aprovechar el potencial de la GPU, CUDA divide el trabajo en pequeñas tareas que se ejecutan en paralelo en la GPU.

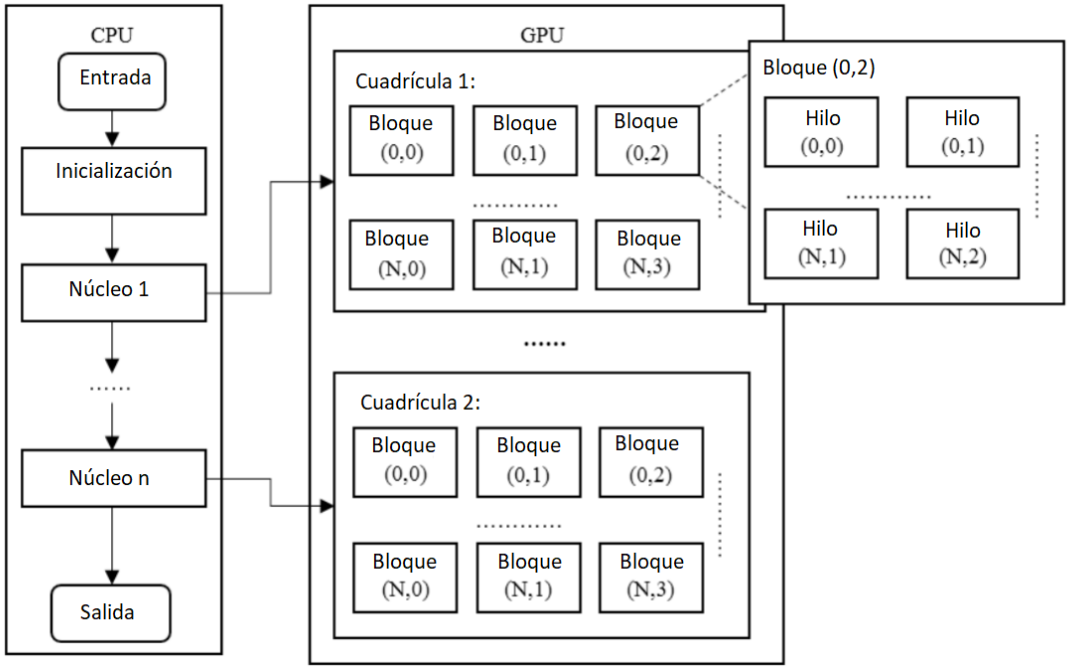


Figura 33:*Modelo de programación de CUDA*
(Tabares Soto, 2016).

CUDA Toolkit proporciona un entorno de desarrollo integral para desarrolladores de C y C++ que crean aplicaciones aceleradas por GPU de alto rendimiento con bibliotecas CUDA. El kit de herramientas incluye Nsight Eclipse Edition, herramientas de depuración y creación de perfiles, incluida Nsight Compute, y una cadena de herramientas para aplicaciones de compilación cruzada (NVIDIA, 2015; J. Wang et al., 2014).

- NVIDIA Nsight Systems es una herramienta de creación de perfiles de todo el sistema de baja sobrecarga que proporciona los conocimientos que los desarrolladores necesitan para analizar y optimizar el rendimiento del software.

- NVIDIA Nsight Graphics es una aplicación independiente para depurar y crear perfiles de aplicaciones gráficas.

- NVIDIA Nsight Deep Learning Designer es un entorno de desarrollo integrado que ayuda a los desarrolladores a diseñar y desarrollar de manera eficiente redes neuronales profundas para la inferencia en la aplicación.

- SDK y herramientas compatibles

- NVIDIA DeepStream SDK es un conjunto completo de herramientas de análisis para el procesamiento de múltiples sensores basado en IA y la comprensión de audio y video.

2.6.4 Python

Es uno de los lenguajes más utilizados en diversas áreas de aplicación. Sin embargo, tiene limitaciones cuando se trata de optimizar y paralelizar aplicaciones debido a la naturaleza de su intérprete oficial CPython (Milla & Rucci, 2022); para ello requiere un intérprete denominado GIL que permite que sólo un hilo tome el control del intérprete, es decir, que solo un hilo puede estar en ejecución a la vez (Milla & Rucci, 2022). En la Figura 34 se presenta el modelo de programación mediante paralelismo en el que se basa Python. El intérprete limita el acceso concurrente a objetos de memoria compartida.

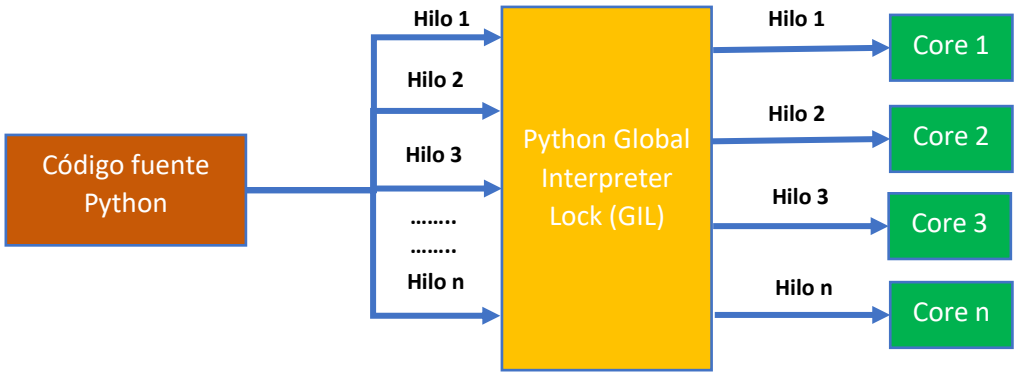


Figura 34: *Modelo de programación paralela de Python.*
(Python, 2022)

2.6.5 Matlab

Matlab incluye la herramienta “Parallel Computing ToolBox”, permite resolver problemas con un uso intensivo de cálculos y datos mediante procesadores multinúcleo, GPU y clústeres de computadores. Las construcciones de alto nivel como for-loops paralelos, tipos especiales de arrays y algoritmos numéricos paralelizados permiten paralelizar las aplicaciones sin necesidad de programación MPI o CUDA (Rodríguez, 2010; Suh & Kim, 2014). La ToolBox permite usar funciones compatibles con el cálculo paralelo en MATLAB para ejecutar varias simulaciones de un modelo en paralelo. Los programas y los

modelos se pueden ejecutar en modo interactivo y por lotes (Suh & Kim, 2014). En la Figura 35 se presenta la arquitectura de programación paralela en la que se basa Matlab tanto para procesadores multicore y para unidades de procesamiento gráfico (GPU).

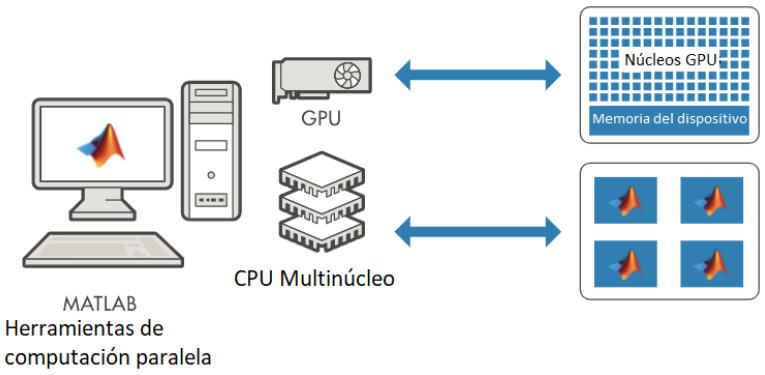


Figura 35:Arquitectura de programación paralela Matlab
(Suh & Kim, 2014).

Un modelo de programación heterogéneo en Matlab genera porciones altamente paralelas de la aplicación de software para asignar a núcleos que se ejecutan en el dispositivo GPU físicamente separado, mientras que el resto del código secuencial se ejecuta en la CPU. A cada núcleo se le asignan varios trabajos o subprocesos, que se organizan en bloques y cuadrículas. Cada subproceso dentro del núcleo se ejecuta simultáneamente con respecto a cada uno de los demás (MathWorks, 2022). En la Figura 36 se presenta la arquitectura de funcionamiento para programación paralela en arquitecturas heterogéneas utilizando Matlab. MATLAB puede trabajar con arquitecturas heterogéneas, incluyendo procesadores multicore, clústeres de computadoras, GPU, FPGA, dispositivos móviles y dispositivos embebidos permitiendo acelerar la ejecución de cálculos y utilizar de manera eficiente los recursos de hardware disponible.

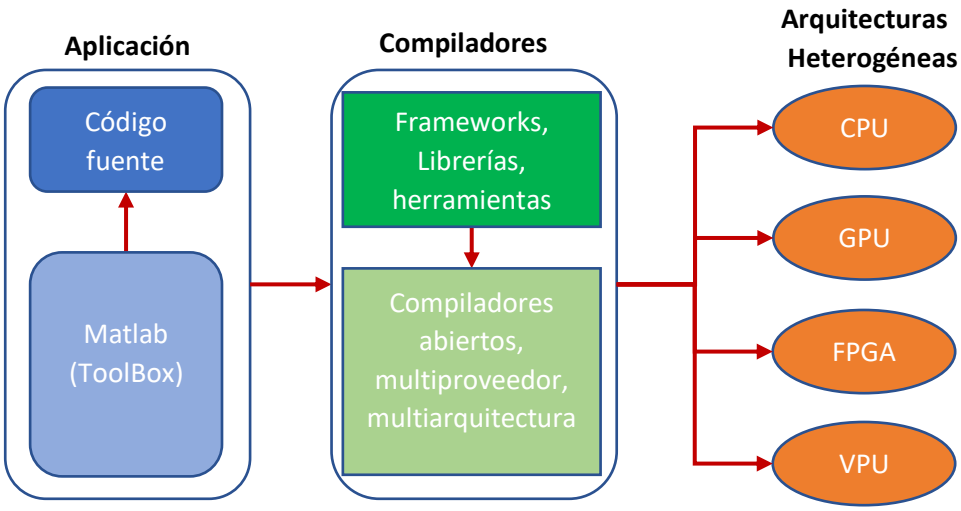


Figura 36: Programación en Arquitecturas Heterogéneas con Matlab. (Hager & Wellein, 2010)

2.7 Diseño de Algoritmos paralelos

El diseño de algoritmos en paralelo consiste en descomponer el problema principal en diferentes subproblemas para ejecutar de manera simultánea en bloques, considerando que los bloques sean los más independientes entre sí y que no dependan de los mismos recursos computacionales como núcleos del procesador, posiciones de memoria y zonas físicas de los archivos.

2.7.1 Teoría de la complejidad computacional

La teoría de la complejidad computacional o teoría de la complejidad informática es una rama de la teoría de la computación que se centra en la clasificación de los problemas computacionales de acuerdo con su dificultad inherente, y en la relación entre dichas clases de complejidad (WikiWand, 2019).

Un problema se cataloga como "inherentemente difícil" si la solución requiere de una cantidad significativa de recursos computacionales, sin importar el algoritmo utilizado. La teoría de la complejidad computacional formaliza dicha aseveración, introduciendo modelos de computación matemáticos para el estudio de estos problemas y la cuantificación de la cantidad de recursos necesarios para resolverlos, como tiempo y memoria (Dean, 2010).

Una de las metas de la teoría de la complejidad computacional es determinar los límites prácticos de qué es lo que se puede hacer en una computadora y qué no. Otros campos relacionados con la teoría de la complejidad computacional son el análisis de algoritmos y la teoría de la computabilidad. Una diferencia significativa entre el análisis de algoritmos y la teoría de la complejidad computacional, es que el primero se dedica a determinar la cantidad de recursos requeridos por un algoritmo en particular para resolver un problema, mientras que la segunda, analiza todos los posibles algoritmos que pudieran ser usados para resolver el mismo problema (Stanford Encyclopedia of Philosophy, 2017).

La teoría de la complejidad computacional trata de clasificar los problemas que pueden ser resueltos con una cantidad determinada de recursos. A su vez, la imposición de restricciones sobre estos recursos, es lo que la distingue de la teoría de la computabilidad, la cual se preocupa por qué tipo de problemas pueden ser resueltos de manera algorítmica (Arora & Boaz, 2009; Dean, 2010).

Un algoritmo resuelve un problema A, si dicho algoritmo se puede aplicar a cualquier instancia I de A, y se garantiza que siempre produce una solución para dicha instancia. De manera general, interesa encontrar el algoritmo más "eficiente" para resolver cierto problema involucrando a todos los recursos computacionales necesarios para la ejecución del algoritmo (Stanford Encyclopedia of Philosophy, 2017).

Por algoritmo "más eficiente" usualmente nos referimos al más rápido. Debido a que los requerimientos de tiempo son usualmente un factor dominante cuando se trata de determinar si un algoritmo es lo suficientemente eficiente para ser útil en la práctica, nos concentraremos en este recurso (Dean, 2010). En la Figura 37 se presenta las áreas y subáreas de la teoría de la complejidad computacional.

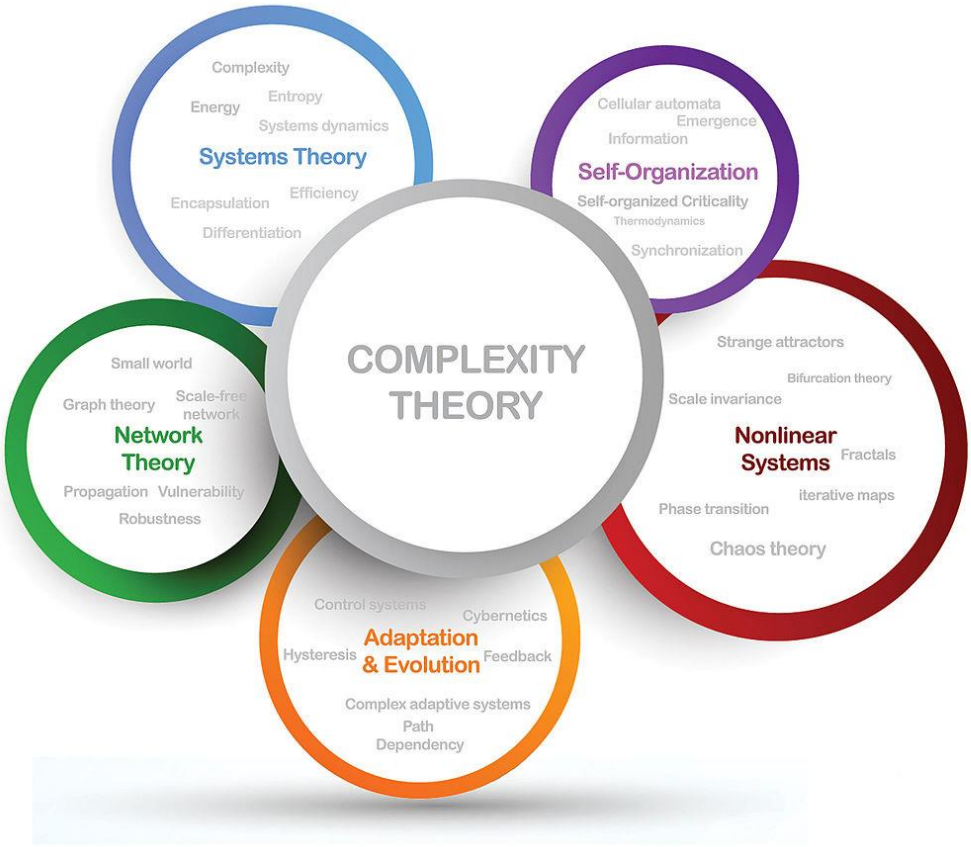


Figura 37: Teoría de la complejidad

(Dean, 2010)

2.7.2 Principios de programación paralela

Para el desarrollo de aplicaciones con paralelismo, se buscan dos objetivos básicos:

- ✓ Mejorar el rendimiento de la aplicación, maximizando la concurrencia y reduciendo los costes adicionales de esta paralelización (con lo que se minimiza el speedup obtenido).
- ✓ Productividad en el momento de programar: legibilidad, portabilidad, independencia de la arquitectura destino.

El objetivo principal es distribuir las tareas/datos aprovechando de mejor manera la concurrencia, luego seleccionar el esquema de aplicación paralela más adecuada y finalmente, adaptar el algoritmo a las estructuras de implementación que se adapten a las unidades de procesamiento (Weiss, 2014), (Yang et al., 2011).

2.7.3 Concurrencia en los algoritmos

El problema general se puede distribuir con las siguientes posibilidades (Hager & Wellein, 2010; Weiss, 2014):

- ✓ Identificar tareas (partes del código que pueden ser ejecutadas concurrentemente
- ✓ Distribuir y analizar las estructuras de datos para saber cuáles son los datos que necesitan tareas, los datos que producen las tareas, y qué datos se comparten. Con este análisis podemos intentar minimizar los movimientos de datos entre tareas, o los datos compartidos.
- ✓ Determinar las dependencias entre tareas para determinar el orden entre ellas y las sincronizaciones necesarias, realizando el grafo de dependencia de tareas y que el resultado final de la paralelización sea el mismo que el del código secuencial.

Para encontrar la concurrencia/paralelismo en un algoritmo, primero debemos determinar si son los datos, las tareas, o el flujo de datos los que determinan este paralelismo. Las posibilidades de distribución son las siguientes:

Descomposición de datos: las estructuras de datos se particionan de tal forma que se asigna una tarea a cada partición realizada.

Descomposición de tareas: identificar las partes del código como tareas, y esto implicará una distribución de los datos.

Descomposición de flujo de datos: el flujo de los datos conllevará que unas tareas se activen y realicen el proceso de estos datos.

2.7.4 Metodología de Foster's

Los algoritmos paralelos son importantes para solucionar problemas computacionales grandes para diversos campos de aplicación. La metodología de Foster enfatiza el desarrollo de software en tres áreas principales: paralelismo, comunicación y tolerancia a fallas. El paralelismo implica dividir las tareas en partes más pequeñas que se pueden ejecutar simultáneamente en diferentes procesadores. La comunicación se refiere al intercambio de datos entre diferentes partes del sistema distribuido. La tolerancia a fallas implica la capacidad del sistema para continuar funcionando incluso si una o más partes fallan.

En la Figura 38 se presenta el esquema básico de algoritmo paralelo según Foster's enfatizando el desarrollo de sistemas distribuidos y altamente paralelos, capaces de comunicarse de manera efectiva y tolerantes a fallas. Este enfoque es especialmente útil para los sistemas informáticos de alto rendimiento que requieren un procesamiento intensivo y manejan grandes cantidades de datos.

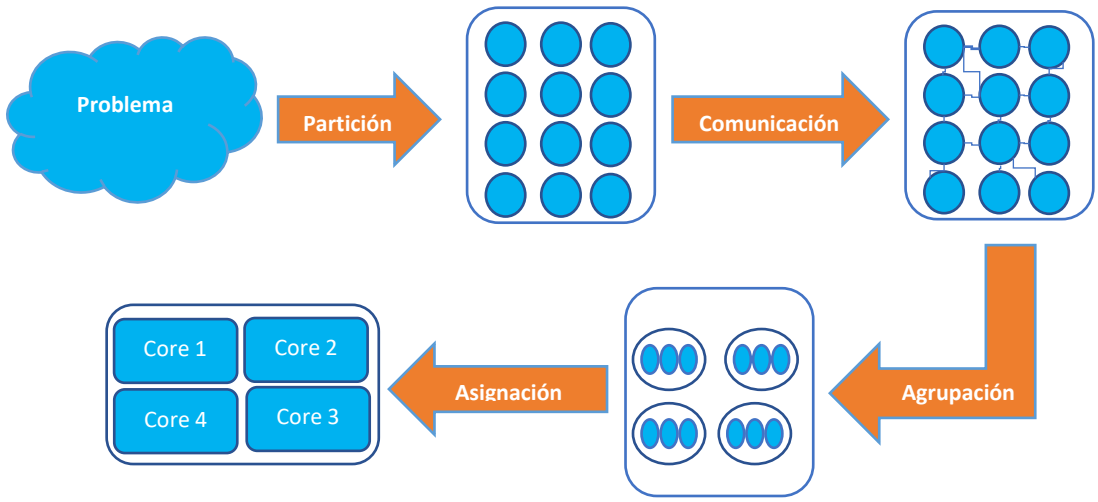


Figura 38: Metodología de diseño según Foster.
(Weiss, 2014)

Partición: el computo o los datos se descomponen en pequeñas tareas. Usualmente es independiente de la arquitectura o del modelo de programación. Un buen particionamiento divide tanto los cálculos asociados con el problema como los datos sobre los cuales opera. Con la partición se buscan oportunidades de paralelismo y se trata de subdividir el problema lo más finamente posible, es decir; que la granularidad sea fina. Un buen particionamiento divide tanto los cómputos como los datos (Díaz, 2018; Weiss, 2014).

- ✓ Descomposición funcional
- ✓ Descomposición de dominio

Al particionar se deben tener en cuenta los siguientes aspectos:

- ✓ El número de tareas debe ser por lo menos un orden de magnitud superior al número de procesadores disponibles para tener flexibilidad en las etapas siguientes.
- ✓ Evitar cómputos y almacenamientos redundantes; de lo contrario el algoritmo puede ser no extensible a problemas más grandes.
- ✓ Tratar de que las tareas sean de tamaños equivalentes ya que facilita el balanceo de la carga de los procesadores.
- ✓ Considerar alternativas de paralelismo en esta etapa ya que pueden flexibilizar etapas subsecuentes
- ✓ El número de tareas debe ser proporcional al tamaño del problema. Así, se podrá resolver problemas más grandes cuando se tenga más procesadores. Es decir, que el algoritmo sea escalable.

Comunicación: las tareas generadas por una partición están propuestas para ejecutarse concurrentemente pero no pueden, en general, ejecutarse independientemente. Los cálculos en la ejecución de una tarea normalmente requerirán de datos asociados con otras tareas. Los datos deben transferirse entre las tareas y así permitir que los cálculos procedan. Este flujo de información se especifica en esta fase. La comunicación requerida por un algoritmo puede ser definida en dos fases (Díaz, 2018; Weiss, 2014).

- ✓ Primero se definen los canales que conectan las tareas que requieren datos con las que los poseen.

- ✓ Segundo se especifica la información o mensajes que deben ser enviado y recibidos en estos canales

En ambientes de memoria distribuida las tareas interactúan enviando y recibiendo mensajes. Pero en ambientes de memoria compartida se debe utilizar mecanismos de sincronización para controlar el acceso a la memoria como:

- ✓ Semáforos
- ✓ Semáforos binarios
- ✓ Barreras.

En la etapa de comunicación hay que tener en cuenta los siguientes aspectos:

- ✓ Todas las tareas deben efectuar aproximadamente el mismo número de operaciones de comunicación. Si esto no se da, es muy probable que el algoritmo no sea extensible a problemas mayores ya que habrá cuellos de botella.
- ✓ La comunicación entre tareas debe ser tan pequeña como sea posible.
- ✓ Las operaciones de comunicación deben poder proceder concurrentemente.
- ✓ Los cómputos de diferentes tareas deben poder proceder concurrentemente

Agrupación: las tareas y las estructuras de comunicación definidas en las dos primeras etapas del diseño son evaluadas con respecto a los requerimientos de ejecución y costos de implementación. Si es necesario, las tareas son combinadas en tareas más grandes para mejorar la ejecución o para reducir los costos de comunicación y sincronización (Díaz, 2018; Weiss, 2014).

