Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

Unidad Zacatenco

Departamento de Computación

Caracterización de tareas y recursos para la simulación de un calendarizador en un ambiente heterogéneo

Tesis que presenta

Raimer Salas González

para obtener el Grado de

Maestro en Ciencias

en Computación

Directores de Tesis

Dr. Amilcar Meneses Viveros

Dr. J. Guadalupe Rodríguez García

Ciudad de México **Agosto de 2023**

Índice general

Resumen

En los últimos años, la industria tecnológica ha experimentado una creciente demanda en la creación y expansión de centros de datos, diseñados para manejar grandes volúmenes de información. Dentro de este panorama, algunos centros de datos presentan una composición heterogénea, integrados por conjuntos de servidores con capacidades variables de procesamiento y almacenamiento. Esta diversidad puede representar un desafío en la optimización y uso eficiente de los recursos, haciendo de la calendarización una tarea intrincada. Para enfrentar estos retos, se han desarrollado diversas estrategias, basadas en indicadores como el tiempo de ejecución y el consumo energético. Estas estrategias requieren la creación de una matriz de costos que vincule tareas y recursos, lo cual implica una predicción precisa de los indicadores seleccionados, en función de las características de las tareas y los servidores. Una solución promisoria para este problema consiste en una detallada caracterización de tareas y recursos, formando un vector de características que pueda ser interpretado por algoritmos de Inteligencia Artificial. Estos algoritmos actuarían como regresores, proporcionando predicciones del indicador de interés. En este trabajo, se busca proponer una caracterización novedosa de tareas y servidores, creando un conjunto de atributos que sirvan como entradas para diferentes algoritmos de Inteligencia Artificial, facilitando así predicciones precisas y permitiendo la evaluación de la efectividad de diferentes algoritmos, la relevancia de características específicas, entre otros factores clave.

Palabras clave: calendarización, sistemas heterogéneos, Inteligencia Artificial, predicción de tiempo.

Abstract

In recent years, the technology industry has experienced a growing demand for the creation and expansion of data centers, designed to handle large volumes of information. Within this landscape, some data centers exhibit a heterogeneous composition, consisting of sets of servers with varying processing and storage capabilities. This diversity can pose a challenge in optimizing and efficiently utilizing resources, making scheduling a complex task. To tackle these challenges, various strategies have been developed, based on indicators such as execution time and energy consumption. These strategies require the creation of a cost matrix that links tasks and resources, which involves a precise prediction of the selected indicators, based on the characteristics of the tasks and servers. A promising solution to this problem involves a detailed characterization of tasks and resources, forming a feature vector that can be interpreted by Artificial Intelligence algorithms. These algorithms would act as regressors, providing predictions of the indicator of interest. In this work, we aim to propose a novel characterization of tasks and servers, creating a set of attributes that can serve as inputs for various Artificial Intelligence algorithms, thus facilitating precise predictions and allowing for the evaluation of the effectiveness of different algorithms, the relevance of specific features, among other key factors.

Keywords: scheduling, heterogeneous systems, Artificial Intelligence, servers.

Agradecimientos

Agradezco al Consejo Nacional de Ciencia y Tecnología (CONACyT) por financiar de manera gratuita los dos años de maestría, además de brindarme seguro médico y una beca decente.

Agradezco al Centro de Investigación y Estudios Avanzados (CINVESTAV) por abrirme sus puertas, y más en lo particular al Departamento de Computación por sus grandes enseñanzas y recursos para desarrollarme profesionalmente.

Agradezco a mis asesores, el doctor Amilcar Meneses Viveros, y el doctor José Guadalupe Rodríguez García, por ser referentes y guías en el transcurso de mi investigación. Además de manera general, a todos los profesores del departamento por sus grandes enseñanzas, mil gracias por brindarme lecciones importantes para el desarrollo profesional y para la vida. Así mismo a todo el resto del personal, muchas gracias por su apoyo.

Agradezco a mi familia y en especial a mis padres, su apoyo y guía en todo momento fue crucial, como siempre. A mi pareja por ser mi apoyo cuando creí que no podía más, por estar en las buenas y malas, y nunca dejar de creer en mi, eternamente gracias.

A mis compañeros de maestría, tanto los de Cuba como los de México, y en especial al Iván y al Moi, por apoyarme en momentos difíciles, y a su familia que me recibió con los brazos abiertos. Gracias enornmes.

Capítulo 1

Introducción

1.1 Motivación

En las últimas décadas se ha observado un aumento significativo en la cantidad y diversidad de los datos. Se estima que en el 2020, de manera aproximada existía 64.2 zettabytes, considerando lo que se produjo, capturó o consumió. Se espera que para 2025 esta cifra aumente a 181 zettabytes esperándose que su evolución continúe Taylor (2022). Se deben tener formas de poder procesar estos datos para producir información útil, que pueda ser consumida por clientes específicos, en esferas como la medicina, la ingeniería, los negocios, la física, las ciencias sociales y muchas más. Debido a esto se hizo necesaria la creación de centros de procesamiento de datos. Éstos son instalaciones tecnológicas que albergan cientos de equipos procesamiento y almacenamiento. Debido a su gran capacidad de cómputo, procesan enormes cantidades de tareas y datos a velocidades altas.

Existe una correspondencia entre el aumento de datos y el aumento de los centros de datos. A fines del 2020, según un informe de Technology (2020), los ingresos por sistemas de hardware para centro de datos aumentaron a nivel mundial a más de $3.3 mil millones de dólares, un aumento de más del 12 % al año. El gasto en el mercado de servidores alcanzó en 2021 los $94.900 millones de dólares. El valor del mercado global de los centros de datos según Fernández (2022) en 2020 fue de 187 billones de dólares, en 2021 215 billones, esperándose que para el 2030 aumente a 517 billones, con una tasa de crecimiento anual compuesto del 10.5 % desde el 2020 hasta el 2030.