Mediante la agrupación de tareas se puede reducir la cantidad de datos a enviar y así reducir el número de mensajes a transmitir y por ende el costo de comunicación. En la etapa de agrupación se debe tener en cuenta los siguientes aspectos:

- ✓ Chequear si la agrupación redujo los costos de comunicación.
- ✓ Si se han replicado cómputos y/o datos, se debe verificar que los beneficios sean superiores a los costos.
- ✓ Se debe verificar que las tareas resultantes tengan costos de cómputo y comunicación similares.

- ✓ Hay que revisar si el número de tareas es extensible con el tamaño del problema.
- ✓ Si el agrupamiento ha reducido las oportunidades de ejecución concurrente, se debe verificar que exista suficiente concurrencia y posiblemente considerar diseños alternativos.
- ✓ Analizar si es posible reducir al máximo el número de tareas sin introducir desbalances de cargas o reducir la extensibilidad.

Asignación: cada tarea es asignada a un procesador de tal modo que intente satisfacer las metas de competencia al maximizar la utilización del procesador y minimizar los costos de comunicación. La asignación puede ser estática (se establece antes de la ejecución del programa) o en tiempo de ejecución mediante algoritmos de balanceo de carga. En la etapa de asignación se determina en que procesador se ejecutará cada tarea (Díaz, 2018; Weiss, 2014).

La asignación de tareas puede ser realizada de las siguientes maneras:

- ✓ Una tarea es asignada a un procesador desde su inicio hasta su fin denominándose estática.
- ✓ Una tarea puede ser migrada durante su ejecución denominándose dinámica. Esto puede agregar un costo adicional.

Capítulo 3. Caso de estudio

En el presente trabajo se enfocó en la investigación y desarrollo de algoritmos de visión por computador utilizando técnicas de programación paralelo utilizando imágenes adquiridas con drones a diferentes alturas, con la finalidad de determinar la altura mínima/máxima de reconocimiento automático de líneas de cultivo/malezas en campos de maíz para comparar el rendimiento de los algoritmos secuenciales con los nuevos algoritmos paralelos. Las imágenes con mayor altura proporcionan mayor superficie de terreno posibilitando obtener datos de grandes extensiones de cultivo en poco tiempo considerando que el tiempo de vuelo del dron (Mavic 2 Pro) es de aproximadamente 30 minutos.

Para el caso de estudio se realizó algunas fotografías con el dron a diferentes alturas considerando la visibilidad del ojo humano. La altura mínima fue de 3 metros y la máxima de 20 metros. Para capturar fotografías a 3 metros en tercera y cuarta semana de seguimiento debido al tamaño de las plantas de cultivo las hélices del dron movían las plantas dificultando obtener fotografías de buena calidad. En cambio, con las fotografías capturadas a 20 metros fue difícil diferenciar las plantas de cultivo del resto de vegetación considerado las similitudes de las plantas de cultivo con las malezas. Con estos antecedentes se consideró trabajar con imágenes digitales con una resolución de 5472×3648 píxeles desde tres alturas (5, 10 y 15) metros de tal manera que sea posible procesar una imagen con mayor superficie de terreno reduciendo tiempos de ejecución, pero obteniendo líneas de cultivo detectadas y porcentaje de maleza lo más cercano a los datos reales presentes en las imágenes del cultivo.

3.1 Herramientas y Equipamiento

3.1.1 Dron DJI Mavic 2 Pro

El modelo de dron utilizado para este estudio fue el DJI Mavic 2 Pro, equipado con una cámara de alta calidad para capturar imágenes originales RGB en formato JPG. El software para controlar el vuelo del dron fue DJI GO 4 a una velocidad promedio de 15 metros/segundo. Las condiciones para la captura de datos fueron realizadas en días soleados y sombreados en horas de la mañana, debido a la presencia de luz solar directa algunas imágenes se dañaron por lo

que se realizaron nuevas tomas hasta obtener imágenes con buena resolución. Las especificaciones técnicas del dron se detallan en la Tabla 1.

Tabla 1: Especificaciones técnicas del dron DJI Mavic 2 Pro (DJI, 2020)

Característica	**Detalle**
Peso	907 g (con batería)
Velocidad máxima	72 km / h (modo sport)
Velocidad ascenso/descenso	18/10.8 Km/h
Distancia máxima de vuelo (sin viento)	8 km (a una velocidad constante de 50 km/h)
Altitud máxima de vuelo	6 km
Cámara - Sensor	CMOS 1 pulgada de 20 MP
Cámara - Lente	Lente de 28 mm (angular 77°)
Foto	20 MP (5472×3648) JPEG – RAW – Panorámicas – Ráfagas 3/5 fotogramas
Gimbal	Motorizado 3 ejes
Batería	31 minutos de duración
Control del dron	App (Android – IOS)
Posicionamiento	GPS + GLONASS
Sensores	Omnidireccional – frontal, laterales, trasero, superior e inferior

3.1.2 Adquisición de imágenes

Las fotografías se realizaron en dos cultivos de maíz en diferentes periodos de tiempo y en dos terrenos de 300 metros cuadrados aproximadamente durante las primeras 4 semanas de crecimiento. Es importante la supervisión del crecimiento de los cultivos desde la siembra para visualizar el comportamiento de las plantas en relación con el tamaño del resto de vegetación y tomar decisiones tempranas con respecto a las malezas evitando problemas para los agricultores.

3.1.3 Detalles de las muestras de cultivo

El terreno usado para la toma de muestras está ubicado en Natabuela, parroquia de Antonio Ante, provincia de Imbabura, granja experimental “La Pradera” perteneciente a la Universidad Técnica del Norte.

El seguimiento y recolección de imágenes se consideró a partir de la segunda semana después de la siembra. Luego de la segunda semana las plantas de cultivos poseen un tamaño representativo para ser visualizado desde el aire. Durante 4 semanas se realizó el seguimiento al cultivo, este periodo es importante porque es la etapa temprana de desarrollo del maíz que se caracteriza por una alta tasa de crecimiento de las plantas de maíz. Las características de las imágenes utilizadas para el procesamiento, reconocimiento de líneas de cultivo y malezas se detallan en la Tabla 2. En la Tabla 3 se muestra los datos del cultivo de maíz durante las 4 semanas de seguimiento.

Tabla 2: Detalles de las imágenes capturadas desde el dron

Característica	**Medida (metros)**
Longitud aprox. de cada línea de cultivo:	10
Distancia aprox. entre líneas de cultivo:	0.8
Distancia aprox. entre plantas dentro de la línea de cultivo:	0.4

Tabla 3: Información del cultivo durante las 4 primeras semanas

Características	**Primera semana**	**Segunda semana**	**Tercera semana**	**Cuarta semana**
Ancho de la planta	9 – 12 cm	14 - 15 cm	18 cm	20 - 21 cm
Numero de hojas	3	4	7	8
Altura de la planta	8 cm	13 cm	29.5 cm	35.5 cm
Longitud de las hojas	5.2 cm	5.7 cm	11.43 cm	12.89 cm

3.1.4 Equipo de procesamiento

Las pruebas del algoritmo secuencial y paralelo se realizaron en una Laptop HP Pavilion 14 Notebook PC con Windows 11 Pro de 64 bits, procesador Intel Core I7, 16 MB de memoria RAM. Las características técnicas del computador, sistema operativo y procesador utilizado para el procesamiento de los algoritmos se presentan en la Tabla 4.

Tabla 4: Especificaciones técnicas del equipo de procesamiento

Características del equipo	Características del procesador
Sistema operativo: Microsoft Windows 11 Pro	Arquitectura: X86_64
Memoria RAM: 16	CPU: 32 bit, 64 bit
Tipo de sistema: Sistema operativo de 64 bits, procesador basado en x64	Byte order: Little Endian
Intel(R) Core (TM) i7-10750H CPU @ 2.60GHz 2.59 GHz	CPU(s): 8
Edición: Windows 11 Pro	On-line CPU(s) list: 0-3
Versión: 21H2	Threads(s) per core: 2
Versión de sistema operativo: 22000.978	Core(s) per socket: 2
Fabricante del sistema operativo: Microsoft Corporation	Sockets(s): 1
Nombre del sistema: MARCK-PC	Vendor ID: GenuineIntel
Fabricante del sistema: Hewlett-Packard	Model name: Intel ® Core™ i7-5500U CPU @ 2.40 Ghz
Modelo del sistema: HP Pavilion 14 Notebook PC	CPU MHz: 2401.000
Tipo de sistema: PC basado en 64 bits	CPU max MHz: 2401.0000

3.2 Diseño del algoritmo

La paralelización en la CPU se caracteriza por el reparto de la carga de cálculo entre los distintos núcleos (físicos) que incluye el procesador (Azzini et al., 2018; Pascual, 2017; Suh & Kim, 2014). Para la implementación en Matlab de forma paralela se utilizaron sentencias permitidas por la herramienta de programación (Parallel Computing ToolBox) para el procesamiento de imágenes digitales con el objetivo de detectar líneas de cultivo y discriminar las malezas. En Matlab se utiliza el procesamiento paralelo de datos mediante la sentencia **parfor,** la mencionada sentencia trabaja en reemplazo de las sentencias de repetición (for) distribuyendo el trabajo entre los núcleos físicos disponibles y en cada núcleo se distribuye las tareas utilizando los hilos disponibles. La sentencia parfor se utilizó en los ciclos donde todas sus interacciones son distribuidas y ejecutadas entre todos los núcleos físicos (workers) disponibles dentro de un espacio de trabajo (MatLabpool) (MathWorks, 2020). En la Figura 39 se presenta el funcionamiento de la sentencia parfor, la cual mediante un hilo principal distribuye el trabajo entre un conjunto de hilos (pool). Los hilos en cada procesador pueden tener zonas

compartidas o privadas de memoria para culminar todas las tareas asignadas. Luego del trabajo de los hilos en cada procesador se consolida el resultado en una zona común de la memoria principal.

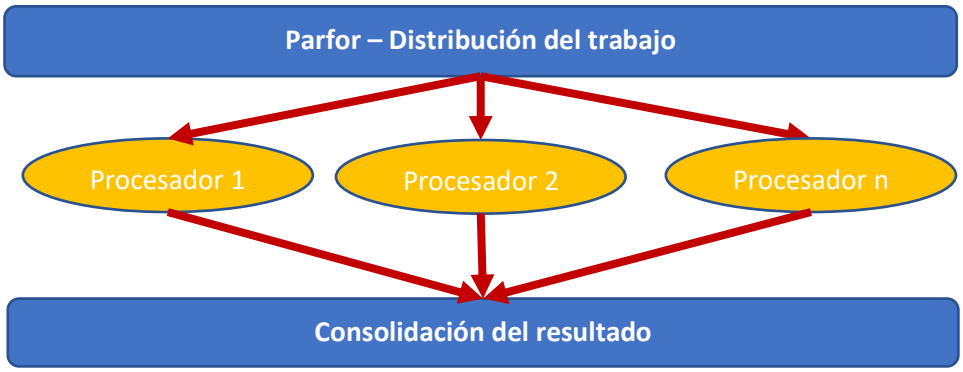


Figura 39: Distribución de trabajo de la sentencia parfor

3.2.1 Alturas y longitudes en píxeles

Para determinar algunas características como la longitud y ancho de las plantas con relación a los píxeles, se capturó imágenes a diferentes metros de altura (5, 10, 15) de una lámina de cartón de 50x50 centímetros (cm) como referencia para determinar el número de píxeles por metro cuadrado representado en el cultivo. En la Tabla 5 se presentan los valores de las imágenes obtenidas a 5, 10 y 15 metros de altura, tanto en píxeles (lámina 50x50 cm) como área de cobertura (metros cuadrados - m^2) de las imágenes adquiridas por el dron. El área de cobertura se obtiene mediante el número de líneas de cultivo reales existentes por la distancia de cada línea de cultivo y la longitud de cada línea de cultivo (Tabla 2).

Tabla 5. Alturas, longitudes en píxeles (px), área cobertura de las imágenes.

Altura (metros)	Largo (px)	Ancho (px)	Área (px)	Líneas de cultivo reales	Área cobertura(m^2)
15	140	140	19.600	7	
10	208	208	43.264	14	
5	432	432	89.856	20	

En la Figura 40 se presentan las imágenes a 5 m (40a), 10 m (40b) y 15 m (40c) enfocando a la lámina de 50x50cm, para determinar el número aproximado de píxeles que cada imagen representa dependiendo de la altura utilizada por el dron.

a) lámina a 5 metros b) lámina a 10 metros c) lámina a 15 metros

Figura 40: Lámina para determinar la cantidad de píxeles en las imágenes

3.2.2 Algoritmo paralelo con imágenes obtenidas con cámara

El algoritmo secuencial propuesto por (García-Santillán, Montalvo, et al., 2017), permite la detección automática de líneas curvas y rectas de cultivo, basado en pequeñas regiones de interés (micro-ROI) considerando tres fases: (i) segmentación, (ii) identificación de puntos de inicio y (iii) detección de líneas de cultivo. La fase detección de líneas de cultivo rectas y curvas siguen los siguientes pasos: (i) Extracción de puntos candidatos; (ii) análisis de regresión; y (iii) selección y verificación. Estas secciones se paralelizaron porque incluyen diferentes operaciones basadas en micro-ROI para considerar el máximo número de plantas de cultivo, dejando a la mayoría de vegetación fuera de la micro-ROI en el espaciado entre las líneas de cultivo. En la Figura 41 se presentan las secciones paralelizadas del algoritmo secuencial, en cada sección paralelizada se utiliza el mismo tipo de paralelismo como se indica previamente en la Figura 39.

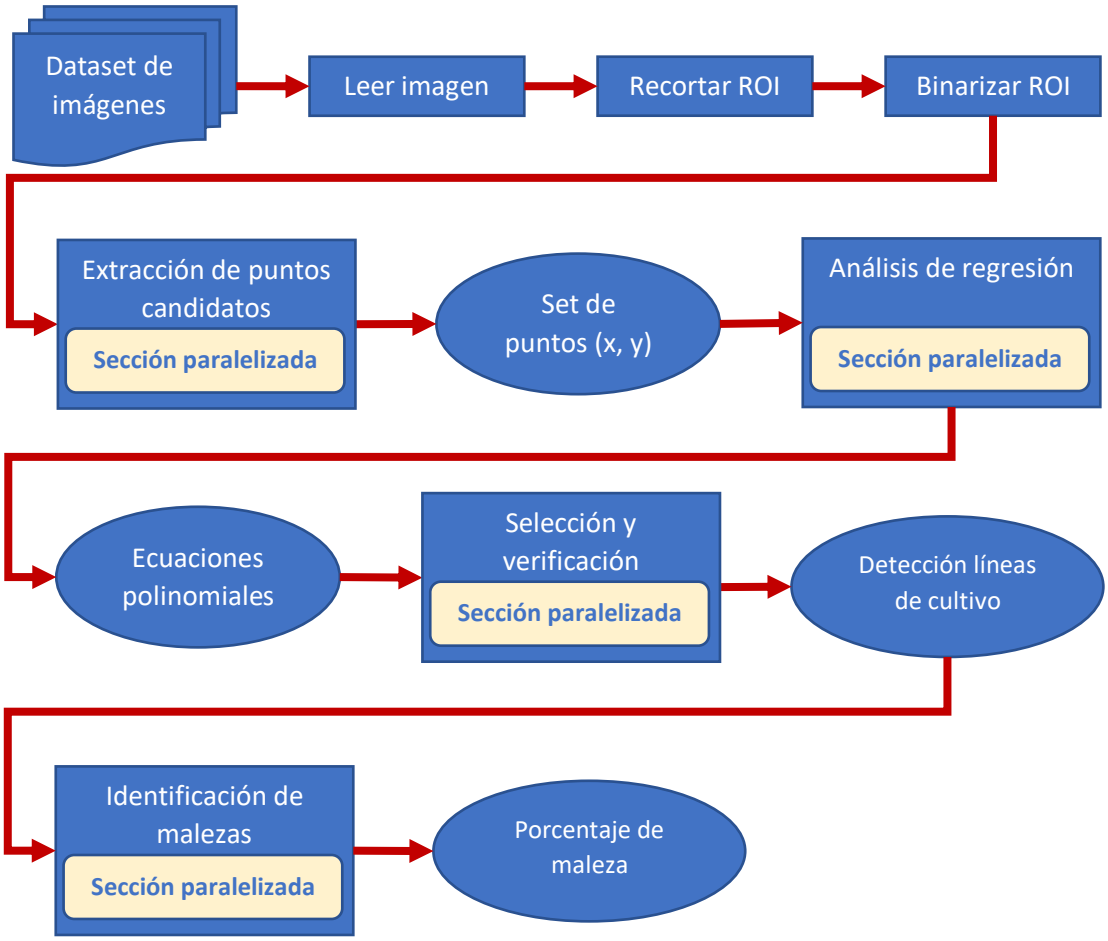


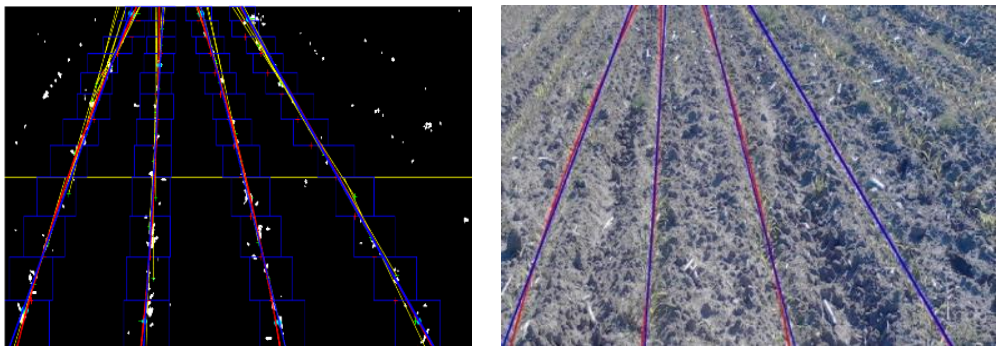
Figura 41: Secciones paralelizadas del algoritmo basado en micro-ROI.

Extracción de puntos candidatos: en esta sección se paralelizó el código para la definición de los micro-ROI por cada línea de cultivo, el micro-ROI inicial tiene un ancho de 150 píxeles, el resto de micro-ROI se reducen gradualmente el 5% de ancho tomando como base el anterior micro-ROI generado hasta llegar a un total de 12 para formar la curva cada línea de cultivo. Las líneas de cultivo se extraen de manera secuencial de izquierda a derecha numeradas de 1 a 4 considerando las alineaciones de cada micro-ROI desde un punto inicial en relación con la pendiente de línea recta estimada utilizada como guía.

Análisis de regresión: en esta fase se utilizó paralelismo para dibujar los micro-ROI tomando las coordenadas como referencia para identificar las plantas de cultivo, adicionalmente se paralelizó las generaciones de las rectas de las pendientes para la detección de la trayectoria de las plantas de cultivo.

Selección y verificación: en esta fase se utilizó paralelismo para la selección de líneas de cultivo y posteriormente la verificación de las líneas seleccionadas identificando las curvas que mejor se ajustan a cada línea de cultivo (Figura 42b). En la verificación de líneas de cultivo se considera el espacio de separación entre las líneas de cultivo y la orientación de las mismas para aceptar o rechazar la imagen.

El resultado del algoritmo presenta la división de las micro-roi Figura 42a y la detección de las líneas de cultivo en la Figura 42b.



a) Generación de micro-ROI b) identificación de líneas de cultivo

Figura 42: Detección de líneas en cultivos de maíz basado en micro-ROI

Identificación de malezas: en esta fase se utilizó paralelismo para la separación de las líneas de cultivo detectadas del resto de vegetación. Para separar las líneas de cultivo se creó una imagen como plantilla que contiene píxeles blancos Figura 43a, siguiendo las líneas trazadas en la imagen de la fase anterior como se indica previamente en la Figura 42b. Los píxeles blancos que se encuentran fuera de las líneas de cultivo se consideran malezas como se presenta en la Figura 43b.

a) Imagen binaria que representa las líneas de cultivo detectadas

b) píxeles que representan a las malezas entre líneas de cultivo

Figura 43: Procesamiento de imágenes para detección de malezas

3.2.3 Algoritmo secuencial con imágenes obtenidas con dron

El algoritmo secuencial desarrollado consta de 4 fases: Segmentación, detección de líneas de cultivo, exclusión de cultivo e identificación de malezas. En la Figura 44 se presenta las fases de procesamiento con imágenes digitales adquiridas con dron para identificar las líneas de cultivo y malezas.

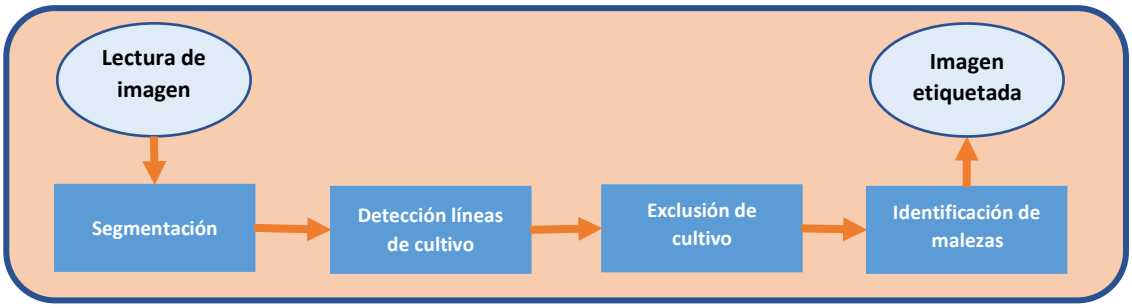
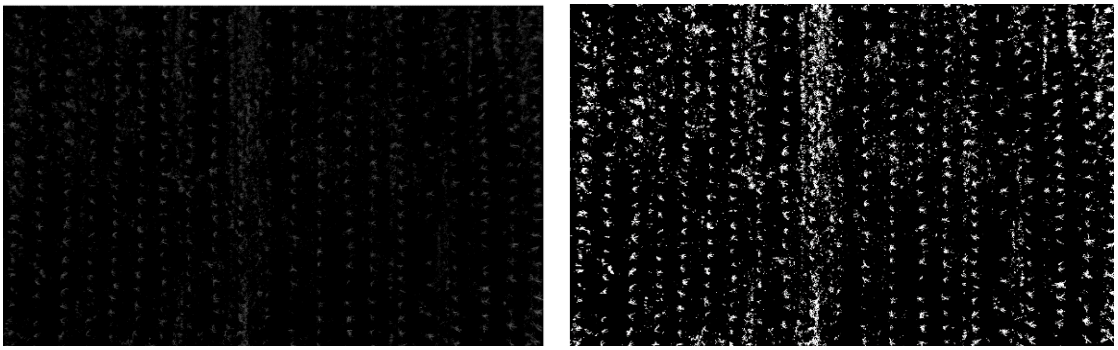


Figura 44: Fases del algoritmo secuencial con imágenes obtenidas con dron.