De la misma forma en que han crecido los centros de datos, también ha crecido la tecnología que lo sustenta. No obstante, una mayor inversión no significa mejor utilización de los recursos. Según Ryckbosch et al. (2011) el indicador *Total Cost Ownership* (TCO), relaciona los costos de uno o varios recursos de un centro de datos con las ganancias que proporcionan en su vida útil. Enfatizando que los costos capitales de los servidores dominan el TCO, con un 69 porciento del TCO mensual, relacionado con los costos de compra y mantenimiento. Sin embargo, se explica que el costo de toda la infraestructura y la energía para mantener un datacenter equivale a dos veces los costos de compra y mantenimiento. Concluyendo que con los costos de electricidad y energía subiendo, los costos de las instalaciones de los centros de datos proporcionales al consumo de energía del mismo, constituirán una parte mayor del indicador TCO. En otras palabras el TCO de un data center será una función que dependerá en su mayor parte del consumo de energía. Por tanto, representa un problema a tomar en cuenta en el uso eficiente de los recursos para disminuir el consumo de energía. Sin embargo, no es un problema trivial

de abordar, y se han brindado soluciones como la compra de equipos con características específicas, para lograr un mejor rendimiento en determinados aspectos. Esto implica que en muchos casos los equipos que se compran no sean de iguales propiedades, generándose un entorno heterogéneo.

Los indicadores asociados a los centros de datos permiten estudiar y analizar sus repercusiones económicas, sociales y ambientales. Algunos de estos indicadores pueden ser, consumo de energía (que tiene una relación directa con la huella de carbono y huella de agua), costos de productos específicos, rendimiento o una combinación de éstos. Uno de los indicadores principales es el de proporcionalidad energética que mide la cantidad de potencia que usa un servidor por nivel de carga. Muchos de los indicadores asociados a los centros se pueden expresar en función de la proporcionalidad energética Ryckbosch et al. (2011). Sin embargo existen otros indicadores que no tienen correlación con la proporcionalidad energética, lo que permite que se puedan generar indicadores multivariados para el estudio sustentable de centros de datos Roque Díaz (2021).

En un entorno heterogéneo se dificulta la asignación eficiente de recursos, y esto se aprecia en el comportamiento de los indicadores, donde se pueden encontrar mínimos o máximos(por ejemplo una asignación eficiente puede repercutir en la cantidad de energía que se utiliza en horas pico). Los centros de datos consumen, según la International Energy Agency (IE), alrededor de 200 teravatios de electricidad por hora. Lo que representa el 1 % de la demanda eléctrica global; generando un estimado del 0.3 % de la huella de carbono a nivel mundial, y es que la carga de trabajo de los centros de datos entre 2010 y 2020 se multiplicó por 6. Una solución al problema de asignación de recursos en entornos heterogéneos se le conoce como calendarizadores para entornos heterogéneos.

1.2 Antecedentes

Existen varios métodos y propuestas que han tratado de abordar el problema desde varios enfoques. Por ejemplo en el Departamento de Computación del Centro de Investigación y Estudios Avanzados, se propuso Cabello Sánchez (2017), donde se presenta un marco de desarrollo para soportar programación paralela por tareas en sistemas de cómputo híbridos heterogéneos. Esta propuesta tiene como objetivo simplificar el diseño y la implementación de aplicaciones, aprovechando las ventajas de los sistemas híbridos heterogéneos y asegurando la escalabilidad de las aplicaciones que se están desarrollando. En el trabajo se permite la creación y distribución de tareas y está compuesto por una interfaz de programación de aplicaciones (Application Programming Interface o API) y un middleware que proporciona servicios en tiempo de ejecución como distribución de datos y calendarización.

En Roque Díaz (2021) se propuso un indicador multivariado ponderado para analizar de forma más eficiente el rendimiento energético de un servidor de supercomputadoras. En donde se pudo concluir la relevancia que tienen factores como el tipo de gabinete y el sistema de refrigeración con respecto a la proporcionalidad energética en este tipo de servidores.

1.3 Planteamiento del Problema

Existen principalmente dos grandes grupos de algoritmos de calendarización, calendarizadores dinámicos y calendarizadores estáticos. En los calendarizadores estáticos, no hay falla en las tareas, los recursos están disponibles todo el tiempo, y se tienen previamente

definidas las tareas y sus precedencias, esto no pasa en la calendarización dinámica Amalarethinam and P (2011). Dentro de la calendarización estática, según Gilda (2013a) se definen dos grandes grupos de algoritmos, el primero se basa en heurísticas y el segundo en búsquedas aleatorias guiadas.

Una tendencia actual es abordar el problema de calendarización en entornos heterogéneos con algoritmos de Inteligencia Artificial, demostrando que la combinación de estos con heurísticas entregan mejorías en indicadores puntuales tales como tiempo de ejecución y energía consumida. Por ejemplo en Hu et al. (2020) se propone una nueva heurística, en donde se pretende optimizar como restricción principal el consumo de energía.

El objetivo principal de los calendarizadores en supercomputadoras es optimizar el uso de sus recursos. Esta optimización se puede ver en los tiempos de respuesta, en las cargas, o en la energía que usan las supercomputadoras, entre otras. Por lo que se puede asociar diversos incadores para cada uno de estos comportamientos.

El tipo de problemas de los calendarizadores es representado en la clase NP-completos, en Ullman (1975) se aborda una prueba detallada del por qué, el problema de la calendarización es de tipo NP-completo. Sabiendo esta característica, se puede entender que para brindar soluciones a este tipo de problemas, no se puede abarcar todo el espacio de posibilidades que se genera, ya que es demasiado extenso para ser computable en tiempo polinomial, por lo que es necesaria la utilización de heurísticas o de algoritmos no determinísticos, que sirvan como guía para ir encontrando de manera progresiva soluciones que se consideren buenas.

Debido al auge que ha tenido la Inteligencia Artificial en los últimos años y los enormes avances que se han hecho en este campo, se ha intentado vincular las heurísticas con algoritmos de Aprendizaje Automático y más específicamente de Aprendizaje por Refuerzo, para brindar soluciones más exactas y que minimicen el tiempo total de calendarización Rjoub et al. (2020).

Sin embargo muchas heurísticas no son totalmente utilizables en la práctica, ya que asumen directamente que se tiene una matriz de costo de ejecución en donde las filas representan las tareas, las columnas las distintas unidades de procesamiento con las que cuente el sistema y la intersección es el tiempo de ejecución de la tarea en la unidad de procesamiento. Esa presuposición, no tiene en cuenta en muchos casos lo complejo que puede ser predecir el tiempo de ejecución de una tarea específica en alguna unidad de procesamiento Agarwal et al. (2006).