Segmentación: en esta fase consiste en la distinción e identificación de las plantas en el terreno, sin clasificar entre cultivo y malezas. La utilización ExG (exceso de verde) permite identificar de una manera simple la presencia de vegetación en una imagen. Una vez leída y cargada la imagen se extraen los valores de los canales en rojo, verde y azul (RGB) para obtener una imagen en escala de grises. Con la imagen previamente obtenida se aplica un umbral basado en el método OTSU (García-Santillán & Pajares, 2016) para generar una imagen binaria. En la Figura 45 se presenta el resultado de la operación EXG (45a) y aplicación de umbral a la imagen binaria (45b).



a) Imagen aplicado operación EXG b) Imagen aplicado umbral OTSU

Figura 45: Imagen binarizada con dilatación digital

Detección de líneas de cultivo: considerando que una línea de cultivo, en una imagen, es una acumulación de píxeles verdes siguiendo una alineación específica. En esta fase se aplicó la operación morfológica de apertura a la imagen binaria, la cual dilata y erosiona las regiones blancas con el fin de aumentar el número de píxeles y el tamaño de los puntos blancos, pero a su vez eliminar las regiones más pequeñas que no tenga un tamaño considerable. Una vez realizadas las operaciones anteriores se aplicó la transformada de Hough modificando los valores de los parámetros requeridos para una mejor precisión en la obtención de líneas de cultivo.

La transformada de Hough fue utilizada para detectar líneas rectas, la funcionalidad está basada en la presentación de la recta representada por la ecuación 9

$$y = m\mathrm{x} + \mathrm{b} \qquad (9)$$

En donde m es la pendiente y se representa como delta de y sobre delta de x. Esto se puede definir como b que es la intersección con el eje (y) y c que es la intersección con el eje (x). Adicionalmente se antecede el signo (-) debido a que la pendiente es negativa.

$$m = \frac{\Delta y}{\Delta x} = -\frac{b}{c} \qquad (10)$$

En el cálculo de líneas rectas se encuentran otro tipo de parámetros como rho (ρ) la cual representa la distancia mínima del origen a la recta, por medio de la perpendicular de la recta. Esta perpendicular forma un ángulo con respecto al eje (x), el cual se representa mediante la variable theta (θ). En la Figura 46 se presenta los parámetros necesarios para calcular líneas rectas.

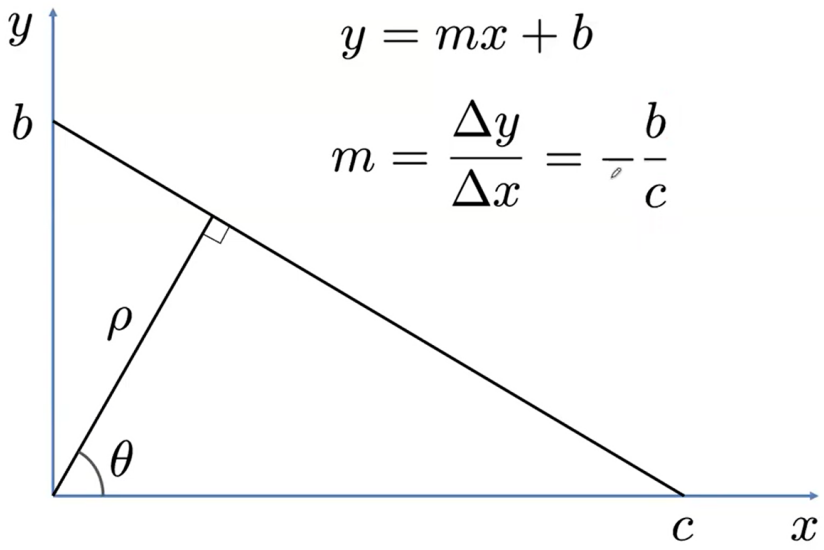


Figura 46: Representación de la recta
(Chung et al., 2014)

Aplicando reglas de trigonometría se obtiene los valores de (b) y (c) con respecto al ángulo (θ). Las ecuaciones 11 y 12 representan los dos valores.

$$b = \frac{\rho}{sin\theta} \qquad c = \frac{\rho}{cos\theta} \tag{11}$$

$$m = -\frac{b}{c} = -\frac{cos\theta}{sin\theta} \tag{12}$$

Una vez obtenidos los valores de la pendiente (m), (b) y (c), se procede a reemplazar en la ecuación 9, dando lugar a la ecuación 13:

$$y = -\frac{cos\theta}{sin\theta}\mathrm{x} + \frac{\rho}{sin\theta} \tag{13}$$

De esta manera se obtiene la ecuación de la recta utilizada en la transformada de Hough, representada por la ecuación 14.

$$\rho = xcos\theta + ysin\theta \tag{14}$$

El origen del sistema de coordenadas se supone que está en el centro del píxel de la esquina superior izquierda. El valor de rho es la distancia perpendicular desde el origen hasta la recta. El valor de theta es el ángulo de la proyección perpendicular desde el origen hasta la recta medido en grados en el sentido de las agujas del reloj desde el eje positivo x. El intervalo de theta es $-90° \leq \theta < 90°$. El ángulo recta es $\theta + 90°$, también medido en el sentido de las agujas del reloj con respecto al eje positivo x (MathWorks, 2021). En la Figura 47 se presentan los elementos necesarios utilizados para aplicar la transformada de Hough en el procesamiento de imágenes digitales.

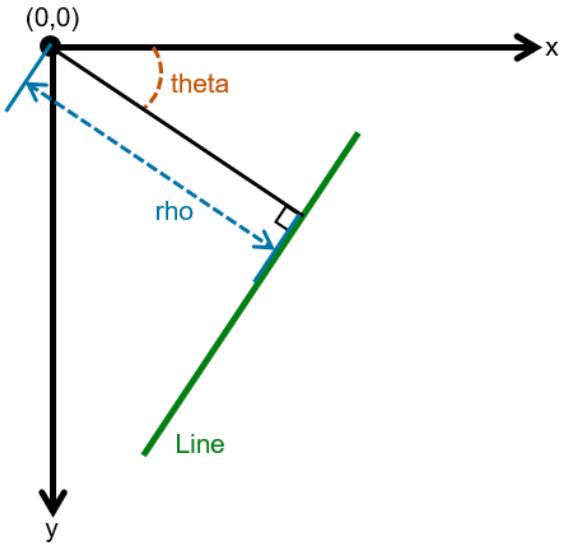


Figura 47: Valores de parámetros de la transformada de Hough (MathWorks, 2021)

La función utiliza la representación paramétrica de una recta: rho = x*cos(theta) + y*sin(theta). La función devuelve rho, la distancia desde el origen hasta la recta a lo largo de un vector perpendicular a la recta, y theta, el ángulo en grados entre el eje x y este vector. La función también devuelve la SHT, H, que es una matriz del espacio de parámetros cuyas filas y columnas corresponden a los valores rho y theta respectivamente (MathWorks, 2021). Los puntos obtenidos de las rectas corresponden al número de líneas encontradas. En la Figura 48 se presenta las líneas obtenidas aplicando la transformada de Hough a una imagen digital agrícola obtenida a 15 metros de altura.

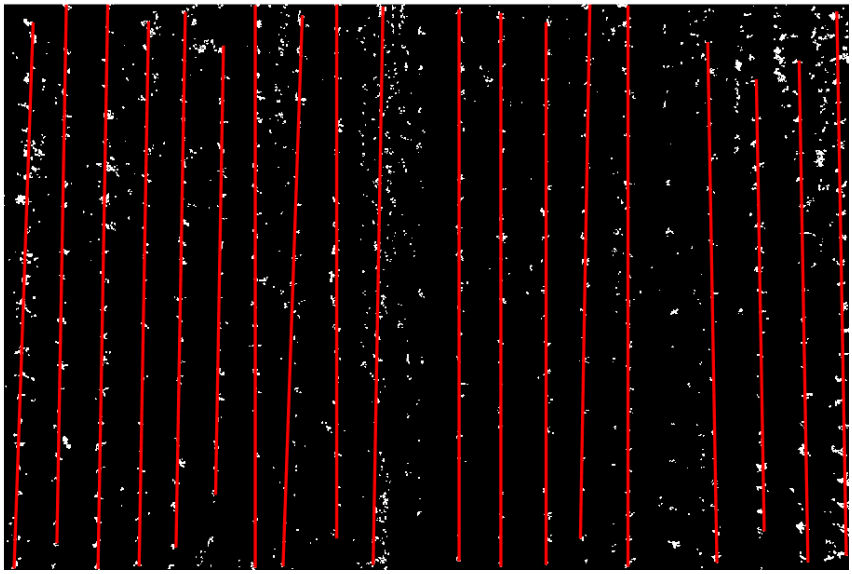


Figura 48: Aplicación de la transformada de Hough

En imágenes con presencia de gran cantidad de malezas entre las líneas de cultivo al aplicar la transformada de Hough para un ángulo, que va desde -4 grados a 4 grados, en intervalos de 0.1 grados, debido a la cantidad de píxeles cercanos a las líneas de cultivo reales, el algoritmo determina malezas como líneas de cultivo o viceversa. Para corregir este problema se utilizó algoritmos adicionales que permitan agrupar líneas y obtener un promedio que se considere como cultivo. En la Figura 49 se presenta los puntos cercanos obtenidos mediante la transformada de Hough que luego son unidos y agrupados para obtener una línea total.

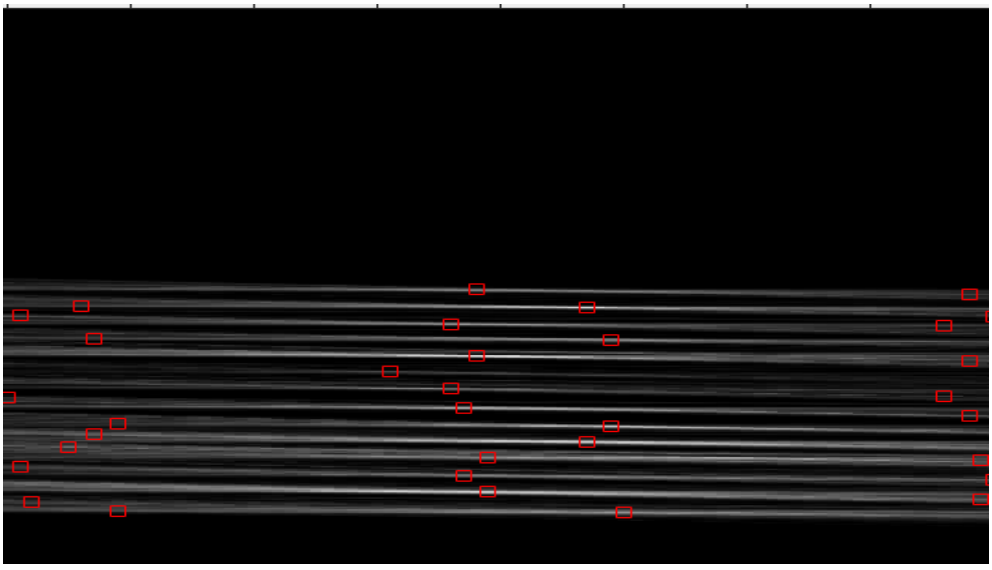
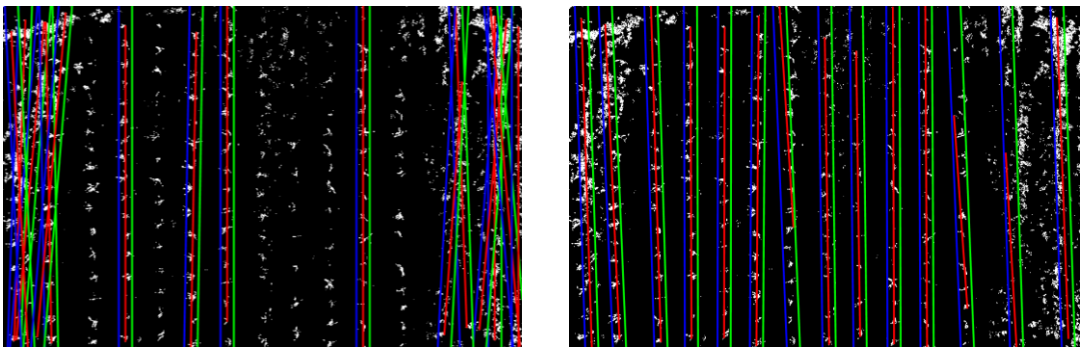


Figura 49: Puntos de Hough para agrupamiento de líneas

Luego de agrupar los puntos se reduce las líneas en exceso y se unifican en una principal. En la Figura 50a se presenta el resultado de una imagen con líneas adicionales que no corresponden a líneas de cultivo y la Figura 50b se encuentra corregido e identificado líneas de cultivo reales.



a) Imagen con exceso de líneas b) Imagen corregido líneas en exceso

Figura 50: Imagen agrícola con presencia excesiva de líneas

Exclusión de cultivo: cada línea de cultivo abarca una pequeña área del terreno. La línea trazada (roja) representa el centro de las plantas de cultivo, cada planta tiene un espacio aproximado de 15 píxeles a cada lado que representa el ancho aproximado de cada planta; con esta situación se agrega dos líneas auxiliares a la izquierda (azul) y derecha (verde) que se consideran los márgenes de separación entre cultivo y malezas, los valores de los márgenes varían de acuerdo a la altura que fue adquirida la imagen. Los píxeles que se encuentran fuera de las 3 líneas se considera malezas. En la Figura 51 se presenta el resultado de las líneas de cultivo en una imagen obtenida a 15 metros de altura, todo el espacio que se encuentra entre los márgenes marcados será excluido para identificar las malezas.

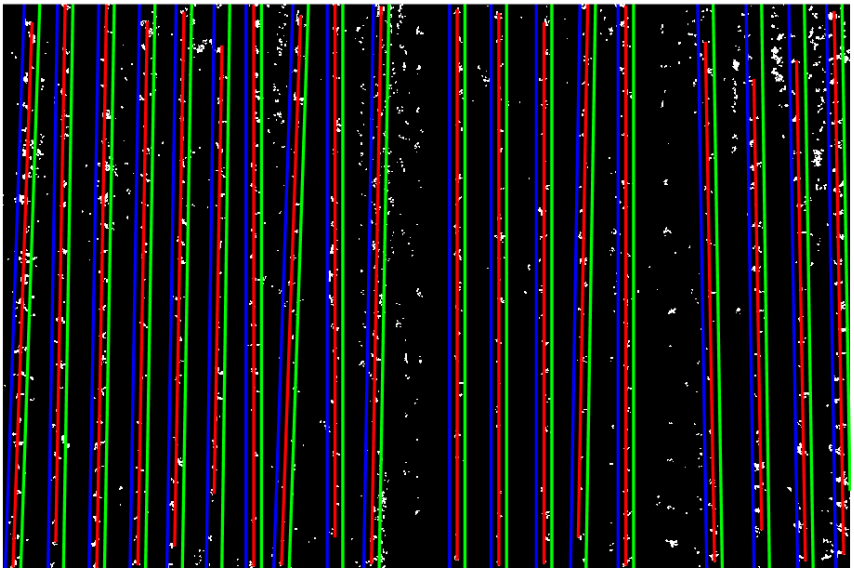


Figura 51: Imagen aplicado márgenes para diferenciar líneas de cultivo

Detección de malezas: para la detección de las malezas se toma como base la imagen obtenida en el paso anterior, se cambió los valores a cero (0) de todos los píxeles que se encuentran dentro de los márgenes porque se consideran líneas de cultivo. Los píxeles que se encuentran fuera de los márgenes con valores uno (1) se consideran malezas. En la imagen binaria obtenida, existen algunos píxeles blancos que son muy pequeños y probablemente no corresponden a malezas; para evitar la confusión se aplicó la operación morfológica de erosión disminuyendo y agrupando las regiones blancas. En la Figura 52 se presenta el resultado de la imagen binaria obtenida después de eliminar los márgenes de las líneas de cultivo para considerar los píxeles blancos como malezas.

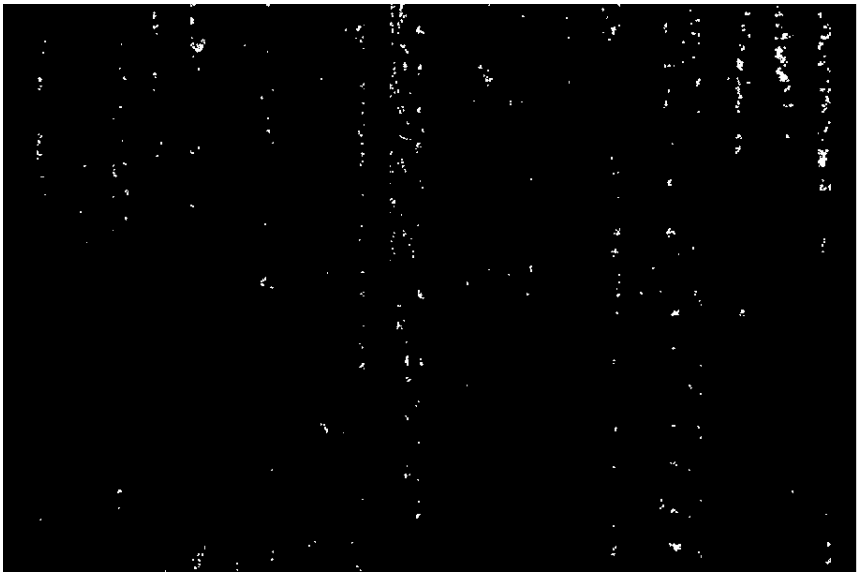


Figura 52: Imagen binaria eliminado las líneas de cultivo

Para mejorar la discriminación de las malezas en esta fase, se realizó cambios necesarios al algoritmo con el objetivo de considerar vegetación intra-cultivo (malezas entre las plantas de cultivo), para ello se detectaron las plantas presentes construyendo un elemento morfológico de disco de radio de 10 píxeles. Luego se aplica la dilatación a la imagen resaltando las figuras de las plantas. Para mejorar la precisión se hace un filtro dilatación considerando solo las áreas de interés. En la Figura 53 se presenta el resultado de la implementación con dilatación para clasificar mediante píxeles las plantas de cultivo.

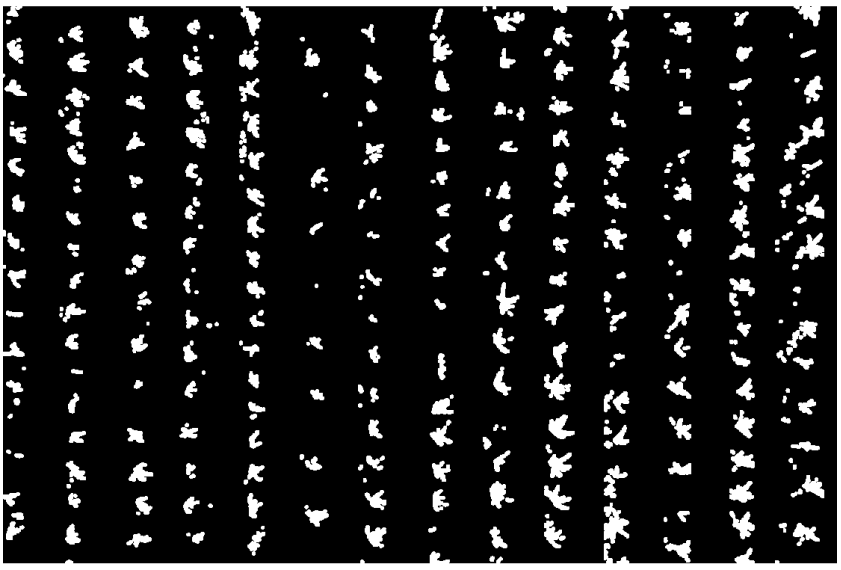


Figura 53: Imagen con plantas aplicado dilatación digital

Una vez que se han dilatado las plantas se procedió a identificar las regiones de interés (plantas). Las áreas detectadas son representadas mediante puntos en cajas de texto (boundingBox) y los centroides de las regiones se almacenan en arreglos por separado para posteriores operaciones en la imagen. Con los datos almacenados se recorre todas las regiones y se grafica las cajas de texto de las regiones detectadas. En la Figura 54 se presenta el resultado del proceso de etiquetación de las plantas detectadas en una imagen obtenida a 5 metros de altura, el mismo proceso se realizó con las imágenes de 5 y 10 metros de altura de las 4 semanas de seguimiento.

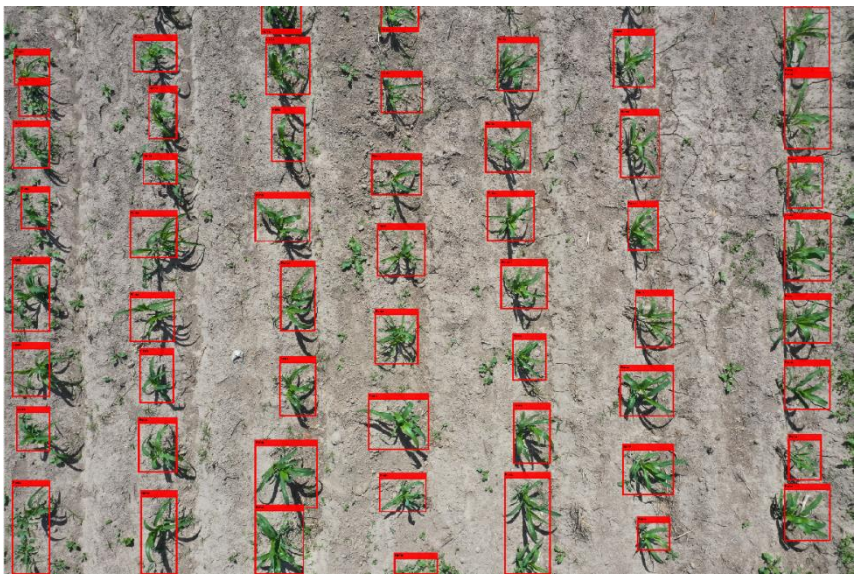
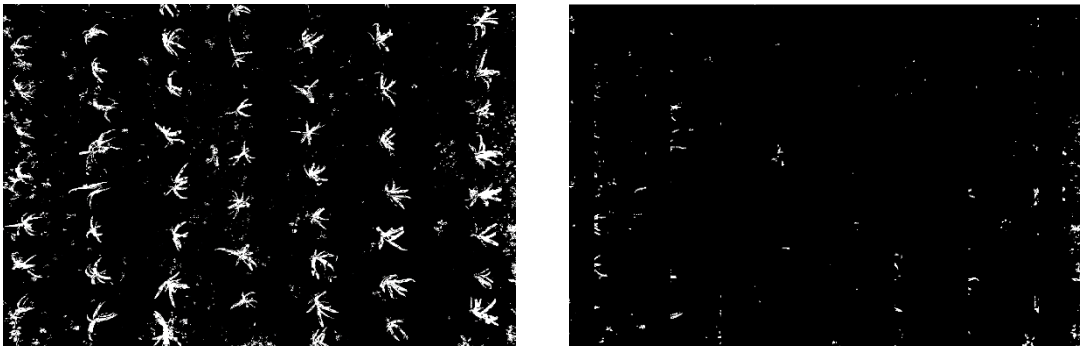


Figura 54: Plantas identificadas en las líneas de cultivo

Para determinar la cantidad de malezas presente en la imagen se realizó un cálculo matemático para obtener un porcentaje promedio a través del conteo de píxeles blancos en la imagen original e imagen procesada excluido las plantas detectadas en cada línea de cultivo como se presenta previamente en la Figura 54. En la Figura 55a se presenta los píxeles de una imagen adquirida a 5 metros de altura con toda la vegetación y en la Figura 55b se presenta únicamente los píxeles considerados como malezas que se encuentran fuera y entre las plantas de cultivo.



a) Imagen binarizada original

b) Imagen binarizada excluido plantas de cultivo

Figura 55. Binarización de la imagen para excluir plantas de cultivo

Tomando en cuenta que la mayoría de los píxeles que pertenecen a malezas se encuentran localizados entre las líneas de cultivo y la mayoría de píxeles que pertenecen a las plantas de cultivo se encuentran en cada línea detectada. Así se realiza la regla de tres entre la matriz binaria de toda la vegetación (Figura 55a) y la matriz binaria de la vegetación excluido las plantas de cultivo (Figura 55b). Los píxeles blancos resultantes de la operación corresponden al porcentaje de maleza detectado.