Muchos algoritmos se han propuesto atacar el problema de predecir el tiempo de ejecución de una tarea específica (o de un conjunto de ellas) en distintas unidades de procesamiento o recursos. Por ejemplo en Bielecki and Śmiałek (2022) hacen uso del KNN(K-vecinos más cercanos), para establecer la semejanza que existe entre la ejecución de una tarea con sus k-vecinas más cercanos y así poder estimar su tiempo. Basado en el supuesto de que dos tareas si son muy parecidas deben tener tiempos parecidos.

Sin embargo para hacer uso de algoritmos de Inteligencia Artificial se necesita tener una caracterización de tareas y recursos. Según Agarwal et al. (2006) que sea lo suficientemente descriptiva para lograr obtener un error relativo de predicción lo suficientemente bajo. Para que entonces, las matrices de costos con las que trabajan los algoritmos de calendarización, sean lo suficientemente exactas.

Entonces, actualmente no hay un consenso general para lograr un vector de características genérico de tareas y recursos, que sirva como entrada a cualquier algoritmo de Inteligencia Artificial y resuelva todo tipo de problemas, según Gilda (2013a) cada algoritmo posee ventajas y desventajas que dependen de muchas variables. La adquisición y representación de datos para entrenar un algoritmo de Inteligencia Artificial representa uno de sus principales retos.

Por tanto, es posible establecer una relación directa que sugiere que para lograr una asignación eficiente mediante un algoritmo de calendarización que se base en una matriz de costos, es necesario disponer de información detallada sobre las propiedades de las tareas y los recursos disponibles. Estas características pueden incluir aspectos como el tamaño del problema, el nivel de recursividad y la cantidad de hilos asociados a cada tarea. Además, resulta fundamental conocer las especificaciones de los recursos donde se ejecutarán las tareas, tales como la memoria de virtualización, el tipo de sistema de archivos, la capacidad de la memoria RAM y la velocidad de comunicación, entre otros.

Entonces, sería interesante preguntarnos ¿Qué tan eficaces son las descripciones de tareas y recursos actuales para lograr un vector de características que sirva como entrada a un algoritmo de Inteligencia Artificial? ¿Cuál sería la mejor caracterización de tareas y recursos para obtener una mejora en algún indicador que previamente se defina? ¿Se podrá predecir mejor una asignación entre tarea y recurso teniendo un vector de características solamente de tareas?, ¿Qué algoritmo de Inteligencia Artificial se ajustaría más al vector de características para brindar predicciones con errores bajos?

1.4 Propuesta de Solución

Se pretende llevar a cabo una descripción exhaustiva de los atributos inherentes a cada tarea. Esto incluirá consideraciones como la cantidad de accesos a discos, la memoria requerida y otros aspectos relevantes. Además, se busca identificar y definir los atributos característicos de las unidades de procesamiento disponibles. Estos atributos abarcarán la cantidad de memoria, la frecuencia del procesador, el tamaño del disco, el tipo de procesador y la tecnología utilizada.

Para lograr una organización y análisis efectivos, se planifica agrupar tanto las tareas como las unidades de procesamiento en conjuntos específicos. Estos conjuntos serán definidos en base a características compartidas, lo que permitirá optimizar la asignación de tareas a recursos afines.

Además de la optimización en términos de tiempo de ejecución, se incorporará el factor de proporcionalidad como un elemento clave en la toma de decisiones. Esto implica no solo buscar un rendimiento óptimo, sino también priorizar la sustentabilidad del sistema, considerando aspectos como el consumo energético.

Para lograr un enfoque completo y sólido, se emplearán tanto métodos cualitativos como cuantitativos en el proceso de análisis y diseño de la asignación de tareas en este entorno heterogéneo.

Una estrategia clave para abordar este desafío es la construcción de un vector de características para cada tarea. Este vector, que contendrá información relevante sobre cada tarea, será utilizado como entrada para diversos algoritmos de Inteligencia Artificial. Estos algoritmos serán capaces de predecir el tiempo de demora de las tareas, lo que permitirá una asignación más eficiente y equilibrada.

Además de la predicción del tiempo de demora, se llevará a cabo una evaluación exhaustiva de indicadores clave. Entre estos indicadores se destacan la potencia promedio, que se refiere al consumo de energía de los recursos durante la ejecución de las tareas, y el tiempo de ejecución, que mide la duración total de la asignación y ejecución de las tareas.

La combinación de la aplicación de técnicas de Inteligencia Artificial con la medición de estos indicadores permitirá obtener una visión más completa y precisa del rendimiento del sistema de asignación de tareas. Esto facilitará la toma de decisiones informadas y la optimización tanto en términos de eficiencia temporal como de consumo energético.

1.5 Objetivo General

Desarrollar una caracterización de tareas y recursos, que sirva como entrada a un conjunto de algoritmos de Inteligencia Artificial para predecir un indicador que se utilice en una función objetivo para optimizar la calendarización estática en entornos heterogéneos.

1.6 Objetivos Específicos

1. Revisar una revisión de el estado del arte sobre calendarizadores, políticas de calendarización y algoritmos utilizados para calendarizar.

2. Realizar una revisión del estado del arte sobre las formas utilizadas para predecir el tiempo de ejecución de una tarea basándose en las caracterización de la tareas y las caracterización de los recursos en donde se ejecuten

3. Analizar y seleccionar un conjunto de algoritmo de Inteligencia Artificial que permita predecir el consumo de energía o el tiempo de ejecución como indicador.

4. Desarrollar una caracterización de tareas que cubra la mayor cantidad de descriptores que se puedan realizar.

5. Desarrollar una caracterización de recursos que cubra la mayor cantidad de descriptores.

6. Realizar simulaciones para medir el tiempo de ejecución o el consumo de energía de una tarea en un determinado recurso, y poder obtener un vector de características que sirva como entrada a un conjunto de algoritmos de Inteligencia Artificial para predecir el tiempo de ejecución y el consumo enérgetico de una tarea en un determinado recurso.

7. Determinar la importancia de cada característica encontrada, en el vector que se introduzca al conjunto de algoritmos de Inteligencia Artificial.

1.7 Estado del Arte

En sistemas distribuidos y paralelos la calendarización es una parte amplia del problema de asignación de recursos y probablemente es una de las más cuidadosamente estudiadas CHAPIN (1996). Según CHAPIN (1996) se puede plantear de la siguiente forma "¿Cómo ejecutar un conjunto de tareas T en un conjunto de recursos P sujeto a alguna

condición o conjunto de éstas C?" Uno de los objetivos principales de la calendarización ha sido minimizar el tiempo de ejecución. Sin embargo no es el único, ejemplos de otros objetivos incluyen la minimización del costo, minimización en la demora de las comunicaciones, minimización de la energía consumida, confiabilidad del sistema, darle prioridad a los procesos de determinados usuarios o necesidades de hardware especializados CHAPIN (1996).