3.2.4 Algoritmo paralelo con imágenes obtenidas con dron

El diseño del algoritmo se basa en la metodología de Foster's (Weiss, 2014), incluyendo 4 etapas recomendadas (Partición, Comunicación, Agrupación y Asignación). Adicionalmente se consideró la arquitectura del lenguaje de programación Matlab (MathWorks, 2020). Las 4 fases del algoritmo secuencial son separadas en 4 particiones (Segmentación, Detección, exclusión, comparación). Cada partición tiene diferentes tareas que son paralizadas con funciones y sentencias que aprovechan el paralelismo en procesadores con varios cores como se indicó previamente en la Figura 39. La estructura del algoritmo propuesto se presenta en la Figura 56, cada fase del algoritmo propuesto agrupa diferentes tareas que posteriormente son asignadas al procesador parametrizando el número de núcleos disponibles.

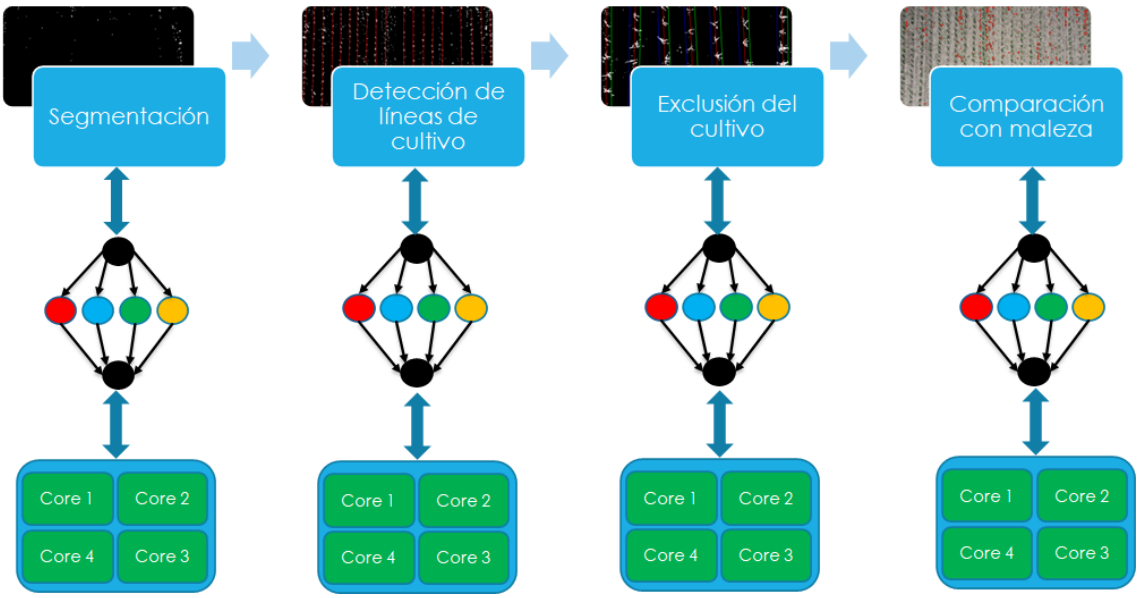


Figura 56: Modelo de algoritmo adaptado a la metodología de Foster's

Las tareas en cada fase se realizan operaciones de procesamiento de imágenes que se distribuyen de acuerdo con el número de procesadores disponibles y parametrizados por Matlab. En la Figura 57 se presenta el modelo básico del procesamiento de las actividades asignadas al procesador por cada partición. El hilo principal (contiene una cruz) de cada núcleo distribuye las tareas en paralelo a todos los hilos disponibles. Los hilos internos trabajan por separado, pero comparten memoria hasta finalizar todas las tareas asignadas. El hilo principal es el responsable de recuperar todos los datos de cada tarea procesada. Al final cada hilo consolida los resultados de todas las tareas paralelizadas.

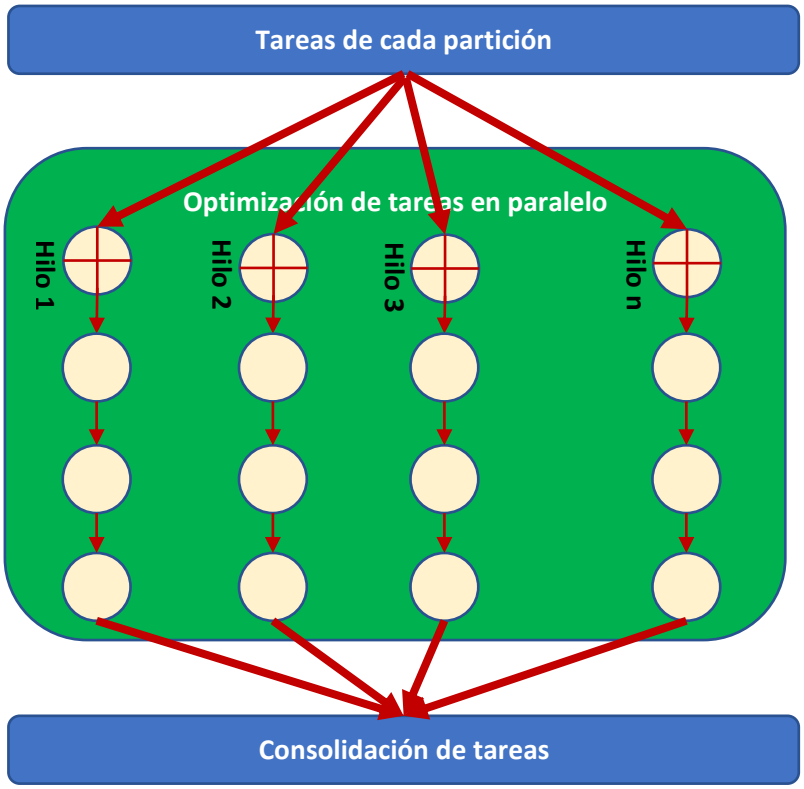


Figura 57: Procesamiento de imágenes con paralelismo

Los resultados de detección de líneas de cultivo y malezas con imágenes digitales obtenidas con el dron son iguales a las obtenidas en el algoritmo secuencial. La diferencia se da en los tiempos de ejecución de acuerdo con el número de núcleos utilizados para el análisis y procesamiento de imágenes mediante las tareas asignadas al procesador multicore.

Capítulo 4. Resultados

Las imágenes recolectadas presentan propiedades y características diferentes dependiendo de la semana (1-4) en que fueron realizadas las capturas, por tal motivo se dividieron los resultados en cuatro partes correspondientes al número de semanas de seguimiento del cultivo. Durante las 4 semanas de recolección de imágenes existió lluvias frecuentes, por lo que el crecimiento del cultivo al igual que las malezas fue muy rápido, haciendo que se disperse la vegetación por el terreno y en ciertos sectores crecieron grandes cantidades de malezas.

4.1 Procesamiento de imágenes

Con el propósito de determinar la precisión del algoritmo, se realizó la etiquetación manual de líneas de cultivo y plantas existentes. Para determinar la cantidad de malezas existentes en el cultivo se consideró toda la vegetación menos las plantas de cultivo encontradas en cada línea. Las imágenes recolectadas corresponden a las 4 semanas de seguimiento del cultivo; sin embargo, en la primera semana fue difícil la detección de líneas de cultivo y malezas. Debido al problema mencionado se consideraron las 3 siguientes semanas de seguimiento a los cultivos de maíz. Para obtener mejores resultados de procesamiento, se seleccionó 300 imágenes adquiridas con cámara y dron, evitando factores como el desenfoque, oscuridad o poca notoriedad del cultivo a simple vista.

Las métricas consideradas en la evaluación de los resultados obtenidos por los algoritmos fueron las siguientes: Líneas de cultivo reales (LCR), líneas de cultivo detectadas (LCD), porcentaje malezas detectada (PMD), tiempo secuencial (TS) y tiempo promedio paralelo (TPP).

LCR: son las líneas visibles al ojo humano que son detectadas de forma natural en las imágenes de diferentes alturas. En la Figura 58 se presenta una imagen obtenida desde dron a 5 metros de altura en la cuarta semana de seguimiento, las plantas de cultivo se pueden diferenciar con facilidad del resto de vegetación que se considera como malezas en el procesamiento de los algoritmos. De la misma manera se procedió a etiquetar manualmente en las imágenes de 10 y 15 metros de altura

Figura 58: Líneas etiquetadas manualmente en imagen a 5 metros

LCD: son las líneas de cultivo detectadas por los algoritmos luego del procesamiento de las imágenes capturadas con el dron a alturas de 5, 10 y metros. En la Figura 59 se presenta una imagen obtenida a 5 metros de altura en la cuarta semana de seguimiento. Los algoritmos diferencian las plantas de cultivo del resto de vegetación y son marcadas con 3 líneas (que se considera como malezas en el procesamiento de los algoritmos. De la misma manera se procesó imágenes de 10 y 15 metros de altura de las anteriores semanas.

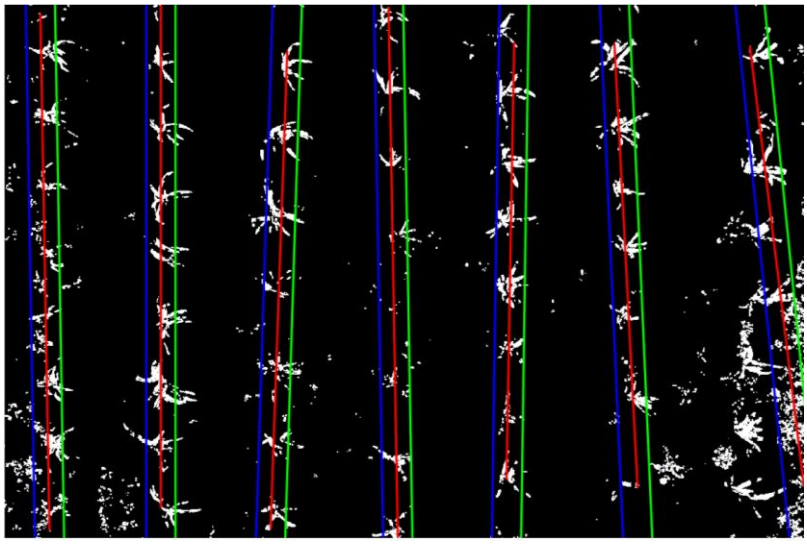


Figura 59: Imagen etiquetada por el algoritmo con líneas de cultivo

PMD: es la cantidad de vegetación obtenida en el procesamiento de las imágenes excluyendo las plantas de cultivo existentes en cada línea de cultivo

detectada. En la Figura 60 se presenta una imagen obtenida a 5 metros de altura en la cuarta semana de seguimiento. Los algoritmos diferencian las plantas de cultivo del resto de vegetación y son marcadas con cajas de texto (rojo) para luego ser binarizada la imagen excluyendo las plantas detectadas. El porcentaje obtenido representa el valor calculado por los algoritmos en relación con las malezas totales reales presentes en la imagen digital. Un valor alto del porcentaje de maleza es importante para determinar la precisión del algoritmo, lo ideal sería llegar a calcular el 100% de maleza mediante los algoritmos implementados. De la misma manera se procesó imágenes de 10 y 15 metros de altura de las anteriores semanas

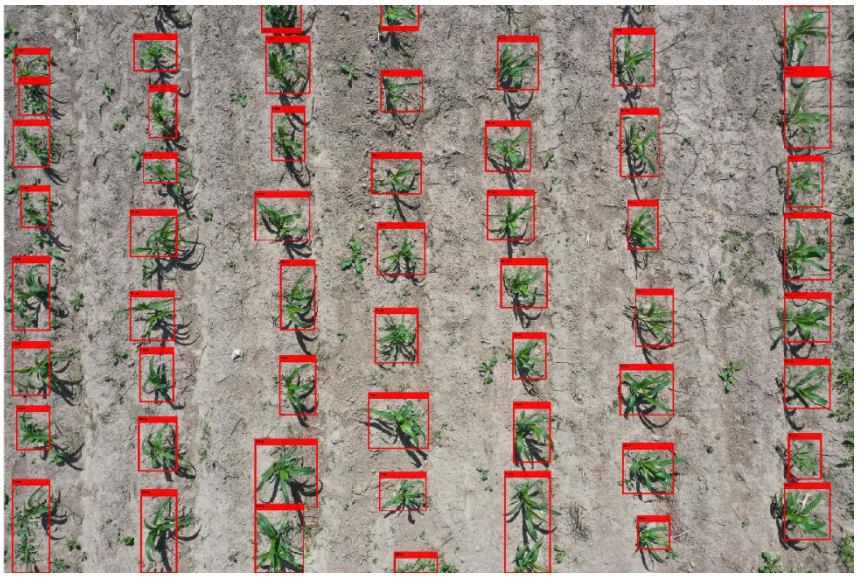


Figura 60: Imagen etiquetada por el algoritmo con plantas de cultivo

TS: es el tiempo de ejecución de los algoritmos secuenciales durante el procesamiento de las imágenes digitales.

TPP: es el tiempo promedio de todos los tiempos obtenidos con diferentes procesadores (2, 4, 6, 8). Es un tiempo resumen que puede ser utilizado como parámetro para comparar el tiempo promedio con paralelismo contra el tiempo secuencial.

4.1.1 Primera semana

En la primera semana se tuvo inconvenientes con la visualización del cultivo en las imágenes adquiridas debido al tamaño de las plantas, aún en las alturas más

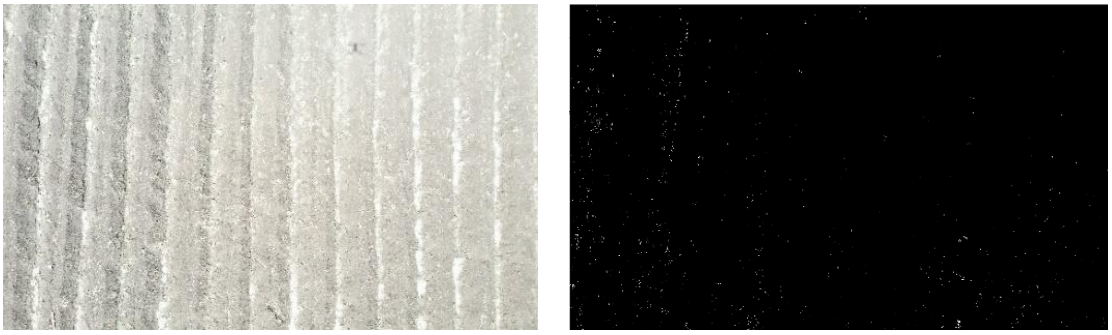
bajas (5 m), por lo que fue difícil distinguir incluso para el ojo humano. Además, la iluminación solar hizo casi imperceptible ver la presencia de vegetación en las imágenes; sin embargo, se configuró el brillo y el contraste con el fin de mejorar la calidad de las imágenes sin obtener resultados positivos. Estos factores afectan directamente a la detección de líneas de cultivo y malezas a pesar de los filtros y cambios aplicados en los algoritmos.

En la Tabla 6 se detallan los parámetros establecidos en el código de Matlab para la primera semana. Los parámetros modificados en el código fueron: (i) el tamaño del umbral, (ii) el ángulo de mínimo y máximo de la línea, (iii) el tamaño mínimo de la línea, (iv) entre otros.

Tabla 6. Parámetros en el código establecido para la primera semana

	5 metros	10 metros	15 metros
Valor del umbral de amplitud	6	3	2
Ángulo mínimo de la línea	-6	-5	-5
Ángulo máximo de la línea	10	5	5
Rango de líneas vecinas	[301 3]	[121 3]	[401 3]
Distancia entre líneas	2200 px	2500 px	2000 px
Tamaño mínimo de la línea	500 px	750 px	1500 px
Desplazamiento a la izquierda de la línea	150 px	70 px	50 px
Desplazamiento a la derecha de la línea	150 px	70 px	50 px
Área máxima de reconocimiento de líneas	950 px	440 px	500 px

En la Figura 61a se presenta la imagen original de la muestra tomada a 10 metros de altura en la primera semana de estudio. En la Figura 61b se presenta puntos muy pequeños de la vegetación en los que es muy difícil considerar cuales corresponden a malezas y definir las líneas de cultivo.



a) Imagen original b) Imagen binarizada

Figura 61: Imagen a 10 metros de altura en la primera semana de crecimiento.

A pesar de los filtros y cambios aplicados a las imágenes y haber modificado los parámetros de funcionalidad al algoritmo como la dilatación de regiones blancas no bastaron para poder identificar las líneas de cultivo.

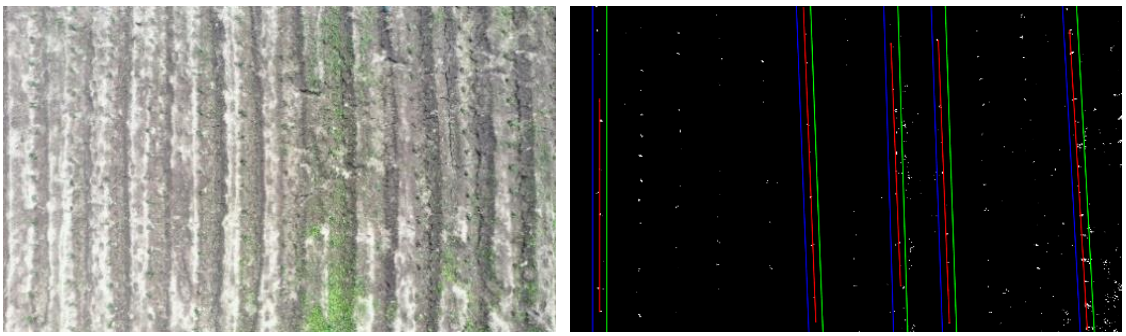
4.1.2 Segunda semana

En la segunda semana, el cultivo creció lo suficiente como para ser visualizado en las imágenes. Sin embargo, el tamaño de las malezas fue similar al del cultivo, por lo cual seguía siendo difícil distinguir entre ambos, sobre todo al momento del proceso de erosión al reducir notablemente tanto las regiones de interés (cultivo) como las malezas. En algunos sectores del cultivo las malezas se presentan en mayor densidad de vegetación. En la Tabla 7 se detallan los parámetros establecidos en el código para la segunda semana para las tres alturas de las imágenes.

Tabla 7. Parámetros en el código establecido para la segunda semana.

	5 metros	10 metros	15 metros
Valor del umbral de amplitud	5	3	3
Ángulo mínimo de la línea	-5	-5	-5
Ángulo máximo de la línea	5	5	5
Rango de líneas vecinas	[501 3]	[351 3]	[501 3]
Distancia entre líneas	2000 px	2000 px	1700 px
Tamaño mínimo de la línea	500 px	750 px	1500 px
Desplazamiento a la izquierda de la línea	150 px	70 px	50 px
Desplazamiento a la derecha de la línea	150 px	70 px	50 px
Área máxima de reconocimiento de líneas	750 px	220 px	300 px

En la Figura 62 se presenta la imagen original con presencia de alta cantidad de malezas en algunas líneas de cultivo. En la Figura 62b se presenta la identificación de líneas de cultivos en una imagen tomada a 10 metros de altura, la detección de líneas de cultivo en algunas imágenes no fue precisa debido al tamaño reducido de las plantas de cultivo y alta presencia de malezas en algunos sectores del cultivo. De manera similar se procesó imágenes con alturas de 5 y 15 metros con similares resultados.



a) Imagen con alta densidad de vegetación segunda semana

b) líneas detectadas segunda semana

Figura 62*:* Imagen procesada en la segunda semana

En esta semana de estudio, se verifica que existe zonas del terreno donde la presencia de vegetación es tanta que a pesar de aplicar parámetros en los procesos morfológicos el algoritmo presenta fallas en el reconocimiento de malezas debido a la acumulación de regiones blancas en un mismo lugar, haciendo que se sobrepongan líneas o no se identifique correctamente las líneas de cultivo existentes en la imagen digital. En la Tabla 8 se detallan los resultados promedio obtenidos en la detección de líneas de cultivo, malezas y tiempos de ejecución con imágenes de la segunda semana.

Tabla 8. Resultados en la segunda semana

Altura (metros)	Líneas de cultivo/malezas			Tiempos de ejecución (segundos)				
	LCR	LCD	PMD	TS	2 núcleos	4 núcleos	6 núcleos	8 núcleos
Cámara (1.5)	4	4	88.14	4.92	3.15	3.10	2.85	2.81
5	7	7	92.17	6.23	4.26	4.05	3.87	3.84
10	14	13	89.13	7.98	5.18	4.87	4.52	4.61
15	20	18	87.35	8.81	6.75	6.14	5.85	5.67

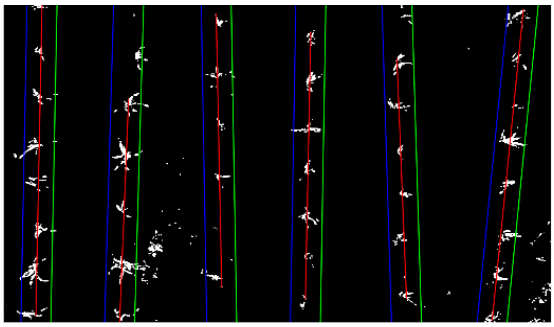
4.1.3 Tercera semana

En la tercera semana los resultados cambiaron significativamente. Las líneas de cultivo son visibles para el ojo humano como para el procesamiento de los algoritmos con alturas de 5, 10 y 15 metros. En la Tabla 9 se detallan los parámetros establecidos en el código para la tercera semana para las 3 alturas de las imágenes.

Tabla 9: Parámetros en el código establecido para la tercera semana

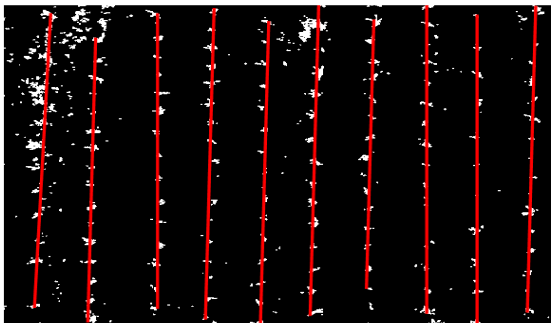
	5 metros	10 metros	15 metros
Valor del umbral de amplitud	5	4	3
Ángulo mínimo de la línea	-15	-5	-4
Ángulo máximo de la línea	15	5	4
Rango de líneas vecinas	[801 5]	[641 2]	[501 3]
Distancia entre líneas	1200 px	2500 px	1700 px
Tamaño mínimo de la línea	2000 px	2000 px	1500 px
Desplazamiento a la izquierda de la línea	150 px	70 px	50 px
Desplazamiento a la derecha de la línea	150 px	70 px	50 px
Área máxima de reconocimiento de líneas	750 px	400 px	300 px

En la Figura 63 se presentan los resultados obtenidos en el procesamiento de las imágenes para la identificación de líneas de cultivo y malezas con imágenes a 5 metros (63a, 63b), 10 metros (63c, 63d) y 10 metros (63e, 63f).

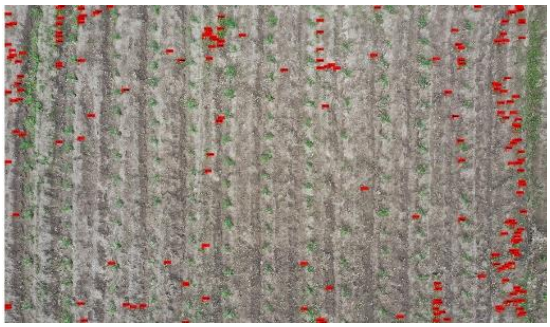


(a) líneas de cultivo detectadas a 5 metros tercera semana

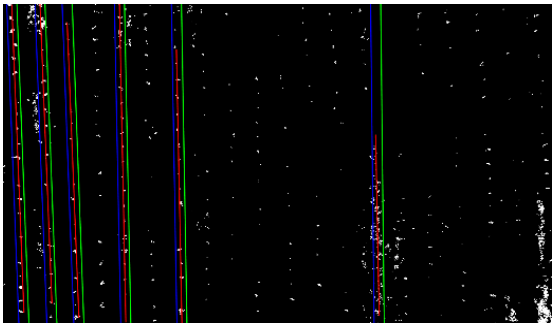
(b) malezas detectadas a 5 metros tercera semana.



(c) líneas de cultivo detectadas a 10 metros tercera semana



(d) malezas detectadas a 10 metros tercera semana



(e) líneas de cultivo detectadas a 15 metros tercera semana

(f) malezas detectadas a 5 metros tercera semana

Figura 63*:* Imágenes procesadas en la tercera semana

En esta semana, la precisión en la detección de líneas de cultivo fue superior, casi infalible con imágenes a 5 metros de altura, mientras que con imágenes a 10 metros de altura hubo ciertos errores de detección en imágenes en las que existía exceso de malezas, debido a que se aglomeraban las regiones blancas y se trazaban varias líneas en una sola parte del terreno. En las imágenes tomadas a 15 metros también se detectaron errores en la detección de líneas de cultivo y malezas debido a que algunas zonas presentaban poca vegetación (posiblemente la semilla no germinó). En la Tabla 10 se detallan los resultados

promedio obtenidos durante el procesamiento de imágenes en la tercera semana para la detección de líneas de cultivo, malezas y tiempos de ejecución.

Tabla 10. Resultados en la tercera semana

Altura (metros)	**Líneas de cultivo/malezas**			**Tiempos de ejecución (segundos)**				
	LCR	**LCD**	**PMD**	**TS**	**2 núcleos**	**4 núcleos**	**6 núcleos**	**8 núcleos**
Cámara (1.5)	4	4	87.14	5.24	3.15	3.10	2.85	2.81
5	7	7	91.38	6.78	4.62	4.31	3.91	3.64
10	14	12	88.95	8.63	5.26	4.99	4.61	4.44
15	20	17	87.15	9.14	6.54	5.42	5.24	4.83

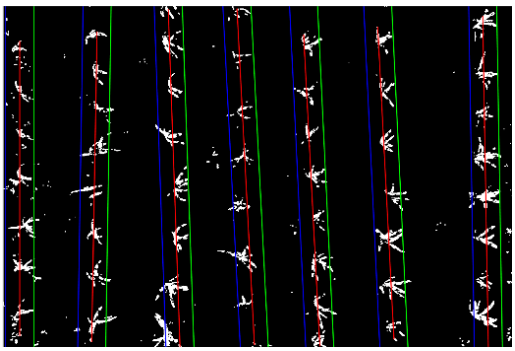
4.1.4 Cuarta semana

La última semana de seguimiento del cultivo en la que se obtuvieron los mejores resultados en la precisión del algoritmo. Para esta etapa, el crecimiento de las hojas es considerable, haciendo tratable la exclusión de cultivo y malezas en las tres alturas (5, 10 y 15 metros). En la Tabla 11 se detallan los parámetros establecidos en el código para la cuarta semana con las tres alturas de las imágenes.

Tabla 11: Parámetros en el código establecido para la cuarta semana

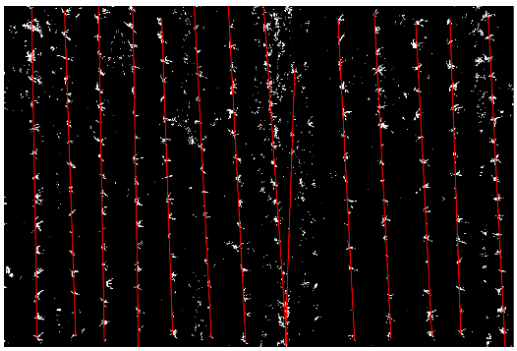
	5 metros	10 metros	15 metros
Valor del umbral de amplitud	5	4	5
Ángulo mínimo de la línea	-15	-5	-5
Ángulo máximo de la línea	15	5	5
Rango de líneas vecinas	[1001 3]	[411 3]	[401 3]
Distancia entre líneas	1200 px	1500 px	1250 px
Tamaño mínimo de la línea	2000 px	2000 px	1700 px
Desplazamiento a la izquierda de la línea	150 px	70 px	50 px
Desplazamiento a la derecha de la línea	150 px	70 px	50 px
Área máxima de reconocimiento de líneas	750 px	400 px	300 px

En la Figura 64 se presentan los resultados obtenidos en el procesamiento de las imágenes de la cuarta semana para la identificación de líneas de cultivo y malezas con imágenes a 5 metros (64a, 64b), 10 metros (64c, 64d) y 10 metros (64e, 64f).

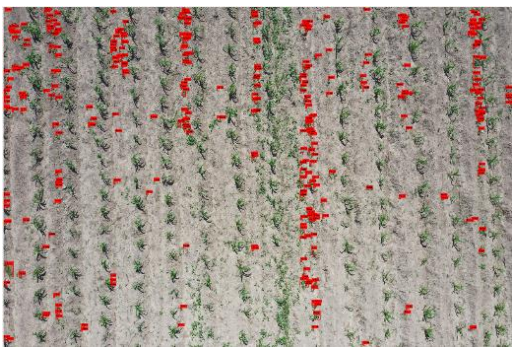


(a) líneas de cultivo detectadas a 5 metros cuarta semana

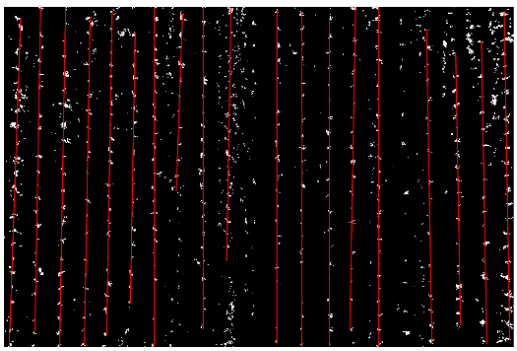
(b) líneas de cultivo detectadas a 5 metros cuarta semana



(c) líneas de cultivo detectadas a 10 metros cuarta semana.



(d) malezas detectadas a 10 metros cuarta semana



(e) líneas de cultivo detectadas a 15 metros cuarta semana

(f) malezas detectadas a 15 metros cuarta semana

Figura 64*:* Imágenes procesadas en la cuarta semana

No obstante, durante la cuarta semana se presentan ciertos problemas tratados en las semanas anteriores como sobreposición de líneas o ignorar líneas de cultivo presentes en la imagen. En la Tabla 12 se detallan los resultados promedio obtenidos durante el procesamiento de imágenes en la cuarta semana para la detección de líneas de cultivo, malezas y tiempos de ejecución.

Tabla 12. Resultados en la cuarta semana

Altura (metros)	Líneas de cultivo/malezas			Tiempos de ejecución (segundos)				
	LCR	LCD	PMD	TS	2 núcleos	4 núcleos	6 núcleos	8 núcleos
Cámara (1.5)	4	4	84.71	5.97	5.02	4.51	4.35	4.24
5	7	7	95.84	7.05	5.25	4.79	4.10	3.71
10	14	12	94.26	8.08	5.68	5.23	4.85	4.87
15	20	18	93.28	9.41	6.36	5.56	4.48	4.54

El procesamiento de imágenes con alturas superiores a 15 metros dificultó la detección de líneas de cultivo y malezas debido a la similitud de los píxeles de cultivo y vegetación. Después de la cuarta semana el tamaño de malezas, en algunas partes del cultivo es similar o superior al tamaño de las plantas de cultivo dificultando la precisión del algoritmo secuencial y paralelo.

4.2 Medidas de rendimiento

En la evaluación de los algoritmos se consideraron algunas de las medidas más utilizadas para determinar el rendimiento de una arquitectura paralela. Se aplicaron las medidas de speedup, eficiencia, rapidez y costo computacional. Las fórmulas de cada medida se indica previamente en la sección 2.4.

4.2.1 Tiempos de ejecución

Los tiempos de ejecución evaluados durante las 4 semanas de seguimiento al fueron cambiando debido a la cantidad de píxeles que representan a la vegetación del cultivo. En la primera semana no se realizaron mediciones debido a mínima detección de líneas de cultivo. Desde la 2 semana que fue posible identificar líneas de cultivo por el tamaño de las plantas se realizaron mediciones tanto para el algoritmo secuencial como para el algoritmo paralelo. Los datos obtenidos en el proceso de ejecución (segundos) de los algoritmos de la segunda

semana se presentan en la Tabla 13 (segunda semana), Tabla 14 (tercera semana) y Tabla 15 (Cuarta semana). Los tiempos obtenidos del algoritmo paralelo son inferiores a los tiempos del algoritmo secuencial.

Tabla 13: Tiempos promedio algoritmos en la segunda semana.

Altura (metros)	**Tiempos de ejecución (segundos)**					
	Secuencial	**2 núcleos**	**4 núcleos**	**6 núcleos**	**8 núcleos**	**TPP**
Cámara (1.5)	4.92	3.15	3.10	2.85	2.81	2.98
5	6.23	4.26	4.05	3.87	3.84	4.01
10	7.98	5.18	4.87	4.52	4.61	4.80
15	8.81	6.75	6.14	5.85	5.67	6.10

Tabla 14: Tiempos promedio algoritmos en la tercera semana.

Altura (metros)	**Tiempos de ejecución (segundos)**					
	Secuencial	**2 núcleos**	**4 núcleos**	**6 núcleos**	**8 núcleos**	**TPP**
Cámara (1.5)	5.24	3.15	3.1	2.85	2.81	2.98
5	6.78	4.62	4.31	3.91	3.64	4.12
10	8.63	5.26	4.99	4.61	4.44	4.83
15	9.14	6.54	5.42	5.24	5.26	5.51

Tabla 15: Tiempos promedio algoritmos en la cuarta semana.

Altura (metros)	**Tiempos de ejecución (segundos)**					
	Secuencial	**2 núcleos**	**4 núcleos**	**6 núcleos**	**8 núcleos**	**TPP**
Cámara (1.5)	5.97	5.02	4.51	4.35	4.24	4.53
5	7.05	5.25	4.79	4.10	3.71	4.46
10	8.08	5.68	5.23	4.35	4.37	5.16
15	9.41	6.36	5.56	4.48	4.54	5.24

Los tiempos de ejecución en la cuarta semana con imágenes de 10 y 15 metros de altura son más altos utilizando máximo de número núcleos del procesador, las imágenes tienen mayor cantidad píxeles con vegetación y existe sobrecarga de tareas asignadas para procesar las imágenes.

Las gráficas de comportamiento de los algoritmos obtenidos en el proceso de ejecución (segundos) se presentan en la Figura 65 (segunda semana), Figura 66 (tercera semana) y Figura 67 (Cuarta semana).

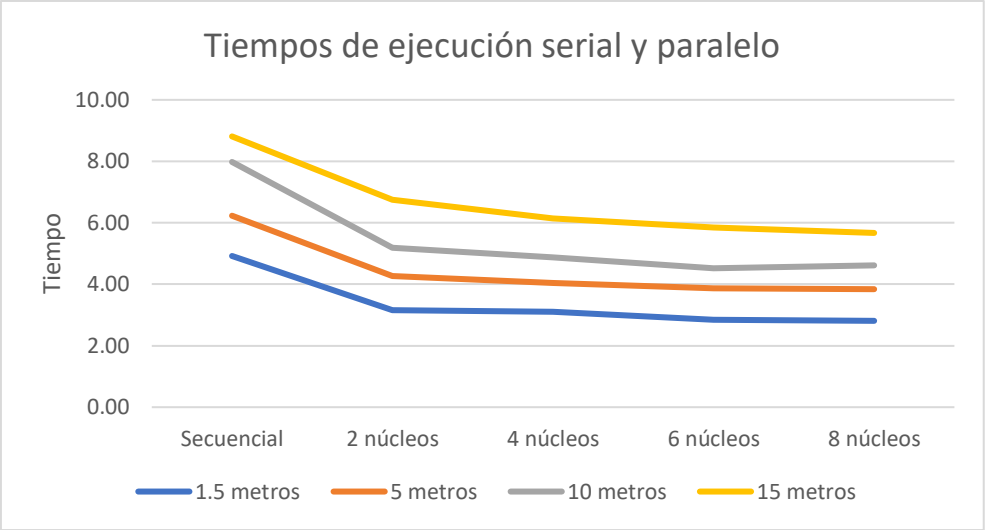


Figura 65: Tiempos de ejecución algoritmos para la segunda semana

En la segunda semana, los tiempos de ejecución para el procesamiento de imágenes del cultivo disminuyen a medida que aumenta el número de núcleos de forma creciente dependiendo del número de núcleos utilizados. A partir de 4 o más núcleos utilizados en la ejecución los tiempos decrecen en menor proporción al tiempo obtenido con 2 núcleos; sin embargo, el tiempo de ejecución del algoritmo paralelo con 8 núcleos (capacidad máxima del equipo utilizado) es inferior en un 35.64% comparado con el tiempo secuencial en el procesamiento de imágenes capturadas a 15 metros de altura (altura máxima experimentada).

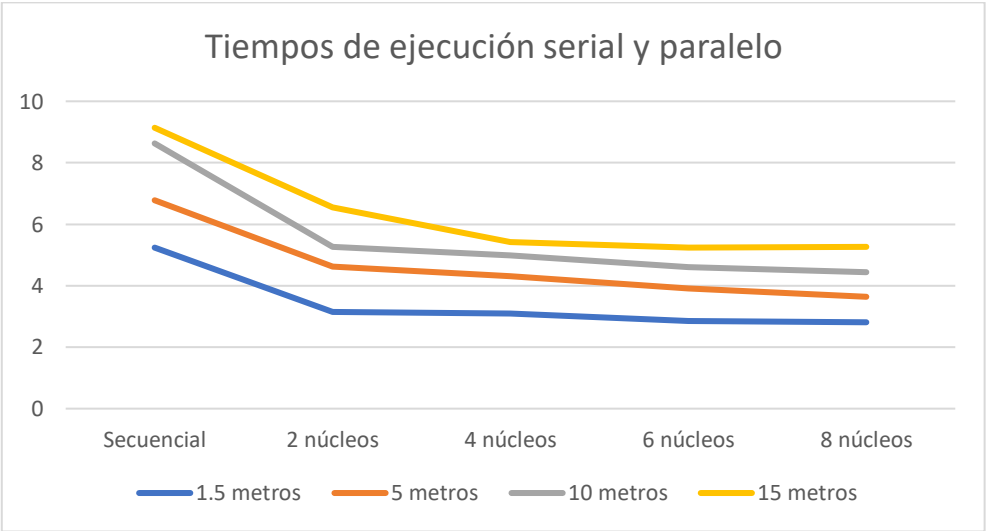


Figura 66: Tiempos de ejecución algoritmos para la tercera semana

En la tercera semana, los tiempos de ejecución para el procesamiento de imágenes de las tres semanas correspondientes a las tres alturas experimentadas disminuyen de manera similar a los tiempos de ejecución de la

segunda semana considerando el número de núcleos utilizados. Con las imágenes obtenidas a 15 metros de altura el tiempo de ejecución evaluado con 8 núcleos incrementa, si se relaciona con el tiempo obtenido con 6 núcleos. A pesar de obtener un tiempo de ejecución superior con la máxima cantidad de núcleos, es 42.45% más rápido comparado con el tiempo secuencial.

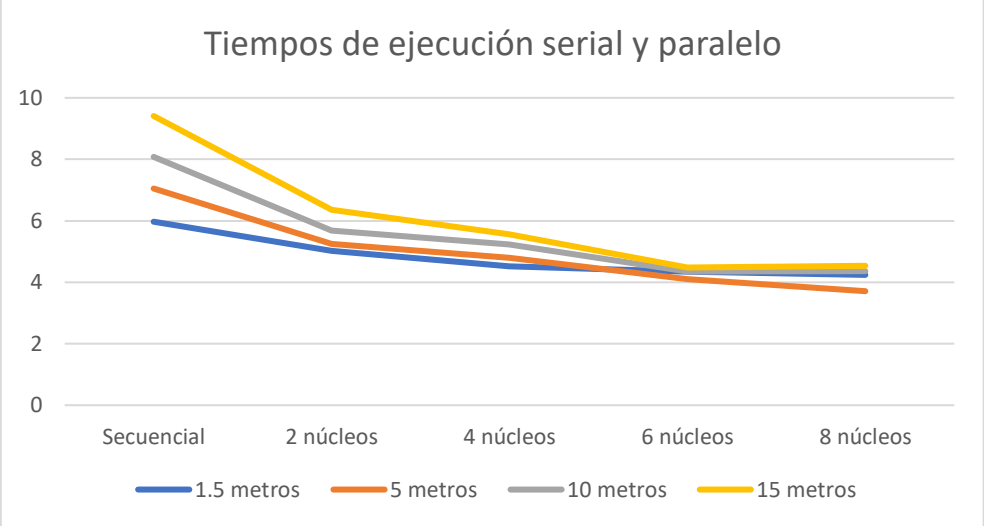


Figura 67: Tiempos de ejecución algoritmos para la cuarta semana

Como resultado de las ejecuciones de los algoritmos implementados se evidencia una reducción de tiempo significante

En la cuarta semana, los tiempos de ejecución del algoritmo paralelizado para el procesamiento de imágenes del cultivo son menores en relación a las dos semanas anteriores, especialmente con la utilización de 6 y 8 núcleos. De manera similar a la tercera semana en ciertas imágenes el tiempo obtenido con 8 núcleos es superior al tiempo obtenido con 6 núcleos. La vegetación existente incrementó en varios sectores del cultivo y por ende al procesar dichas imágenes se genera una sobrecarga de trabajo en las tareas paralelizadas dando como resultado tiempos superiores. El tiempo obtenido utilizando 8 núcleos se reduce en un 51.75% comparado con el tiempo secuencial en el procesamiento de imágenes capturadas a 15 metros de altura.

4.2.2 Aceleración (Speedup)

Los valores de la aceleración obtenidos de los algoritmos después de aplicar la ecuación 3 de la sección 2.4.3 para las 3 semanas de seguimiento se presentan en la Tabla 16 (segunda semana), Tabla 17 (tercera semana) y Tabla 18 (cuarta semana).

Tabla 16: Aceleración algoritmos en la segunda semana

Altura (metros)	Aceleración (segundos)				
	Secuencial	2 núcleos	4 núcleos	6 núcleos	8 núcleos
Cámara (1.5)	1.00	1.56	1.59	1.73	1.75
5	1.00	1.46	1.54	1.61	1.62
10	1.00	1.54	1.64	1.77	1.73
15	1.00	1.31	1.43	1.51	1.55

Tabla 17: Aceleración algoritmos en la tercera semana

Altura (metros)	Aceleración (segundos)				
	Secuencial	2 núcleos	4 núcleos	6 núcleos	8 núcleos
Cámara (1.5)	1.00	1.66	1.69	1.84	1.86
5	1.00	1.47	1.57	1.73	1.86
10	1.00	1.64	1.73	1.87	1.94
15	1.00	1.40	1.69	1.74	1.89

Tabla 18: Aceleración algoritmos en la cuarta semana

Altura (metros)	Aceleración (segundos)				
	Secuencial	2 núcleos	4 núcleos	6 núcleos	8 núcleos
Cámara (1.5)	1.00	1.19	1.32	1.37	1.41
5	1.00	1.34	1.47	1.72	1.90
10	1.00	1.42	1.54	1.67	1.66
15	1.00	1.48	1.69	2.10	2.07

En la cuarta semana con imágenes a 15 metros de altura se obtiene el doble de aceleración del algoritmo paralelo con 8 núcleos con respecto al algoritmo secuencial.

Las gráficas de comportamiento de la aceleración (segundos) del algoritmo paralelo se presentan en la Figura 68 (segunda semana), Figura 69 (tercera semana) y Figura 70 (Cuarta semana).