En la mayoría de los casos, la calendarización se realiza en entornos donde los nodos, o clusters, o sistemas de computación, no poseen las mismas propiedades, a estos se les llama entornos heterogéneos. Según Grasso et al. (2013) los clústeres de nodos heterogéneos se utilizan cada vez más para la computación de alta demanda, debido a los beneficios que brinda en rendimiento máximo y eficiencia energética. Sin embargo esta situación viene acompañada de problemas para calendarizar. Al ser de distintas propiedades los equipos, resulta complicado decidir efectivamente a cuál nodo irá una tarea, ya que los nodos con mejores propiedades pueden estar ocupados, las tareas que quizás ocuparon los nodos con mejores propiedades no son tan demandantes, por lo que puede haber una infrautilización de los recursos, a veces la mejor máquina no garantiza el mejor rendimiento para una tarea que se ejecute en ella, entre otras razones.

Por tanto los estudios se han enfocado en atacar el problema de la calendarización en entornos heterogéneos de distintas formas, usando Algoritmos Genéticos, proponiendo nuevas heurísticas, haciendo analogía con problemas de Teoría de Juegos, creando metaheurísticas bioinspiradas tales como el algoritmo del enjambre, Inteligencia Articifial en muchas variantes(Redes neuronales, Reconocimiento de Patrones, Aprendizaje Automático, entre otras).

Por ejemplo en Hu et al. (2020) se propone una nueva heurística para ordenar las prioridades de las tareas. Además, se describe cómo el consumo de energía ha aumentado en los últimos años y por ende existe la necesidad de mejorar este aspecto. Proponiéndose así un método de calendarización, donde se pretende optimizar como restricción principal el consumo de energía. Así como un mecanismo de preasignación basado en los niveles de energía de cada tarea, donde se proveen estrictas pruebas para asegurarse del resultado. Para comprobar esta hipótesis se realizan diversas simulaciones y experimentos, para luego compararlos con otros algoritmos hechos. Dando como resultado una mejora considerable en el apartado de consumo de energía.

Para representar el problema de calendarización en entornos heterogéneos, en muchos casos, se ha optado por crear grafos dirigidos acíclicos (DAG),Sakellariou and Zhao (2004). Donde cada nodo representa una tarea a ejecutar y cada dirección del grafo representa una restricción de precedencia, o simplemente una precedencia. En la cual los nodos enlazados no pueden ejecutarse hasta que su antecesor termine. En Sakellariou and Zhao (2004) se describe una nueva heurística basada en la observación que diferentes métodos para computar los pesos, cuando se realiza la calendarización de grafos de tipo DAG en máquinas heterogéneas, puede conducir a variaciones significativas en la calendarización generada. Con la heurística descrita en este artículo se minimizaron dichas variaciones resolviendo series de problemas de calendarización de tareas independientes. Luego se realizan comparaciones con otras heurísticas de vanguardias logrando resultados favorables.

De la misma forma en Tariq et al. (2019) intentan a partir de un grafo DAG, probar una nueva heurística, para calendarizar las tareas intentando que los costos de comunicaciones promedio se minimicen y se escoja, una vez calendarizado, el mejor recurso disponible. Para comprobar la hipótesis anterior se realizan comparaciones con otros algoritmos de

calendarización de tareas bien conocidos, en apartados tales como, tiempo de calendarización, eficiencia y tamaño del calendarizador. Una vez que se hacen dichas comparaciones se pueden observar mejoras con respecto a los elementos de tiempo de calendarización y eficiencia, con respecto a los algoritmos comparados.

En Zaman et al. (2019) se realiza una observación, en la cual la conocida heurística para calendarizar tareas en recursos limitados llamada $MinMin$, presenta resultados de utilización desbalanceados en los recursos, especialmente cuando la mayoría de las tareas tienen bajos requirimientos computacionales. Por esa razón se propone un nuevo modelo computacional, donde cada recurso tiene una cierta capacidad limitada para ejecutar un cierto número de tareas simultáneamente. Basado en el modelo mencionado anteriormente, se propone una nueva heurística llamada Extended High to Low Load (ExH2LL), que intenta balancear la carga de trabajo a través de los recursos computacionales disponibles, mientras se mejora dicha utilización de recursos y el tiempo total de calendarización. Para la comprobación de esta hipótesis se compara ExH2LL con $MinMin$, y las variaciones que se le han realizado a $MinMin$. Los resultados de las simulaciones demuestran que ExH2LL supera a MinMin y a todas sus variaciones, con respecto al tiempo total de calendarización y a la eficiencia en la utilización de recursos.

En Kuchaki Rafsanjani and Khatibi Bardsiri (2012) se propone una nueva heurística unida a un algoritmo para calendarizar meta-tareas en entornos heterogéneos. La heurística creada tiene como objetivo mejorar el rendimiento tanto en el tiempo total de calendarización como en la utilización efectiva de los recursos, reduciendo el tiempo de inactividad del recurso a utilizar. El análisis de rendimiento mostró que el algoritmo y la heurística propuesta, tienen una mejor tasa de utilización de recursos y reduce significativamente el tiempo total de calendarización que otros algoritmos y heurísticas conocidos.

Otro enfoque para atacar este tipo de problema es la creación de Algoritmos Genéticos que ayuden a minimizar o maximizar objetivos concretos, por ejemplo en Zhang et al. (2016) se propone un algoritmo genético bi-objetivo BOGA. En éste se trata de integrar dos objetivos, que normalmente entran en conflicto, la confiabilidad del sistema y el consumo de energía. Sobre la base de problemas de la vida real y situaciones generadas aleatoriamente. En este artículo se presenta cómo este algoritmo comparado con otros excelentes algoritmos tales como Multi-objective heterogeneous earliest finish time (MOHEFT) funciona mucho mejor en el apartado de consumo de energía como resultado final de la calendarización.