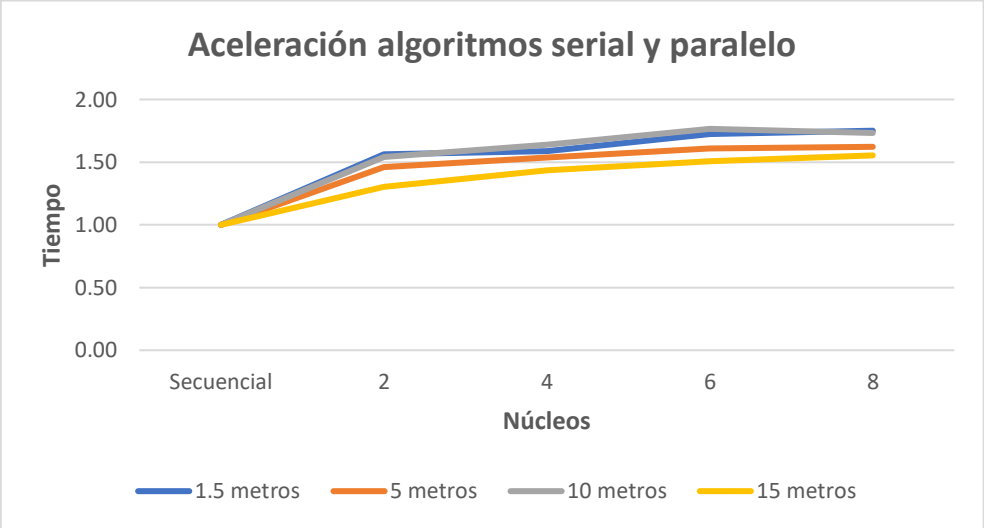


Figura 68: Aceleración algoritmo paralelo de la segunda semana

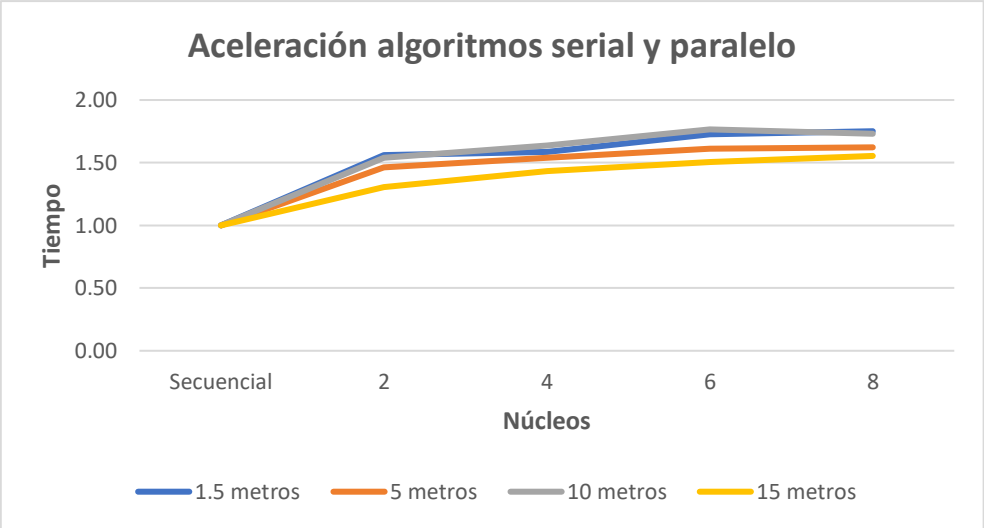


Figura 69: Aceleración algoritmo paralelo de la tercera semana

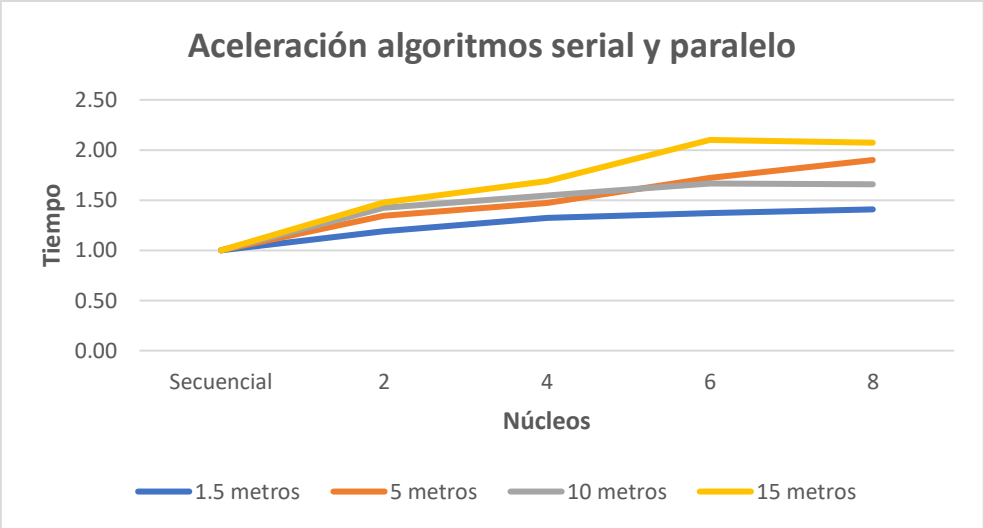


Figura 70: Aceleración algoritmo paralelo de la cuarta semana

El procesamiento de las imágenes adquiridas en las tres semanas de seguimiento mediante el algoritmo paralelo como mínimo 1.5 y máximo 2.1 veces más rápido comparado con el algoritmo secuencial para las tres alturas experimentadas, si bien es cierto el resultado no es tan notorio si se relaciona con el número de núcleos utilizados, el algoritmo mediante paralelismo es el doble de rápido que el algoritmo secuencial para procesar imágenes de la cuarta semana a 15 metros de altura.

4.2.3 Eficiencia

La Eficiencia se identifica como la porción de tiempo que los elementos de proceso se dedican a trabajo útil, el valor oscila entre 0 y 1. Los valores de eficiencia de los algoritmos obtenidos de las 3 semanas de seguimiento aplicando la ecuación 4 de la sección 2.4.4 se presentan en la Tabla 19 (segunda semana), Tabla 20 (tercera semana) y Tabla 21 (cuarta semana).

Tabla 19: Eficiencia algoritmos paralelos segunda semana

Altura (metros)	Eficiencia (segundos)				
	Secuencial	**2 núcleos**	**4 núcleos**	**6 núcleos**	**8 núcleos**
Cámara (1.5)	0.00	0.78	0.40	0.29	0.22
5	0.00	0.73	0.38	0.27	0.20
10	0.00	0.77	0.41	0.29	0.22
15	0.00	0.65	0.36	0.25	0.19

Tabla 20: Eficiencia algoritmos paralelos tercera semana

Altura (metros)	Eficiencia (segundos)				
	Secuencial	**2 núcleos**	**4 núcleos**	**6 núcleos**	**8 núcleos**
Cámara (1.5)	0.00	0.83	0.42	0.31	0.23
5	0.00	0.73	0.39	0.29	0.23
10	0.00	0.82	0.43	0.31	0.24
15	0.00	0.70	0.42	0.29	0.24

Tabla 21: Eficiencia algoritmos paralelos cuarta semana

Altura (metros)	Eficiencia (segundos)				
	Secuencial	**2 núcleos**	**4 núcleos**	**6 núcleos**	**8 núcleos**
Cámara (1.5)	0.00	0.59	0.33	0.23	0.18
5	0.00	0.67	0.37	0.29	0.24
10	0.00	0.71	0.39	0.28	0.21
15	0.00	0.74	0.42	0.35	0.26

Las gráficas de eficiencia (segundos) del algoritmo paralelo se presentan en la Figura 71 (segunda semana), Figura 72 (tercera semana) y Figura 73 (Cuarta semana). En el procesamiento de las imágenes obtenidas con cámara como imágenes obtenidas con dron, la tendencia de la eficiencia disminuye a medida que se incrementa el número de núcleos.

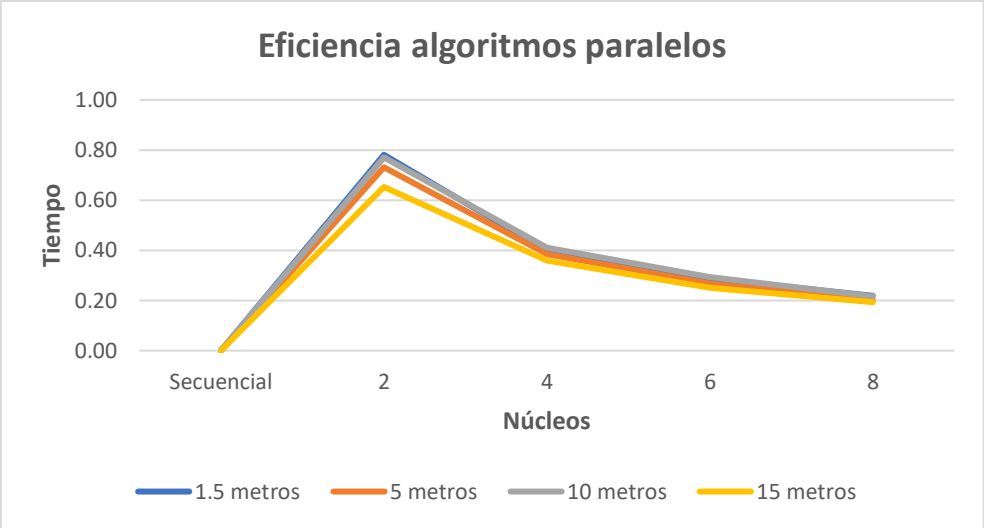


Figura 71: Eficiencia algoritmo segunda semana

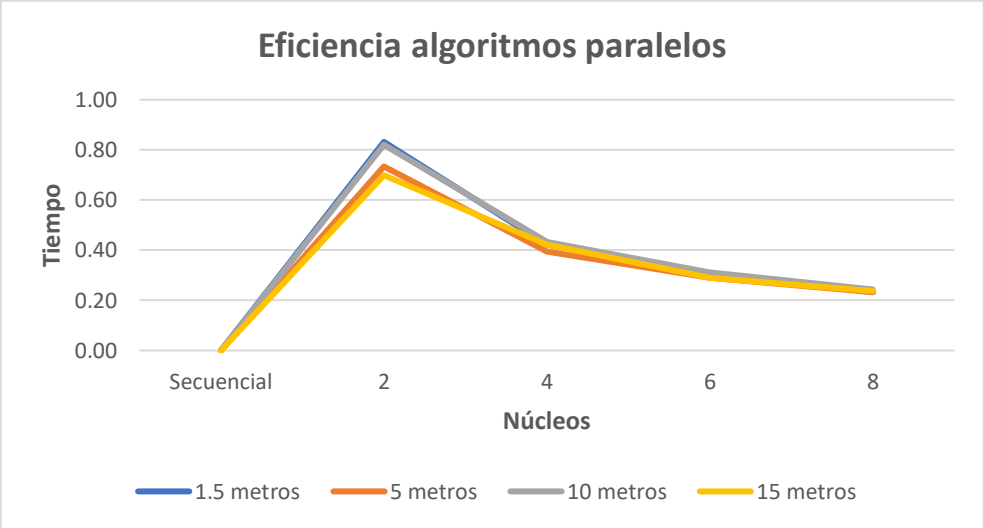


Figura 72: Eficiencia algoritmo tercera semana

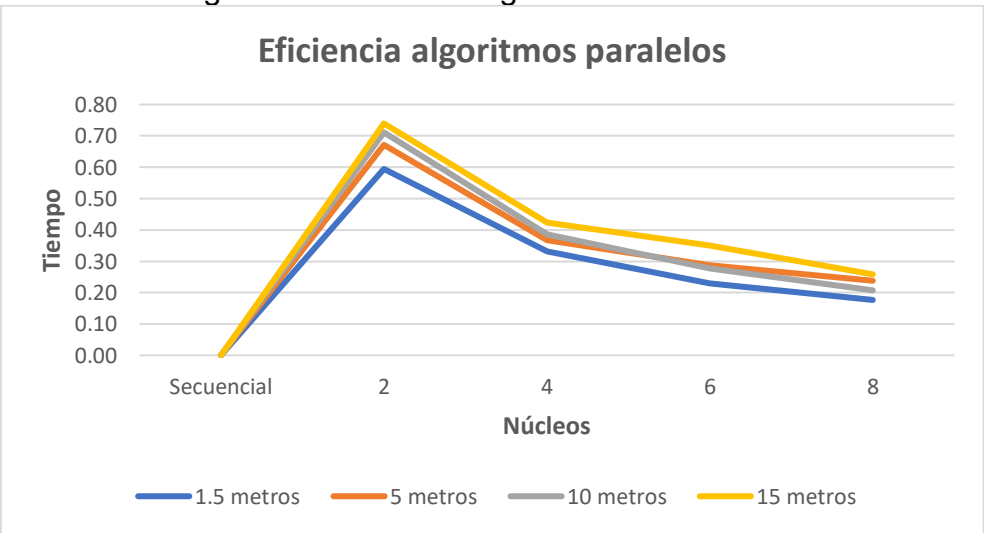


Figura 73: Eficiencia algoritmo cuarta semana

Considerando que el valor óptimo de eficiencia es (1), el algoritmo paralelo se desempeña de mejor manera utilizando 2 núcleos alcanzando una eficiencia de 0.82 aproximadamente (segunda semana) con imágenes obtenidas a 10 metros de altura (Tabla 20). Con mayor cantidad de núcleos se reduce la eficiencia; la razón puede darse por el tipo de paralelismo utilizado en el algoritmo o por la sobrecarga de tareas del procesador y la gestión de recursos de hardware. A pesar de los datos obtenidos el valor de eficiencia es mejor al algoritmo secuencial con imágenes adquiridas a 15 metros de altura en la cuarta semana de seguimiento al cultivo (Tabla 21).

4.2.4 Costo computacional

El costo computacional que representa el total de operaciones para identificar líneas de cultivo y malezas realizadas por los algoritmos paralelos para procesar imágenes obtenidas con cámara e imágenes obtenidas con dron durante las 3 semanas de seguimiento aplicando la ecuación definida en la sección 2.4.5 se presentan en la Tabla 22 (segunda semana), Tabla 23 (tercera semana) y Tabla 24 (cuarta semana).

Tabla 22: Costo computacional algoritmos paralelos segunda semana

Altura (metros)	**Costo computacional (operaciones)**				
	Secuencial	**2 núcleos**	**4 núcleos**	**6 núcleos**	**8 núcleos**
Cámara (1.5)	1.00	6.30	12.40	17.10	22.48
5	1.00	8.52	16.20	23.22	30.72
10	1.00	10.36	19.48	27.12	36.88
15	1.00	13.50	24.56	35.10	45.36

Tabla 23: Costo computacional algoritmos paralelos tercera semana

Altura (metros)	**Costo computacional (operaciones)**				
	Secuencial	**2 núcleos**	**4 núcleos**	**6 núcleos**	**8 núcleos**
Cámara (1.5)	1.00	6.30	12.40	17.10	22.48
5	1.00	9.24	17.24	23.46	29.12
10	1.00	10.52	19.96	27.66	35.52
15	1.00	13.08	21.68	31.44	38.64

Tabla 24: Costo computacional algoritmos paralelos cuarta semana

Altura (metros)	Costo computacional (operaciones)				
	Secuencial	2 núcleos	4 núcleos	6 núcleos	8 núcleos
Cámara (1.5)	1.00	10.04	18.04	26.10	33.92
5	1.00	10.50	19.16	24.60	29.68
10	1.00	11.36	20.92	29.10	38.96
15	1.00	12.72	22.24	26.88	36.32

Las gráficas del costo computacional de los algoritmos paralelos se presentan en las Figura 74 (segunda semana), Figura 75 (tercera semana) y Figura 76 (Cuarta semana). Los valores del costo computacional obtenido de los algoritmos paralelos, tanto para procesar imágenes obtenidas con cámara como imágenes obtenidas con dron, aumentan a medida que incrementa el número de núcleos.

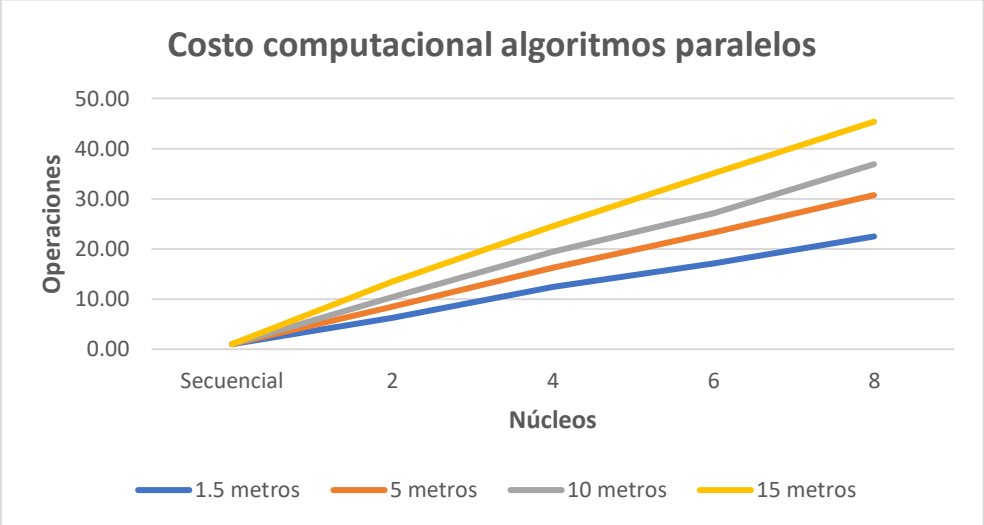


Figura 74: Costo computacional segunda semana

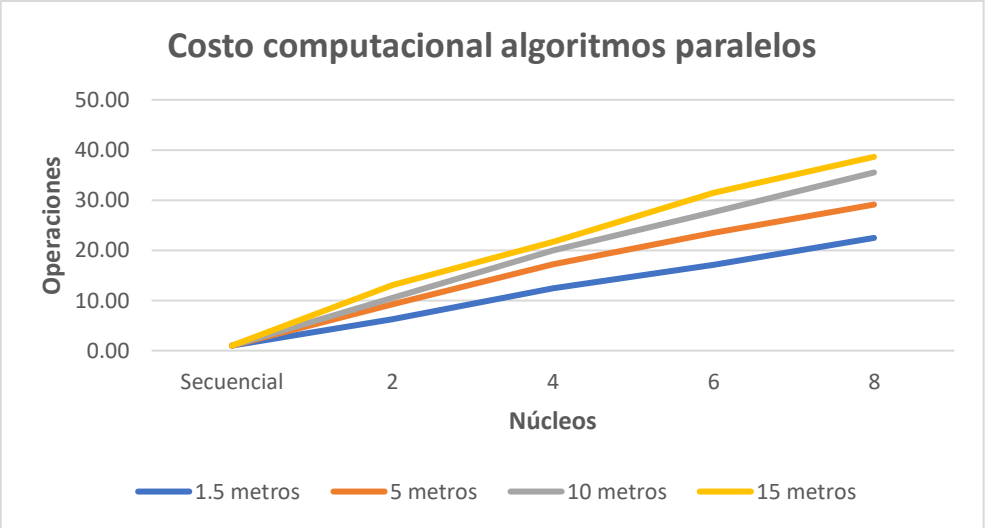


Figura 75: Costo computacional tercera semana

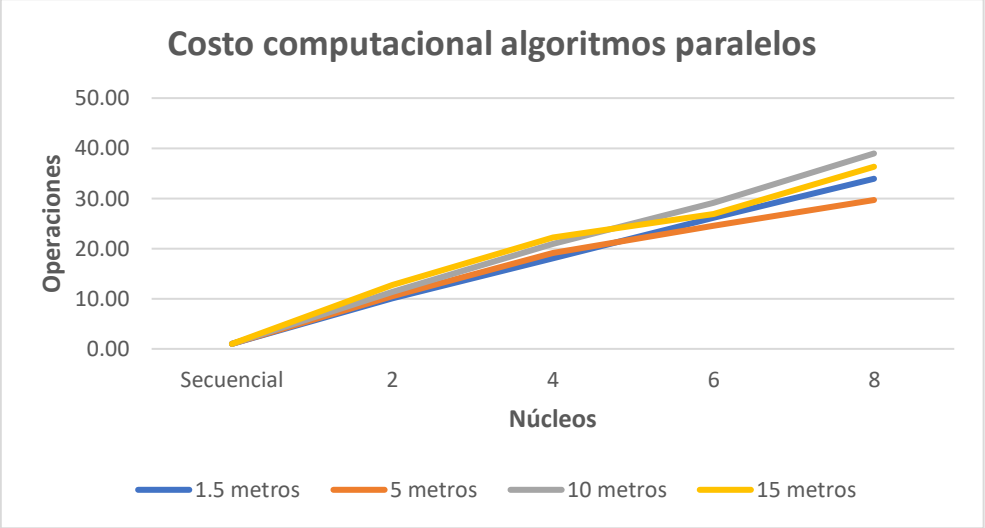


Figura 76: Costo computacional cuarta semana

Los resultados obtenidos sobre en costo computacional para procesar imágenes obtenidas a 15 metros de altura es superior al resto de imágenes de 5 o 10 metros. Tomando en cuenta las imágenes adquiridas a 15 metros de altura las imágenes tienen mayor cantidad de píxeles representadas por líneas de cultivo y malezas, al utilizar mayor cantidad de núcleos el número de operaciones también incrementa.

4.3 Pruebas estadísticas

Con la finalidad de evaluar la validez de la propuesta utilizando programación paralela, se utilizó la técnica estadística deductiva T-Student (Amat, 2016), en el lenguaje de programación R (Conesa, 2018), para determinar si existe una diferencia significativa entre las medidas de dos tipos de algoritmos (Serie, Paralelo). La prueba T-Student a ejecutarse es una prueba paramétrica por lo que antes de su ejecución se verificó que la muestra cumpla con los supuestos de linealidad, normalidad, homogeneidad y homocedasticidad. Para esto, se ejecutó un análisis de falsa regresión, en el cual se obtuvieron los cuantiles estandarizados para cada elemento de la muestra, los cuales fueron comparados con los cuantiles teóricos de la distribución χ^2.

Para detectar y eliminar posibles observaciones atípicas (Conesa, 2018), con base en el puntaje Z obtenido para cada elemento de la base de datos empleando un estadístico de 3, donde se obtuvo un puntaje de corte de 1.069. De esta manera, se concluyó que, no se detectaron observaciones atípicas y la base de datos quedó constituida por las mismas 300 observaciones, para confirmar la distribución de los datos evaluados se utilizó un diagrama de caja (boxplot) como se describe en la Figura 77, en la cual se muestra de cuartiles para las configuraciones en serie y paralelo.

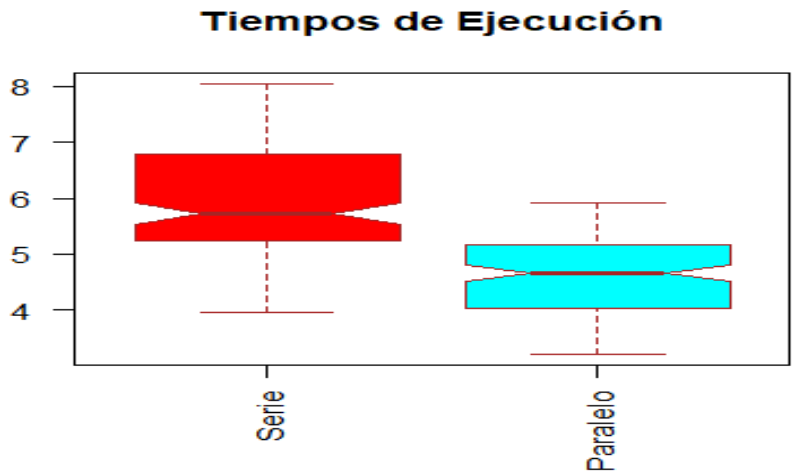


Figura 77: Distribución de cuartiles para configuraciones Serie y Paralelo

La Figura 78 muestra los resultados de la prueba de linealidad aplicando el análisis de regresión falsa con los tiempos de ejecución obtenidos para los dos

tipos de algoritmos. El gráfico Q-Q Plot donde se observa que los cuantiles comprendidos en un intervalo de -2 a 2 para el estadístico, se orientan en su gran mayoría en una tendencia lineal con una fuerte correlación positiva, por lo que se aceptó el supuesto de linealidad.

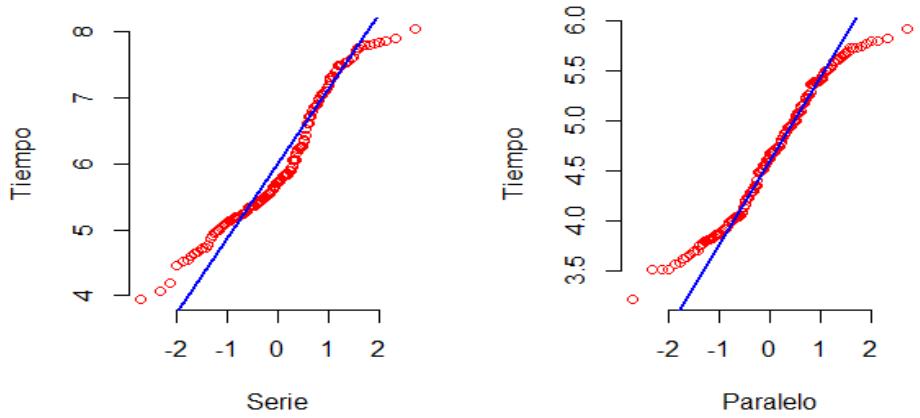


Figura 78: Prueba de linealidad configuraciones serie y paralelo

En la Figura 79, se puede apreciar el histograma de los cuantiles estandarizados, donde se visualiza una distribución centrada en cero y muy similar a una distribución normal, por lo que también se aceptó el supuesto de normalidad.