En Paneerselvam et al. (2010) se propone un enfoque basado en los paradigmas de la computación evolutiva para resolver problemas de calendarización. La solución al problema se divide en tres fases; planificación, programación y optimización. Inicialmente, la lógica difusa se aplica para la planificación y luego la programación se optimiza utilizando algoritmos informáticos evolutivos como el algoritmo genético (GA) y la optimización de enjambre de partículas (PSO). El conocido problema de instancia 10×10 de Adams, Balas y Zawack se selecciona como el problema de referencia experimental y se simula utilizando MATLAB R2008b. Los resultados de las técnicas de optimización se comparan con parámetros como el intervalo de producción, el tiempo de espera, el tiempo de finalización y el tiempo transcurrido. Se analiza la evaluación del desempeño de las técnicas de optimización y se determina la técnica evolutiva superior para resolver el problema de calendarización heterogénea. Además se concluye que se pueden realizar mejoras significativas modificando los objetivos expuesto en el artículo y adoptando técnicas para ampliar el conocimiento de los problemas de calendarización heterogénea. La investigación abordó específicamente el problema clásico de calendarización heterogénea de 10×10 con

el objetivo de minimizar el período de inactividad. Entendiendo que puede extenderse a un problema de mayor tamaño y analizar las actuaciones.

En Dongarra et al. (2007) se trata de lograr algo parecido al artículo Zhang et al. (2016). Utilizando de igual forma un algoritmo genético, pero esta vez sin tomar en cuenta el indicador de consumo de energía. En éste se describe un algoritmo bi-objetivo para lograr optimizar la confiabilidad y el tiempo de ejecución. Para ello se asume que cada recurso va a tener una probabilidad de fallo ajustado a una regla exponencial. Luego para optimizar los dos indicadores propuestos se propone un algoritmo que aproxima la curva de pareto.

En El-Shorbagy (2016) se propone un algoritmo genético híbrido. Se realiza una representación cromosómica del problema que se basa en llaves generadas aleatoriamente. Los calendarizadores propuestos se construyen utilizando una regla de prioridad en donde dichas prioridades se definen por el mismo algoritmo. En estos calendarizadores se realiza un procedimiento que genera calendarizadores parametrizados activos. Después de calendarizar se aplica una heurística de búsqueda local, que mejora la solución obtenida. Este enfoque es probado en un conjunto de instancias estándar que se toman de la literatura, y se comparan con otros enfoques resultando en cálculos computacionales que validan la eficacia del algoritmo propuesto.

Otra forma de atacar el problema, como se ha mencionado es a través de Inteligencia Artificial, y dentro de la calendarización, también se ha podido ver cómo han existido variantes para afrontarla. Un ejemplo sería Agarwal et al. (2006) en este artículo se provee una nueva heurística, junto con una formulación de una red neuronal aumentada (AugNN) para minimizar el tiempo. Se exploran cuatro reglas para las tareas y tres reglas para máquinas o recursos, lo que da como resultado doce combinaciones de heurísticas simples. Las reglas o políticas para las tareas son Highest-Level-First (HLF), Highest-Total-Remaining-Processing-Time-First (HTRPTF), Smallest-Latest-Finish-Time-First (SLFTF) y Minimum-Slack-First (MSF). Para el caso de las máquinas o recursos, se provee una nueva regla de tipo greedy o voraz, llamada Fastest-Available-Machine-First (FAMF), y dos reglas no voraces llamadas, Fastest-Available-Machine-First-With-Conditional-Wait-1 (FAMF-CW-1), y (2) Fastest-Available-Machine-First-With-Conditional-Wait-2 (FMF-CW-2). El enfoque de AugNN integra el aprendizaje de Redes Neuronales con el dominio y el conocimiento específico del problema a través de la heurística, para producir buenos resultados. Se brinda la formulación AugNN para cada una de las 12 heurísticas y se muestran resultados computacionales en 100 problemas generados aleatoriamente de tamaños que van desde 20 a 70 tareas y 2 a 5 máquinas. Los resultados demuestran que AugNN proporciona una mejora significativa utilizando las heurísticas de un solo paso. La reducción en la brecha entre la solución obtenida y el límite inferior debido a AugNN sobre la heurística de un solo paso osciló entre el 24,4 por ciento y el 50 por ciento. En cuanto a la heurística, las reglas de prioridad de máquina no voraces funcionaron significativamente mejor que la regla codiciosa. Las brechas promedio para las reglas no codiciosas oscilaron entre el 16,1 por ciento y el 23,5 por cientos.

En Gupta et al. (2020) se explica cómo el crecimiento de la rama de Inteligencia Artificial en Aprendizaje Automático ha brindado herramientas para poder crear modelos de predicción, que sirvan para atacar el problema de calendarización en entornos heterogéneos. Proponiendo así una red neuronal artificial que es usada para lograr una maximización en el rendimiento. Para esto se trata de predecir el comportamiento de la aplicación en los siguientes intervalos de calendarización usando las estadísticas generadas en los núcleos de los recursos y las cargas de trabajo. El método propuesto produce una mejora de rendimiento promedio en el rango del 6.5 por ciento hasta el 9.7 por ciento del

algoritmo convencional de calendarización Fairness-Aware y entre un 1.5 por ciento hasta un 2.5 por ciento de un algoritmo novel dinámico de calendarización que usa igualmente redes neuronales articiales.

En Song et al. (2022) se identifica que en trabajos recientes de calendarización, usando aprendizaje reforzado, se encuentran problemas para lidiar con la flexibilidad del sistema, lo que permite que una operación sea calendarizada en una de varias máquinas existentes. Por tanto en el artículo se considera el bien conocido problema Flexible Job-shop Scheduling Problem (FJSP), que representa un subproblema del problema de calendarización, en donde se intenta maximizar la flexibilidad del sistema, objetivo el cual fue descrito anteriormente. Para atacar el problema descrito anteriormente, se propone un nuevo método de aprendizaje profundo reforzado, para que aprenda nuevas reglas de prioridad de despacho de tareas. Además se trata de combinar la selección de operaciones o también la selección de tareas y la asignación de recursos como un objetivo compuesto. Para lograr esto se basan en una nueva representación de un grafo heterogéneo de estados de calendarización, la cual captura relaciones complejas entre tareas u operaciones y las máquinas o recursos. Los experimentos muestran que el método propuesto es mucho mejor que los métodos tradicionales de despacho de tareas, y es computacionalmente eficiente aún en problemas de escala mayores.

Como bien se ha descrito en párrafos anteriores, una forma muy utilizada para estructurar el problema de calendarización es usando grafos DAG. Sin embargo en Lin et al. (2022) se detecta que en la mayoría de los algoritmos de calendarización, no se puede utilizar la información completa de los calendarizadores, y están diseñados para calendarizar solamente un grafo DAG a la vez. Por tanto en Lin et al. (2022) proponen un algoritmo de calendarización basado en aprendizaje reforzado para computación en entornos heterogéneos. Primeramente crean una red neuronal, con atención a grafos para procesar completamente la información generada en la calendarización y poder codificarla. Luego una red neuronal completamente conectada selecciona un nodo apropiado del grafo DAG. Finalmente los experimentos hechos en varios escenarios de calendarización con varios tipos de tareas muestran que el algoritmo propuesto reduce el tiempo de terminación promedio de las tareas en un 13.03 por ciento sobre los algoritmos existentes.

En Nemirovsky et al. (2017) se presenta un algoritmo de calendarización de CPUs que maximiza el rendimiento del sistema. Para ello se utiliza un modelo de Aprendizaje Automático que prediga el rendimiento de múltiples subprocesos que se realizan en diversos recursos del sistema. Se demuestra además que las redes artificiales ligeras pueden proporcionar predicciones de rendimiento altamente precisas para un conjunto diverso de aplicaciones. Lo que ayuda a mejorara la eficiencia de la calendarización en entornos heterogéneos. Por consiguiente se aprecia que el enfoque propuesto produce mejores cuantificables en torno al 25 o 31 por ciento en comparación con los calendarizadores convencionales para tareas que se abordan de forma intensiva en el CPU y la memoria.

Capítulo 2

Marco de referencia

Este capítulo abordará algunos elementos que permitirán tener un mejor entendimiento acerca de la calendarización. Tales como, el significado de proceso, cómo están compuestos los procesos, qué es una tarea, cómo se pueden representar los procesos, así como la relación entre las tareas y los recursos.

2.1 Tareas

En el contexto de la calendarización en servidores HPC, una tarea es una unidad básica de trabajo que se ejecuta en un recurso computacional. En el artículo de Ilavarasan and Thambidurai (2007) los autores tratan a las tareas como unidades calendarizables al igual que en el libro "Scheduling for Parallel Processing"de Drozdowski (2009) aunque en dicho libro agregan que la definición de tareas puede variar dependiendo del modelo de calendarización y del nivel de abstracción.

Por ejemplo se menciona que en un sistema de computación real una tarea puede ser un hilo de ejecución de un proceso, puede también ser un proceso en si misma, un conjunto de procesos, una comunicación o un conjunto de comunicaciones. En el artículo Bhatti et al. (2018) los autores señalan que una tarea puede representar un pequeño segmento de código que sea computable, aunque no se define exactamente qué tamaño o propiedades debe tener. Por esta razón en muchos casos se opta por la utilización de un algoritmo de particionado que sea capaz de dividir la tarea en el segmento de código apropiado donde se le pueda hacer un tratamiento eficiente, como es el caso del artículo de Ilavarasan and Thambidurai (2007), donde se hace referencia del uso de este tipo de algoritmo, evidenciando que una vez que se obtenga este particionado de tarea se puede representar en un grafo DAG.

Por otro lado en el artículo de Bittencourt et al. (2018) se refieren a las tareas como un trabajo que se contiene a si mismo y posee su propio código ejecutable. Sin embargo en otros casos se pueden considerar a las tareas como unidades ejecutables más complejas que pudieran ser divididas en subtareas más pequeñas.

Por ejemplo en el libro Hager and Wellein (2010) ellos consideran la multiplicación de números flotantes como una tarea, que puede ser dividida en varias subtareas de menor tamaño. Como no existe un nivel de granularidad evidente en la bibliografía para definir a la tarea, en ésta tesis se va a referir a tarea con el concepto dado en el libro "Scheduling

for Parallel Processing"de Drozdowski (2009) donde se refiere a tarea como la entidad o unidad más que pequeña que pueda ser calendarizable y computable. Un ejemplo de una tarea puede ser el algoritmo de Strassen[1] para la multiplicación de matrices.

Sin embargo en este ejemplo y pudiendo ser extensible, es importante señalar que existen algunos conjuntos de códigos dentro del algoritmo de Strassen que pueden ser considerados también tareas, como por ejemplo la suma de matrices. Por tanto se puede deducir que como se explicó anteriormente una tarea compleja puede ser descompuesta en otras pequeñas, pero depende de lo que se quiera lograr y los retos que se deriven dentro de la calendarización, ya que quizás pueda ser más conveniente mantener una granularidad alta en algunos casos.

2.2 Procesos

Según el libro de Silberschatz et al. (2021e) un proceso se define típicamente como una unidad de trabajo en un sistema moderno de tiempo compartido, además se refiere también al proceso como un programa en ejecución como también se señala en , por ejemplo un procesador de texto en ejecución, un navegador web. Sin embargo en este libro se usa el término trabajo y proceso como conceptos intercambiables, no estableciendo una diferencia clara entre ellos. En el libro de Drozdowski (2009) igualmente mencionan que un proceso se refiere a un programa en ejecución y representa una entidad del sistema operativo. También se menciona que un proceso posee una cierta cantidad de recursos computacionales, como la asignación para su ejecución de una porción de memoria.

Los procesos poseen distintos estados como se menciona en Silberschatz et al. (2021e) que dependen de la actividad actual de dicho proceso.

Los estados se muestran a continuación:

- Nuevo. El proceso está siendo creado.
- Ejecutándose. Se están ejecutando instrucciones.
- Esperando. El proceso está esperando que ocurra algún evento (como la finalización de una operación de E/S o la recepción de una señal).
- Listo. El proceso está esperando ser asignado a un procesador.
- Terminado. El proceso ha finalizado su ejecución.

También se destaca que los nombres de estos estados varían en dependencia del sistema operativo, sin embargo de manera general todos los estados que se representaron se encuentran en todos los sistemas operativos.

En la figura 2.1 se puede observar cómo un calendarizador puede cambiar el estado de un proceso de "listo" a "ejecución".

Los sistemas operativos están compuestos por dos grandes tipos de procesos, los procesos de sistema operativo(son aquellos que ejecutan código máquina) y los procesos de usuarios (son aquellos que ejecutan código usuario). En el libro de Silberschatz et al.