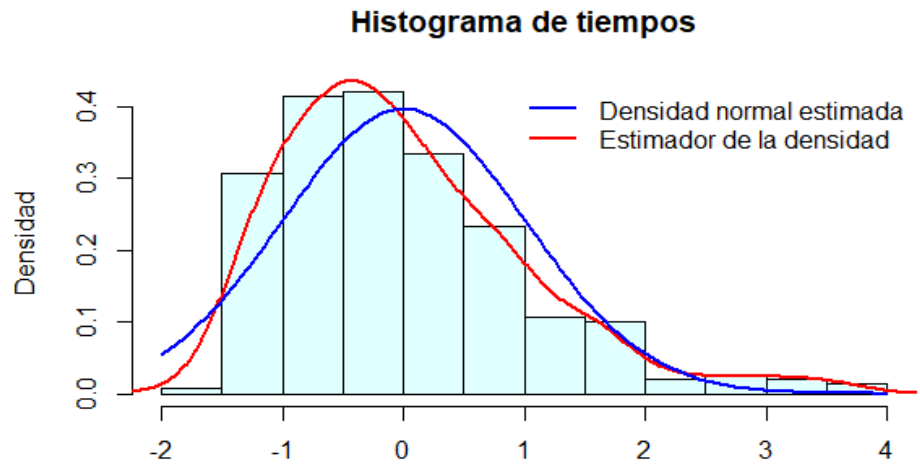


Figura 79: Prueba de normalidad de los tiempos estandarizados de los algoritmos en serie y en paralelo

Finalmente, en la Figura 69, se puede apreciar el diagrama de dispersión (scatter plot) de los cuantiles ajustados respecto a los estandarizados, donde se visualiza una distribución casi homogénea de los cuantiles en los cuatro cuadrantes y no existen patrones o tendencias en la distribución de los puntos, por lo que se aceptaron los supuestos de homogeneidad y homocedasticidad.

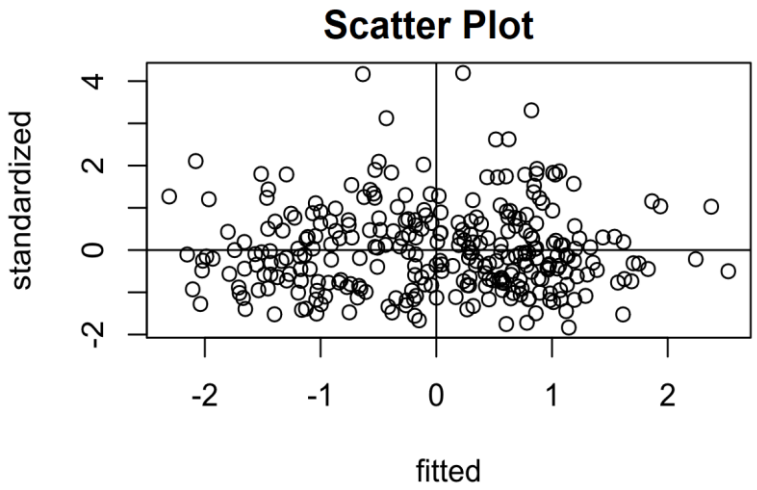


Figura 80: Prueba de homogeneidad y homocedasticidad con tiempos estandarizados de las configuraciones serie y paralelo

4.4 Análisis de Resultados

El algoritmo propuesto muestra una mejora en el tiempo de ejecución (51.75%) con relación al algoritmo secuencial con imágenes de la cuarta semana de seguimiento de los cultivos de maíz adquiridas con dron a 15 metros de altura cubriendo una extensión de 160 metros cuadrados aproximadamente. El algoritmo propuesto para procesar imágenes adquiridas con drone mediante paralelismo permiten la detección de un promedio de 18 líneas de cultivo de maíz que equivale a un 90% del total existentes (20). Con respecto a las malezas el algoritmo paralelizado en la cuarta semana procesando imágenes obtenidas a 15 metros de altura con drone detecta un 93.28% del total de malezas presente en el área de terreno (vegetación excluida las plantas de cultivo). El tiempo promedio de procesamiento con 8 núcleos (4.54 segundos) es moderado (Tabla 12) considerando el área de terreno cubierta en la imagen agrícola.

En base a los resultados obtenidos en el estudio, se evidenció que existe un mejor desempeño del algoritmo con imágenes adquiridas con un drone a 15 metros de altura en la cuarta semana de crecimiento, para detectar la mayor cantidad de líneas de cultivo (18); sin embargo, se conoce que existe otros trabajos realizados por (Costa et al., 2020; J. Zhang et al., 2018b; Zheng et al., 2020) que utilizan cámaras multiespectrales arrojan mejores resultados al usar multibandas para clasificar vegetación. A diferencia del trabajo (García-Santillán & Pajares, 2018) realizado sobre detección de malezas en cultivos de maíz con imágenes tomadas con un ángulo en perspectiva (15 grados) utilizando una cámara instalada en un tractor, cubren una área de 5 metros cuadrados con 4 líneas de cultivo, se reconoce un 87.9% de malezas en líneas de cultivo curvas y 90.7 en líneas de cultivo rectas; mientras que en el presente trabajo se reconoce el 90% (18 de líneas de cultivo), utilizando imágenes en vista cenital adquiridas a una altura de 15 metros.

La metodología propuesta se puede adaptar en cualquier cultivo considerando los parámetros intrínsecos (geometría y óptica) y extrínsecos (posición y orientación) de la cámara del dron, distancia entre líneas de cultivo y altura de las plantas; el algoritmo puede ser adaptado a cultivos similares o nuevos cultivos de la zona o diferentes países que requieran mayor preprocesamiento.

Es importante realizar un seguimiento del cultivo durante las 4 semanas para monitorear el crecimiento de las malezas en las primeras semanas de crecimiento del cultivo para extraer las malezas a tiempo y evitar afectaciones de nutrientes, luz solar y otras factores de crecimiento necesarios para las plantas de maíz.

Conclusiones

En la revisión de literatura, la mayoría de los artículos y revisiones analizados se centran en el procesamiento y análisis de imágenes utilizando técnicas (segmentación, índices de vegetación, aprendizaje, wavelets) para sistematizar tareas agrícolas de diferentes cultivos como maíz, café, lechuga, granadilla, arroz, papa.

El procesamiento de imágenes digitales es importante en la automatización de tareas agrícolas complejas, reduciendo la demanda de tiempos, recursos económicos, erosión del suelo y la contaminación ambiental. Los procesos agrícolas sistematizados mediante procesamiento de imágenes digitales están enfocados en la detección de plantas, plagas, insectos, enfermedades, calidad de suelo y monitoreo de cultivos, por lo que reducen el trabajo físico y manual, mejorando la productividad de los cultivos y la calidad de vida de los productores.

La utilización de arquitecturas heterogéneas, técnicas avanzadas de inteligencia artificial y lenguajes de programación de alto rendimiento para procesamiento paralelo son alternativas prometedoras para el procesamiento de imágenes de alta resolución y grandes tamaños, lo que conduce a la reducción de tiempo y costos de procesamiento computacional sirviendo de base para aplicar en otras disciplinas.

La herramienta Parallel Computing ToolBox incluida en Matlab permite la implementación de algoritmos secuenciales (serial) y paralelos para la detección de líneas de cultivo en imágenes de cultivos de maíz con la finalidad de reducir tiempos de procesamiento, aprovechando recursos de hardware disponibles en los computadores actuales. Con la utilización de paralelismo de datos es posible mejorar el rendimiento de los algoritmos, obteniendo menores tiempos de ejecución hasta un 51.75% en promedio para procesar imágenes agrícolas obtenidas a 15 metros de altura de la cuarta semana de seguimiento a los cultivos.

Estadísticamente la utilización de la distribución T-Student, confirma que la implementación de algoritmos paralelos en lenguajes de programación de alto rendimiento es viable porque los tiempos evaluados de los algoritmos

paralelizados son inferiores a la media de los tiempos en serie de una muestra de 300 mediciones para las diferentes alturas experimentadas.

Un aspecto importante a tener en cuenta en el diseño de algoritmos paralelos es la disponibilidad de varios núcleos físicos en los computadores actuales, posibilitando la disminución en los tiempos de ejecución mediante instrucciones de paralelismo en los segmentos de código que utilizan ciclos para realizar operaciones con los píxeles de las imágenes digitales; debido a que las herramientas de Matlab (parfor) distribuyen el procesamiento entre los núcleos físicos disponibles y en cada núcleo se distribuye las tareas utilizando los hilos disponibles; por lo tanto los algoritmos se adaptan de manera automática a cada computador.

En el procesamiento de imágenes adquiridas con el dron durante la primera semana fue difícil la detección de líneas de cultivo y malezas en las 3 alturas experimentadas debido a poca vegetación y tamaño mínimo de las plantas de maíz. A diferencia que en la cuarta semana las condiciones mejoran sustancialmente logrando la detección máxima de líneas de cultivo y un alto porcentaje de malezas.

El procesamiento de imágenes obtenidas con cámara fotográfica go pro mediante algoritmos paralelos posibilitan la detección de líneas de cultivo y malezas en cultivos de maíz en una cobertura limitada de máximo 6 metros cuadrados detectando hasta el 100% de líneas de cultivo y discriminando las malezas en un 84.71% con imágenes de la cuarta semana. El procesamiento paralelo con imágenes adquiridas mediante un dron a 15 metros de altura, posibilitan la detección máxima de líneas de cultivo (18) y un alto porcentaje de malezas (93.28%) del total de vegetación presente en las imágenes de los cultivos de maíz con una cobertura de hasta 160 metros cuadrados (Tabla 12), logrando disminuir tiempos de respuesta para extensiones amplias de cultivos con poco recorrido de vehículos no tripulados (drones). En el mejor de los casos (5 metros de altura) se alcanza a detectar hasta el 100% de líneas de cultivo discriminando las malezas en un 95.84 % con imágenes que cubren una extensión de .

Los resultados evaluados durante la cuarta semana con imágenes tomadas a 15 metros de altura con el dron, muestran una reducción de tiempo de ejecución aproximado del 51% comparado con el tiempo del algoritmo secuencial procesando las mismas imágenes con 8 núcleos, considerando que la vegetación es mayor en esta semana comparada con las anteriores.

Trabajos futuros

La utilización de arquitecturas heterogéneas y técnicas avanzadas de inteligencia artificial son alternativas prometedoras para el trabajo futuro en aplicaciones de procesamiento de imágenes en arquitecturas heterogéneas para aumentar la capacidad de procesamiento de imágenes con alta resolución y grandes tamaños, lo que permitirá reducción de tiempo, complejidad y costes computacionales en diferentes disciplinas aprovechando las características de los lenguajes de programación de alto rendimiento para procesamiento paralelo.

Discriminar las malezas mediante procesamiento paralelo utilizando arquitecturas heterogéneas y técnicas de inteligencia artificial en lenguajes de programación compatible.

Implementar algoritmos que faciliten la interacción con arquitecturas heterogéneas y otros equipos tecnológicos en tiempo real, de tal forma que sea fácilmente la adaptación a cultivos de maíz no sólo en las 4 semanas de crecimiento.

Desarrollar algoritmos para sistematizar otras actividades agrícolas utilizando imágenes digitales con más de 3 canales y otras técnicas de procesamiento aplicables a cultivos de maíz u otros de la Zona 1 – Ecuador.

Implementar algoritmos con paralelismo para detección de líneas de cultivo y malezas en tiempo real considerando nuevas infraestructuras remotas como cloud computing considerando las características de almacenamiento y comunicación.

Referencias

Abdul, H., Joseph, A., Ajay, G., & Mufeedha, K. (2016). Precision Farming: The Future of Indian Agriculture. *Journal of Applied Biology and Biotechnology*. https://doi.org/10.7324/JABB.2016.40609

Agricultura de Precisión. (2022). *¿Qué es la Agricultura de Precisión y cómo iniciarse en ella? - Agricultura de Precision para el Desarrollo*. http://agriculturadeprecisionparaeldesarrollo.com/que-es-la-agricultura-de-precision-y-como-iniciarse-en-ella/

Alonso, J. M., & Lvarruiz, F. (2018). *Ejercicios de programación paralela con openmp y mpi*. 10. https://www.mdconsult.internacional.edu.ec:2095

Álvarez Pastor, M. L. (2017). *Configuración y ejecución de algoritmos de visión artificial en la tarjeta Nvidia Jetson TK1 DevKit* [Universidad de Alcalá Escuela Politécnica Superior]. https://dspace.uah.es/dspace/bitstream/handle/10017/30204/TFG-Álvarez-Pastor-2017.pdf?sequence=1&isAllowed=y

Amat, J. (2016). *T-test: Comparación de medias poblaciones*. RPubs. https://rpubs.com/Joaquin_AR/218467

Andrade, F., Taboada, M., Lema, D., Maceira, N., Echeverría, H., Posse, G., Prieto, D., Sánchez, E., Ducasse, D., Bogliani, M., Gamundi, J. C., Trumper, E., Frana, J., Perotti, E., Fava, F., & Mastrángelo, M. (2017). *Los desafíos de la agricultura argentina* (Ediciones INTA (Ed.); Primera).

Anthony Simon, N., & Min, C. H. (2020). Neural network based corn field furrow detection for autonomous navigation in agriculture vehicles. *IEMTRONICS 2020 - International IOT, Electronics and Mechatronics Conference, Proceedings*. https://doi.org/10.1109/IEMTRONICS51293.2020.9216347

Arora, S., & Boaz, B. (2009). Computational Complexity: A Modern Approach Appendix A: Mathematical Background. *Cambrige University Press*. http://www.technoeng.ro/files/publications/2_0_book1.pdf

Atanassov, E., Barth, M., Byckling Mikko, & Bodgan Valeriu. (2017). *(PDF) Best Practice Guide – Intel® Xeon Phi™ v2.0*. https://www.researchgate.net/publication/314117889_Best_Practice_Guide_-_IntelR_Xeon_Phi_v20

Azzini, I., Muresano, R., & Ratto, M. (2018). Dragonfly: A multi-platform parallel toolbox for MATLAB/Octave. *Computer Languages, Systems and Structures*, *52*, 21–42. https://doi.org/10.1016/j.cl.2017.10.002

Bah, M. D., Hafiane, A., & Canals, R. (2020). CRowNet: Deep Network for Crop Row Detection in UAV Images. *IEEE Access*, *8*, 5189–5200. https://doi.org/10.1109/ACCESS.2019.2960873

Barbedo, J. G. A. (2016). A review on the main challenges in automatic plant disease identification based on visible range images. *Biosystems Engineering*, *144*, 52–60. https://doi.org/10.1016/j.biosystemseng.2016.01.017

Barlas, G. (2015a). Chapter 1 - Introduction. In G. Barlas (Ed.), *Multicore and GPU Programming* (pp. 1–26). Morgan Kaufmann. https://doi.org/https://doi.org/10.1016/B978-0-12-417137-4.00001-0

Barlas, G. (2015b). Chapter 2 - Multicore and parallel program design. In G. Barlas (Ed.), *Multicore and GPU Programming* (pp. 27–54). Morgan Kaufmann. https://doi.org/https://doi.org/10.1016/B978-0-12-417137-4.00002-2

Bombieri, N., Vinco, S., Bertacco, V., & Chatterjee, D. (2012). *SystemC Simulation on GP-GPUs: CUDA vs. OpenCL **. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.280.4218&rep=rep1&type=pdf

Burgos-Artizzu, X. P., Ribeiro, A., Guijarro, M., & Pajares, G. (2011). Real-time image processing for crop/weed discrimination in maize fields. *Computers and Electronics in Agriculture*, *75*(2), 337–346. https://doi.org/10.1016/j.compag.2010.12.011

Campos, Y., Sossa, H., & Pajares, G. (2016). Spatio-temporal analysis for obstacle detection in agricultural videos. *Applied Soft Computing Journal*, *45*, 86–97. https://doi.org/10.1016/j.asoc.2016.03.016

Chillet, D., & Hübner, M. (2014). Special issue on design and architectures of real-time image processing in embedded systems. *Journal of Real-Time Image Processing*, *9*(1), 1–3. https://doi.org/10.1007/s11554-014-0401-6

Chung, K.-L., Huang, Y.-H., & Tsai, S.-R. (2014). Orientation-based discrete Hough transform for line detection with low computational complexity. *Applied Mathematics and Computation*, *237*, 430–437. https://doi.org/https://doi.org/10.1016/j.amc.2014.03.128

Cisternas, I., Velásquez, I., Caro, A., & Rodríguez, A. (2020). Systematic literature review of implementations of precision agriculture. *Computers and Electronics in Agriculture*, *176*(May), 105626. https://doi.org/10.1016/j.compag.2020.105626

Conesa, D. (2018). *Computación y programación en R : Tema 4 Tema 4 : Análisis de datos con R*. https://www.uv.es/conesa/CursoR/material/handout-sesion4.pdf

Costa, L., Nunes, L., & Ampatzidis, Y. (2020). A new visible band index (vNDVI) for estimating NDVI values on RGB images utilizing genetic algorithms. *Computers and Electronics in Agriculture*, *172*(November 2019), 105334. https://doi.org/10.1016/j.compag.2020.105334

Czech, Z. J. (2017). Introduction to Parallel Computing. In *Introduction to Parallel Computing*. Cambridge University Press. https://doi.org/10.1017/9781316795835

Davies, E. R. (2009). The application of machine vision to food and agriculture: a review. *The Imaging Science Journal*, *57*(4), 197–217. https://doi.org/10.1179/174313109X454756

Dávila-Guzmán, M. A., Alfonso-Morales, W., & Caicedo-Bravo, E. F. (2014). Arquitectura heterogénea para el procesamiento de los algoritmos de

enjambres. *TecnoLógicas*, *17*(32), 11. https://doi.org/10.22430/22565337.197

De Giusti, A., Naiouf, M., Fernando, T., & Horacio, V. (2020). Arquitecturas multiprocesador : Hardware , Software , Modelos , Métricas y Tendencias Resumen Contexto Introducción. *WICC 2020*, *1*, 769–775.

De Giusti, A., Tinetti, F., Naiouf, M., Chichizola, F., De Giusti, L., Villagarcía, H., Montezanti, D., Encinas, D., Pousa, A., Rodriguez, I., Rodriguez, E. S., Iglesias, L., Paniego, J. M., Puig, M., Oso, M., & Mendez, M. (2017). *Arquitecturas Multiprocesador en Computación de Alto Desempeño: Software, Métricas, Modelos y Aplicaciones*. http://sedici.unlp.edu.ar/bitstream/handle/10915/62558/Documento_completo.pdf-PDFA.pdf?sequence=1

Dean, W. (2010). *Computational Complexity Theory*.

Díaz, G. (2018). *Diseño de Algoritmos Paralelos*. http://webdelprofesor.ula.ve/ingenieria/gilberto/paralela/08_DisenoDeAlgoritmosParalelos.pdf

DJI. (2020). *Mavic 2 - DJI*. DJI. https://www.dji.com/mavic-2

Dufrechou, E. (2018). *Accelerating advanced preconditioning methods on hybrid architectures*. Universidad de la República.

Fabiani, S., Vanino, S., Napoli, R., Zajíček, A., Duffková, R., Evangelou, E., & Nino, P. (2020). Assessment of the economic and environmental sustainability of Variable Rate Technology (VRT) application in different wheat intensive European agricultural areas. A Water energy food nexus approach. *Environmental Science and Policy*, *114*(September), 366–376. https://doi.org/10.1016/j.envsci.2020.08.019

Fang, J., Huang, C., Tang, T., & Wang, Z. (2020). Parallel programming models for heterogeneous many-cores: a comprehensive survey. *CCF Transactions on High Performance Computing*, *2*(4), 382–400. https://doi.org/10.1007/s42514-020-00039-4

Feng, X., Jiang, Y., Yang, X., Du, M., & Li, X. (2019). Computer vision algorithms and hardware implementations: A survey. *Integration*, *69*(June), 309–320. https://doi.org/10.1016/j.vlsi.2019.07.005

Fernández, F. J. (2012). Computación científica paralela mediante uso de herramientas para paso de mensajes. *Inge-Cuc*, *8*(1), 51–84.

García-Santillán, I., & Caranqui, V. (2015). La visión artificial y los campos de aplicación. *Tierra Infinita*, *1*, 94–103.

García-Santillán, I., & Caranqui, V. (2014). La visión artificial y los campos de aplicación. *Tierra Infinita*, *4*, 142–150.

García-Santillán, I., Guerrero, J., Montalvo, M., & Pajares, G. (2017). Curved and straight crop row detection by accumulation of green pixels from images in maize fields. *Precision Agriculture*, 1–24. https://doi.org/10.1007/s11119-016-9494-1

García-Santillán, I., Montalvo, M., Guerrero, J., & Pajares, G. (2017). Automatic detection of curved and straight crop rows from images in maize fields. *Biosystems Engineering*, *156*, 61–79. https://doi.org/10.1016/j.biosystemseng.2017.01.013

García-Santillán, I., & Pajares, G. (2016). Detección automática de líneas de cultivo: estado del arte y futuras perspectivas. *Avances y Aplicaciones de Sistemas Inteligentes y Nuevas Tecnologías*, *54*, 381–398.

García-Santillán, I., & Pajares, G. (2018). On-line crop/weed discrimination through the Mahalanobis distance from images in maize fields. *Biosystems Engineering*, *166*, 28–43. https://doi.org/10.1016/j.biosystemseng.2017.11.003

García-Santillán, I., Peluffo-Ordoñez, D., Caranqui, V., Pusdá-Chulde, M., Garrido, F., & Granda, P. (2018). Computer Vision-Based Method for Automatic Detection of Crop Rows in Potato Fields. *Advances in Intelligent Systems and Computing*, *Vol. 721*(ICITS 2018), 355–366. https://doi.org/10.1007/978-3-319-73450-7

García-Santillán, I., Peluffo-Ordoñez, D., Caranqui, V., Pusdá, M., Garrido, F., & Granda, P. (2018). Computer Vision-Based Method for Automatic Detection of Crop Rows in Potato Fields. *Advances in Intelligent Systems and Computing*, *Vol. 721*(ICITS 2018), 355–366. https://doi.org/10.1007/978-3-319-73450-7

García-Santillán, I., Pusdá-Chulde, M., & Pajares, G. (2017). Identificación automática de vegetación utilizando imágenes agrícolas: una revisión de métodos. In D. Imbaquingo (Ed.), *Tecnologías Aplicadas a la Ingeniería* (pp. 155–162). Universidad Técnica del Norte.

García-Santillán, I., Pusdá, M., & Pajares, G. (2017). Identificación automática de vegetación utilizando imágenes agrícolas: una revisión de métodos. In D. Imbaquingo (Ed.), *Tecnologías Aplicadas a la Ingeniería* (pp. 155–162). Universidad Técnica del Norte.