[1]El algoritmo de Strassen es un algoritmo eficiente para multiplicar matrices que fue propuesto por Volker Strassen en 1969. Este algoritmo utiliza una técnica de división y conquista para reducir el número de multiplicaciones requeridas en la multiplicación de matrices Ray et al. (2016).

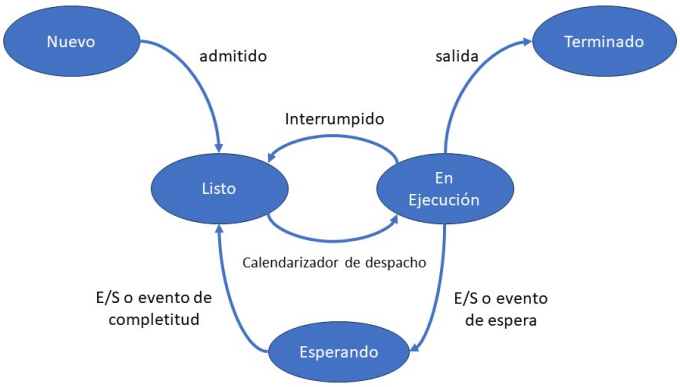

Figura 2.1: Diagrama de estado de un Proceso. Fuente: Silberschatz et al. (2021a).

(2021e) además se menciona que todos los tipos de procesos descritos anteriormente pueden ser ejecutados concurrentemente, por ejemplo aplicando una técnica de multiplexado del procesador. Aunque actualmente para los procesos de usuario existen numerosas técnicas de para la paralelización, por ejemplo el uso de tarjetas gráficas en donde un proceso puede ser enviado para acelerar su procesamiento, al mismo tiempo que se están procesando en el CPU otros procesos.

Tradicionalmente existían muchos sistemas en los cuáles sus procesos contenían un sólo hilo de ejecución, sin embargo los sistemas operativos modernos poseen la capacidad de tener múltiples hilos por proceso, lo que se le conoce también como multihilo Silberschatz et al. (2021e). Esto hace entonces que se tenga una mejor utilización del CPU, ya que diferentes hilos del mismo proceso pueden ejecutarse en paralelo en diferentes núcleos. Esto puede resultar en una mejora significativa del rendimiento. Además en aplicaciones interactivas el multihilo puede mejorar la responsividad al permitir que la aplicación continúe interactuando con el usuario mientras realiza otras actividades en segundo plano.

2.3 Hilos

Existen dos enfoques al momento de abordar los hilos en la computadora, aunque comparten muchas características, es necesario hacer distinciones a lo que se refiere, hilos de hardware e hilos de software.

Hilos de procesador: Los hilos de procesador, también conocidos como hilos de hardware, son unidades de ejecución físicas asociadas con un núcleo de procesador. Cada núcleo de procesador puede tener múltiples hilos de procesador. Estos hilos se ejecutan en paralelo y comparten los recursos del núcleo, como las unidades de ejecución y la caché. Los hilos de procesador permiten la ejecución simultánea de múltiples tareas o subprocesos en un único núcleo, lo que mejora la utilización de los recursos y la capacidad de respuesta en entornos multitarea Patterson and Hennessy (2013).

Hilos virtuales: Los hilos virtuales, también conocidos como hilos de software, son hilos de ejecución que se crean y gestionan a nivel de software. Los hilos virtuales se asignan a los hilos de procesador subyacentes y se ejecutan en ellos. Los hilos virtuales se utilizan para

aprovechar el paralelismo a nivel de hilo, permitiendo que múltiples hilos de software se ejecuten en paralelo en un solo hilo de procesador físico. El sistema operativo y el software de programación gestionan la asignación de los hilos virtuales a los hilos de procesador subyacentes, realizando conmutaciones rápidas entre ellos para lograr el paralelismo y una mejor utilización de los recursos del procesador. Los hilos virtuales se utilizan para mejorar el rendimiento y la eficiencia en la ejecución de tareas paralelas y concurrentes Patterson and Hennessy (2013).

Un hilo, de manera general, se define según el libro de Silberschatz et al. (2021e) como una unidad básica de la utilización del CPU, comprimiendo en su entorno un identificador, un contador de programa, un conjunto de registros, y una pila. En el libro de Drozdowski (2009) se menciona que los hilos de un mismo proceso no se encuentran aislados entre sí, sino que comparten memoria, y los recursos del proceso anfitrión. Por tanto las comunicaciones y la sincronización entre los hilos de un proceso son mucho más rápidas ya que no necesitan involucrar al sistema operativo como intermediario. Este tipo de conceptos constituye la base de las aplicaciones paralelas. En el libro de Silberschatz et al. (2021e) se menciona que un proceso tradicional está compuesto por un solo hilo de ejecución, sin embargo, muchos de los sistemas operativos modernos poseen la capacidad de poder trabajar con más de un hilo por proceso. Por tanto si un proceso tiene más de un hilo en su control entonces puede manejar más de una tarea a la vez. Como se puede observar en la figura 2.2.

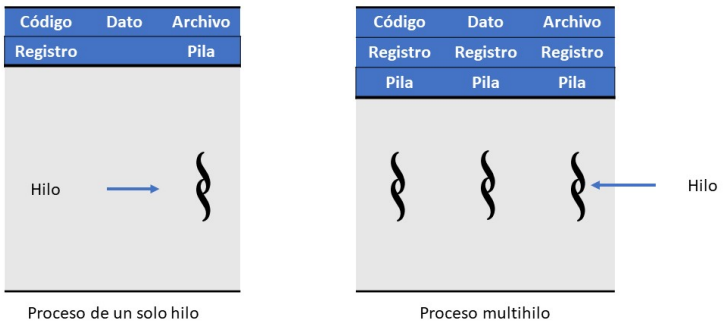

Figura 2.2: Hilo Simple y Múltiple Hilo. Fuente: Silberschatz et al. (2021b).

En el libro de Silberschatz et al. (2021e) se describen algunos beneficios de la programación multihilo.

- **Responsividad**: El uso de subprocesos en una aplicación interactiva suele permitir que el programa continúe ejecutándose, incluso incluso si parte de él está bloqueado o realiza una operación prolongada. Por ejemplo en una aplicación en la cuál el usuario realiza un clic en un botón y se consume mucho tiempo. Una aplicación de un solo hilo sería irresponsiva, ya que habría que esperar que se consumiera el tiempo de la acción prevista para el clic al botón para poder pasar a otro proceso. Sin embargo, si se ejecutara en un sistema multihilo la aplicación seguiría funcionando sin problemas.