Garcia, D. C., Martínez, G. Á., & Garcia, M. E. (2018). Raspberry Pi y Arduino: semilleros en innovación tecnológica para la agricultura de precisión. *Informática y Sistemas: Revista de Tecnologías de La Informática y Las Comunicaciones*, *2*(1), 74–82. https://doi.org/10.33936/ISRTIC.V2I1.1134

García, E., & Flego, F. (2016). *Agricultura de Precisión*. https://www.palermo.edu/ingenieria/downloads/pdfwebc&T8/8CyT12.pdf

Gašparović, M., Zrinjski, M., Barković, Đ., & Radočaj, D. (2020). An automatic method for weed mapping in oat fields based on UAV imagery. *Computers and Electronics in Agriculture*, *173*(February 2019), 105385. https://doi.org/10.1016/j.compag.2020.105385

Gonzáles Itúrbide, J. J. (2016). *OPTIMIZACIÓN DE ALGORITMOS PARA PROCESADO DE IMÁGENES CON GPUs*. http://oa.upm.es/36950/1/TM_GONZALEZ_ITURBIDE_JOSEJAIME.pdf

Grama, A., Gupta, A., Karypis, G., Kumar, V., & Wesley, A. (2003). *Introduction to Parallel Computing, Second Edition* (Wesley Addison (Ed.)). Pearson Education Limited. http://srmcse.weebly.com/uploads/8/9/0/9/8909020/introduction_to_parallel

_computing_second_edition-ananth_grama..pdf

Guerrero, J. M., Pajares, G., Montalvo, M., Romeo, J., & Guijarro, M. (2012). Support Vector Machines for crop/weeds identification in maize fields. *Expert Systems with Applications*, *39*(12), 11149–11155. https://doi.org/10.1016/j.eswa.2012.03.040

Guijarro, M., Riomoros, I., Pajares, G., & Zitinski, P. (2015). Discrete wavelets transform for improving greenness image segmentation in agricultural images. *Computers and Electronics in Agriculture*, *118*, 396–407. https://doi.org/10.1016/j.compag.2015.09.011

Hager, G., & Wellein, G. (2010). Introduction to High Perfomance Computing for Scientists and Engineers. In *Computational Science Series*.

HajiRassouliha, A., Taberner, A. J., Nash, M. P., & Nielsen, P. M. F. (2018). Suitability of recent hardware accelerators (DSPs, FPGAs, and GPUs) for computer vision and image processing algorithms. *Signal Processing: Image Communication*, *68*(November 2017), 101–119. https://doi.org/10.1016/j.image.2018.07.007

HEMAV. (2022). *Teledetección con drones Vs Teledetección satelital*. https://hemav.com/agricultura-de-precision-teledeteccion-satelital-vs-teledeteccion-con-drones/

Herbert, H. (2015). Introducción a la Computación Paralela. *Centro Nacional de Cálculo Científico Universidad de Los Andes*. http://www.saber.ula.ve/bitstream/123456789/15969/1/com_par.pdf

Hough, P. V. C. (1962). A method and means for recognition complex patterns; US Patent: US3069654A. *US Patent*, 6.

INTEL. (2021). *Familia de productos Intel Xeon Phi*. Intel. https://www.intel.la/content/dam/www/public/lar/xl/es/documents/articles/327383-002us-xeonphi-sb-fin-lo-spa.pdf

ISPA. (2022). *Home | International Society of Precision Agriculture*.

https://www.ispag.org/

Jiménez-Brenes, F. M., López-Granados, F., Torres-Sánchez, J., Peña, J. M., Ramírez, P., Castillejo-González, I. L., & de Castro, A. I. (2019). Automatic UAV-based detection of Cynodon dactylon for site-specific vineyard management. *Plos One*, *14*(6), e0218132. https://doi.org/10.1371/journal.pone.0218132

Jiménez-Gonzalez, D. (2010). Introducción a Las arquitecturas paralelas. In *Programas de Estudios* (Vol. 1, Issue Semestre 1). https://www.exabyteinformatica.com/uoc/Informatica/Arquitecturas_de_computadores_avanzadas/Arquitecturas_de_computadores_avanzadas_(Modulo_2).pdf

Kakani, V., Nguyen, V. H., Kumar, B. P., Kim, H., & Pasupuleti, V. R. (2020). A critical review on computer vision and artificial intelligence in food industry. *Journal of Agriculture and Food Research*, *2*(February), 100033. https://doi.org/10.1016/j.jafr.2020.100033

Kamilaris, A., & Prenafeta-Boldú, F. X. (2018). Deep learning in agriculture: A survey. *Computers and Electronics in Agriculture*, *147*(February), 70–90. https://doi.org/10.1016/j.compag.2018.02.016

Korohou, T., Okinda, C., Li, H., Cao, Y., Nyalala, I., Huo, L., Potcho, M., Li, X., & Ding, Q. (2020). Wheat Grain Yield Estimation Based on Image Morphological Properties and Wheat Biomass. *Journal of Sensors*, *2020*, 1571936. https://doi.org/10.1155/2020/1571936

Leibovich, F., Chichizola, F., De Giusti, L., Naiouf, M., Tirado Fernández, F., & De Giusti, A. (2012). *Programación híbrida en clusters de multicore. Análisis del impacto de la jerarquía de memoria*. http://sedici.unlp.edu.ar/bitstream/handle/10915/23629/Documento_completo.pdf?sequence=1

Liao, S. W., Kuang, S. Y., Kao, C. L., & Tu, C. H. (2017). A Halide-based Synergistic Computing Framework for Heterogeneous Systems. *Journal of Signal Processing Systems*, 1–15. https://doi.org/10.1007/s11265-017-

1283-1

Libutti, L. A., Igual, F. D., Piñuel, L., De Giusti, L., & Naiouf, M. (2020). Benchmarking Performance and Power of USB Accelerators for Inference with MLPerf. *AccML*.

López-Granados, F., Torres-Sánchez, J., Jiménez-Brenes, F. M., Arquero, O., Lovera, M., & De Castro, A. I. (2019). An efficient RGB-UAV-based platform for field almond tree phenotyping: 3-D architecture and flowering traits. *Plant Methods*, *15*(1), 1–16. https://doi.org/10.1186/s13007-019-0547-0

Lu, Y., & Young, S. (2020). A survey of public datasets for computer vision tasks in precision agriculture. *Computers and Electronics in Agriculture*, *178*(May), 105760. https://doi.org/10.1016/j.compag.2020.105760

Marques Ramos, A. P., Prado Osco, L., Elis Garcia Furuya, D., Nunes Gonçalves, W., Cordeiro Santana, D., Pereira Ribeiro Teodoro, L., Antonio da Silva Junior, C., Fernando Capristo-Silva, G., Li, J., Henrique Rojo Baio, F., Marcato Junior, J., Eduardo Teodoro, P., & Pistori, H. (2020). A random forest ranking approach to predict yield in maize with uav-based vegetation spectral indices. *Computers and Electronics in Agriculture*, *178*(July), 105791. https://doi.org/10.1016/j.compag.2020.105791

MathWorks. (2020). *Parallel Computing Toolbox*. https://la.mathworks.com/products/parallel-computing.html

MathWorks. (2021). *MATLAB - El lenguaje del cálculo técnico - MATLAB & Simulink*. MATHWORKS. https://la.mathworks.com/products/matlab.html?s_tid=hp_products_matlab

MathWorks. (2022). *GPU Programming Paradigm - MATLAB & Simulink - MathWorks América Latina*. https://la.mathworks.com/help/gpucoder/gs/gpu-prog-paradigm.html

Milla, A., & Rucci, E. (2022). Performance Comparison of Python Translators for a Multi-threaded CPU-Bound Application. In P. Pesado & G. Gil (Eds.),

Computer Science -- CACIC 2021 (pp. 21–38). Springer International Publishing.

Mishra, D., & Natalizio, E. (2020). A survey on cellular-connected UAVs: Design challenges, enabling 5G/B5G innovations, and experimental advancements. *Computer Networks*, *182*(August). https://doi.org/10.1016/j.comnet.2020.107451

Mukherjee, A., Misra, S., & Raghuwanshi, N. S. (2019). A survey of unmanned aerial sensing solutions in precision agriculture. *Journal of Network and Computer Applications*, *148*, 102461. https://doi.org/10.1016/j.jnca.2019.102461

Murilo, B. (2014). *Modelos Paralelos para la Resolución de Problemas de Ingeniería Agrícola*. Universidad Politécnica de Valencia.

Naiouf, M., De Giusti, A., De Giusti, L., Chichizola, F., Sanz, V., Pousa, A., Leibovich, F., Rucci, E., Frati, E., Encinas, D., Gallo, S., Montes De Oca, E., & Balladini, J. (2012). Algoritmos Paralelos y Distribuidos. Fundamentos, Modelos y Aplicaciones. *WICC 2012*. http://sedici.unlp.edu.ar/bitstream/handle/10915/19392/Documento_completo. Fundamentos, Modelos y Aplicaciones.pdf?sequence=1

Naiouf, M., Giusti, A. De, Giusti, L. De, Chichizola, F., Sanz, V., Rucci, E., Gallo, S., Oca, E. M. De, Frati, E., Gaudiani, A., Plata, L., & Aires, B. (2020). Fundamentos, algoritmos y evaluacion de rendimiento en diferentes plataformas de hpc. *SEDICI*, *1*, 763–768.

Njoroge, B. M., Fei, T. K., & Thiruchelvam, V. (2018a). A research review of precision farming techniques and technology. *Journal of Applied Technology and Innovation*, *2*(1), 22–30. https://jati.sites.apiit.edu.my/files/2018/07/2018_Issue1_Paper4.pdf

Njoroge, B. M., Fei, T. K., & Thiruchelvam, V. (2018b). A research review of precision farming techniques and technology. *Journal of Applied Technology and Innovation*, *2*(1), 22–30.

Nvidia. (2019). *Procesamiento paralelo CUDA*. https://www.nvidia.com/es-la/drivers/what-is-gpu-computing/

NVIDIA. (2015). Cuda C Programming Guide. *Programming Guides*, *September*, 1–261. https://docs.nvidia.com/cuda/cuda-c-programming-guide/

NVIDIA. (2021). *Jetson Nano Developer Kit for AI and Robotics | NVIDIA*. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-nano/

OpenMP. (2022). *Home - OpenMP*. https://www.openmp.org/

Osio, J. R., Salvatore, J. E., Kunysz, E., Montezanti, D., Alonso, D., Guarepi, V., & Morales, D. M. (2017). *Análisis de Eficiencia en Arquitecturas Multiprocesador para Aplicaciones de Transmisión y Procesamiento de Datos*. http://sedici.unlp.edu.ar/bitstream/handle/10915/61570/Documento_completo.pdf-PDFA.pdf?sequence=1

Pardo, F. (2002). Arquitecturas Avanzadas. In *Apuntes de la Universidad de Valencia, España*. http://informatica.uv.es/guia/asignatu/2000/AAC/AA/apuntes/aa.pdf

Pascual, O. D. (2017). *Uso de Herramientas de Paralelizado para Optimización de Algoritmos de Procesado de Iimágenes*. Universidad Politécnica de Valencia.

Ponnusamy, V., & Natarajan, S. (2021). Precision Agriculture Using Advanced Technology of IoT, Unmanned Aerial Vehicle, Augmented Reality, and Machine Learning. *Internet of Things*, 207–229. https://doi.org/10.1007/978-3-030-52624-5_14

Printista, M., Gil-Costa, V., Alaniz, M., & Bustos, F. (2011). Optimización de Algoritmos utilizando Sistemas de Cómputo Híbridos. *XIII Workshop de Investigadores En Ciencias de La Computación*, 715–718. http://sedici.unlp.edu.ar/bitstream/handle/10915/19882/Documento_comple

to.pdf?sequence=1

Pusdá-Chulde, M., De Giusti, A., Herrera-Granda, E., & García-Santillán, I. (2021). Parallel CPU-Based Processing for Automatic Crop Row Detection in Corn Fields. In M. Botto-Tobar, H. Cruz, & A. Díaz Cadena (Eds.), *Artificial Intelligence, Computer and Software Engineering Advances* (pp. 239–251). Springer International Publishing.

Pusdá-Chulde, M., Robayo, A., Giusti, A. De, & García-Santillán, I. (2021). Detection of Crop Lines and Weeds in Corn Fields Based on Images Obtained from a Drone. *Communications in Computer and Information Science*, *1444 CCIS*, 31–45. https://doi.org/10.1007/978-3-030-84825-5_3

Pusdá-Chulde, M., Salazar-Fierro, F., Sandoval-Pillajo, L., Herrera-Granda, E., García-Santillán, I., & De Giusti, A. (2020). Image Analysis Based on Heterogeneous Architectures for Precision Agriculture: A Systematic Literature Review. In *Advances in Intelligent Systems and Computing* (Vol. 1078). https://doi.org/10.1007/978-3-030-33614-1_4

Python. (2022). *multiprocessing — Paralelismo basado en procesos — documentación de Python - 3.10.7*. https://docs.python.org/es/3/library/multiprocessing.html

Radoglou-Grammatikis, P., Sarigiannidis, P., Lagkas, T., & Moscholios, I. (2020a). A compilation of UAV applications for precision agriculture. *Computer Networks*, *172*(February), 107148. https://doi.org/10.1016/j.comnet.2020.107148

Radoglou-Grammatikis, P., Sarigiannidis, P., Lagkas, T., & Moscholios, I. (2020b). A compilation of UAV applications for precision agriculture. *Computer Networks*, *172*(February), 107148. https://doi.org/10.1016/j.comnet.2020.107148

Rahmat, R. F., Saputra, T., Hizriadi, A., Lini, T. Z., & Nasution, M. K. M. (2019). Performance Test of Parallel Image Processing Using Open MPI on Raspberry PI Cluster Board. *2019 3rd International Conference on Electrical, Telecommunication and Computer Engineering, ELTICOM 2019*

- *Proceedings*, 32–35. https://doi.org/10.1109/ELTICOM47379.2019.8943848

Ramírez Osuna, D. G. (2012). *Desarrollo de un método de procesamiento de imágenes para la discriminación de maleza en cultivos de maíz* [Universidad Autónoma de Querétaro]. http://ri.uaq.mx/xmlui/bitstream/handle/123456789/697/RI000112.pdf?sequence=1&isAllowed=y

Raspberry. (2018). *Raspberry Datashet*. https://datasheets.raspberrypi.com/pico/pico-datasheet.pdf

Ravi, V. T., Ma, W., Chiu, D., & Agrawal, G. (2012). Compiler and runtime support for enabling reduction computations on heterogeneous systems. *Concurrency and Computation: Practice and Experience*, *24*(5), 463–480. https://doi.org/10.1002/cpe.1848

RHEA. (2014). *Second International Conference on Robotics and associated High-technologies and Equipment for Agriculture and forestry (RHEA-2014)* (P. Gonzalez-de-santos & A. Ribeiro (Eds.); 1st ed.).

Rodríguez, F. (2010). *Programación Matlab En Paralelo Sobre Clúster Computacional : Evaluación De Prestaciones*. Universidad Politécnica de Cartagena.

Roldán-Serrato, K. L., Escalante-Estrada, J. A. S., & Rodríguez-González, M. T. (2018). Automatic pest detection on bean and potato crops by applying neural classifiers. *Engineering in Agriculture, Environment and Food*, *11*(4), 245–255. https://doi.org/10.1016/j.eaef.2018.08.003

Romeo, J., Pajares, G., Montalvo, M., Guerrero, J. M., Guijarro, M., & Ribeiro, A. (2012). Crop Row Detection in Maize Fields Inspired on the Human Visual Perception. *The Scientific World Journal*, *2012*, 1–10. https://doi.org/10.1100/2012/484390

Rovira-Más, F., Zhang, Q., Reid, J. F., & Will, J. D. (2005). Hough-transform-based vision algorithm for crop row detection of an automated agricultural

vehicle. *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, *219*(8), 999–1010. https://doi.org/10.1243/095440705X34667

Salza, P., & Ferrucci, F. (2019). Speed up genetic algorithms in the cloud using software containers. *Future Generation Computer Systems*, *92*, 276–289. https://doi.org/10.1016/j.future.2018.09.066

School of Science and Technology. (2016). *Unit 9 : Fundamentals of Parallel Processing Lesson 1 : Types of Parallel Processing 1.1. Learning Objectives*. Bangladesh Open University.

SENPLADES. (2017). *Buen vivir 2017-2021* (p. 150). http://www.planificacion.gob.ec/wp-content/uploads/downloads/2017/07/Plan-Nacional-para-el-Buen-Vivir-2017-2021.pdf

Shah, J. P., Prajapati, H. B., & Dabhi, V. K. (2016). A survey on detection and classification of rice plant diseases. *2016 IEEE International Conference on Current Trends in Advanced Computing, ICCTAC 2016*. https://doi.org/10.1109/ICCTAC.2016.7567333

Slabaugh, G., Boyes, R., & Yang, X. (2010). Multicore image processing with openMP. *IEEE Signal Processing Magazine*, *27*(2), 134–138. https://doi.org/10.1109/MSP.2009.935452

Stanford Encyclopedia of Philosophy. (2017). *Computational Complexity Theory*. https://plato.stanford.edu/entries/computational-complexity/

Sucar, L. E., & Gómez Giovani. (2015). *Visión Computacional* (G. Gómez (Ed.); 1st ed.). Instituto Nacional de Astrofíısica, Óptica y Electrónica. http://ccc.inaoep.mx/~esucar/Libros/vision-sucar-gomez.pdf

Suh, J. W., & Kim, Y. (2014). MATLAB and Parallel Computing Toolbox. *Accelerating Matlab with GPUs*, 99–125. https://doi.org/10.1016/b978-0-12-408080-5.00005-5

Tabares Soto, R. (2016). *Programación paralela sobre arquitecturas heterogéneas*. 80. http://www.bdigital.unal.edu.co/54267/

Tellaeche, A., Pajares, G., Burgos-Artizzu, X. P., & Ribeiro, A. (2011). A computer vision approach for weeds identification through Support Vector Machines. *Applied Soft Computing Journal*, *11*(1), 908–915. https://doi.org/10.1016/j.asoc.2010.01.011

Tian, H., Wang, T., Liu, Y., Qiao, X., & Li, Y. (2020). Computer vision technology in agricultural automation —A review. *Information Processing in Agriculture*, *7*(1), 1–19. https://doi.org/10.1016/j.inpa.2019.09.006

Torres-Sánchez, J., de Castro, A. I., Peña, J. M., Jiménez-Brenes, F. M., Arquero, O., Lovera, M., & López-Granados, F. (2018). Mapping the 3D structure of almond trees using UAV acquired photogrammetric point clouds and object-based image analysis. *Biosystems Engineering*, *176*, 172–184. https://doi.org/10.1016/j.biosystemseng.2018.10.018

Udal'tsov, A. V. (2018). Gas-phase water: Features of hydrogen bonding deduced from far-infrared VRT spectra proving the cluster formation in vacuum. *Chemical Physics*, *511*(June), 46–53. https://doi.org/10.1016/j.chemphys.2018.06.002

Universidad Europea de Madrid. (2017). *Introducción y Medidas de Rendimiento*. http://www.cartagena99.com/recursos/alumnos/apuntes/ININF1_M10_U1_T4_MT.pdf

Vasudevan, A., Kumar, D. A., & Bhuvaneswari, N. S. (2016). Precision farming using unmanned aerial and ground vehicles. *Proceedings - 2016 IEEE International Conference on Technological Innovations in ICT for Agriculture and Rural Development, TIAR 2016*, *Tiar*, 146–150. https://doi.org/10.1109/TIAR.2016.7801229

Vithu, P., & Moses, J. A. (2016). Machine vision system for food grain quality evaluation: A review. *Trends in Food Science and Technology*, *56*, 13–20. https://doi.org/10.1016/j.tifs.2016.07.011

Wang, J., Ma, X., Zhu, Y., & Sun, J. (2014). Efficient parallel implementation of active appearance model fitting algorithm on GPU. *TheScientificWorldJournal*, *2014*, 528080. https://doi.org/10.1155/2014/528080

Wang, L., Wang, Y., & Xie, Y. (2015). Implementation of a parallel algorithm based on a spark cloud computing platform. *Algorithms*, *8*(3), 407–414. https://doi.org/10.3390/a8030407

Weinstock, J. A. N. H., Murillo, L. G., & Leupers, R. (2016). Parallel SystemC Simulation for ESL Design. *ACM Transactions on Embedded Computing Systems*, *16*(1), 25. https://doi.org/10.1145/2987374

Weiss, S. (2014). Parallel Computing. In *Hunter College* (Department). Department of Computer Science. http://www.compsci.hunter.cuny.edu/~sweiss/course_materials/csci493.65/csci493.65_spr14.php

WikiWand. (2019). *Teoría de la complejidad computacional - Wikiwand*. https://www.wikiwand.com/es/Teoría_de_la_complejidad_computacional

Xie, C., & Yang, C. (2020). A review on plant high-throughput phenotyping traits using UAV-based sensors. *Computers and Electronics in Agriculture*, *178*(August), 105731. https://doi.org/10.1016/j.compag.2020.105731

Yang, C. T., Huang, C. L., & Lin, C. F. (2011). Hybrid CUDA, OpenMP, and MPI parallel programming on multicore GPU clusters. *Computer Physics Communications*, *182*(1), 266–269. https://doi.org/10.1016/j.cpc.2010.06.035

Zareiforoush, H., Minaei, S., Alizadeh, M. R., & Banakar, A. (2015). Potential Applications of Computer Vision in Quality Inspection of Rice: A Review. *Food Engineering Reviews*, *7*(3), 321–345. https://doi.org/10.1007/s12393-014-9101-z

Zaripov, D. I., Li, R., Mikheev, N. I., & Dushin, N. S. (2018). Speed-up algorithm based on parallel projection correlation technique for planar PIV: Accuracy

and limitation. *Flow Measurement and Instrumentation*, *60*, 88–94. https://doi.org/10.1016/j.flowmeasinst.2018.02.019

Zhang, J., Li, M., Sun, Z., Liu, H., Sun, H., & Yang, W. (2018a). Chlorophyll Content Detection of Field Maize Using RGB-NIR Camera. *IFAC-PapersOnLine*, *51*(17), 700–705. https://doi.org/10.1016/j.ifacol.2018.08.114

Zhang, J., Li, M., Sun, Z., Liu, H., Sun, H., & Yang, W. (2018b). Chlorophyll Content Detection of Field Maize Using RGB-NIR Camera. *IFAC-PapersOnLine*, *51*(17), 700–705. https://doi.org/10.1016/j.ifacol.2018.08.114

Zhang, S., Li, W., Jing, Z., Yi, Y., & Zhao, Y. (2017). Comparison of Three Different Parallel Computation Methods for a Two-Dimensional Dam-Break Model. *Mathematical Problems in Engineering*, *2017*, 1970628. https://doi.org/10.1155/2017/1970628

Zheng, H., Zhou, X., He, J., Yao, X., Cheng, T., Zhu, Y., Cao, W., & Tian, Y. (2020). Early season detection of rice plants using RGB, NIR-G-B and multispectral images from unmanned aerial vehicle (UAV). *Computers and Electronics in Agriculture*, *169*(January), 105223. https://doi.org/10.1016/j.compag.2020.105223