- **Compartición de Recursos**: Los procesos tradicionalmente requieren técnicas especiales como la memoria compartida y el paso de mensajes para compartir recursos entre ellos. Estas técnicas deben ser cuidadosamente implementadas por el programador. Por otro lado, los subprocesos comparten automáticamente la memoria y los recursos del proceso al que pertenecen. Esto significa que múltiples subprocesos pueden trabajar juntos en una aplicación sin la necesidad de técnicas adicionales de comunicación. Esto facilita la creación de aplicaciones con múltiples hilos de ejecución que comparten código y datos dentro del mismo espacio de memoria, lo que brinda mayor eficiencia y flexibilidad en el diseño y la ejecución del programa.

- **Econonomía**: La asignación de memoria y recursos para crear procesos es costosa en términos de tiempo y recursos. En cambio, debido a que los subprocesos comparten los recursos del proceso al que pertenecen, es más económico crear y cambiar entre subprocesos. Esto significa que el tiempo y los recursos necesarios para crear y administrar subprocesos son considerablemente menores que los requeridos para los procesos. Un ejemplo concreto es el sistema operativo Solaris, donde crear un proceso lleva aproximadamente treinta veces más tiempo que crear un subproceso Silberschatz et al. (2021e). Esta diferencia en el costo entre procesos y subprocesos hace que los subprocesos sean una opción más eficiente y económica en términos de tiempo y recursos en comparación con los procesos individuales.

- **Escalabilidad**: La escalabilidad de los hilos puede ser aún mayor en una arquitectura multiprocesador, donde los hilos pueden ejecutarse en paralelo en diferentes núcleos de procesamiento. Un proceso de un solo hilo solo puede ejecutarse en un procesador, independientemente de cuántos estén disponibles.

También resulta importante hacer una diferenciación entre lo que significa paralelismo y concurrencia, conceptos que están estrechamente relacionados con los hilos. En el libro de Silberschatz et al. (2021e) se explica de la siguiente manera

- **Concurrencia**: Para ilustrar mejor el concepto se plantea el siguiente ejemplo. Si se tiene un sistema que maneja un sólo núcleo, y se tiene una aplicación que necesita cuatro hilos de ejecución, la concurrencia solamente signficaría una intercalación entre los hilos de ejecución para simular que la ejecución se está realizando al mismo tiempo. Esto se produce ya que un núcleo solo puede ejecutar un hilo a la vez. Esto se podría visualizar en la figura 2.4 donde los cuatro hilos serían intercalados a lo largo del tiempo.

- **Paralelismo**: Si tomamos el mismo ejemplo anterior pero aumentando la cantidad de núcleos, se pudiera entender paralelismo como la capacidad del sistema de ejecutar varios hilos en el mismo instante de tiempo. Esto sucede porque el sistema puede asignar un hilo separado por cada núcleo que se tenga. Esto se puede ilustrar mejor en la figura 2.3 donde existen dos núcleos y cada conjunto de hilos fueron asignados a cada núcleo.

Según el libro de Silberschatz et al. (2021e) existen de manera general dos tipos de paralelismos, el paralelismo de datos y el paralelismo de tareas.

- Paralelismo de datos: Este tipo de paralelismo se enfoca en la distribución de subconjuntos de los datos en múltiples núcleos de cómputo y realizar la misma operación

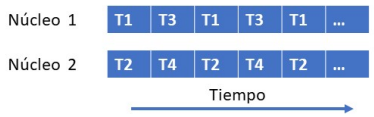

Figura 2.3: Ejecución paralela en un sistema multinúcleo. Fuente: Silberschatz et al. (2021d).

en cada núcleo. En el libro de Silberschatz et al. (2021e) se examina el ejemplo siguiente: Considera sumar los contenidos de un arreglo de tamaño N. En un sistema de un solo núcleo, un hilo simplemente sumaría los elementos $[0] \ldots [N-1]$. Sin embargo, en un sistema de dos núcleos, el hilo A, ejecutándose en el núcleo 0, podría sumar los elementos $[0] \ldots [\frac{N}{2} - 1]$ mientras que el hilo B, ejecutándose en el núcleo 1, podría sumar los elementos $[\frac{N}{2}] \ldots [N-1]$. Los dos hilos estarían ejecutándose en paralelo en núcleos de cómputo separados."

- Paralelismo de Tareas: Este tipo de paralelismo requiere la distribución no de datos sino de tareas(o hilos en el libro de Silberschatz et al. (2021e) lo manejan indistintamente) a través de múltiples núcleos de cómputo.

Sin embargo hay que hacer una diferenciación entre paralelismo a nivel de núcleo y a nivel de hilo. El paralelismo a nivel de hilo se refiere a la capacidad de ejecutar múltiples hilos simultáneamente en un único núcleo de procesador. En este enfoque, se pueden ejecutar múltiples tareas o subprocesos en paralelo, lo que permite una mejor utilización de los recursos del procesador y una mayor capacidad de respuesta en entornos multitarea. Cada hilo de ejecución se ejecuta de manera independiente, pero comparte los recursos del núcleo, como las unidades de ejecución y la caché Patterson and Hennessy (2013).

El paralelismo a nivel de núcleo se refiere a la capacidad de un procesador para ejecutar múltiples núcleos de procesamiento de manera simultánea. Cada núcleo es una unidad de procesamiento independiente que puede ejecutar instrucciones de manera simultánea con otros núcleos. Esto permite una mayor capacidad de procesamiento y rendimiento en tareas paralelizables, ya que los núcleos pueden realizar cálculos en paralelo sin interferir entre sí Patterson and Hennessy (2013).

Los hilos de procesador se implementaron por primera vez con los procesadores multinúcleo a finales de la década de 2000. Los procesadores multinúcleo permiten la ejecución simultánea de múltiples hilos de procesador en un solo chip, lo que mejora la capacidad de procesamiento y la capacidad de respuesta en entornos multitarea Tanenbaum and Bos (2014b) .

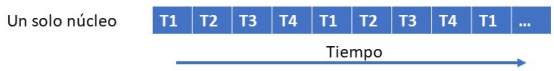

Figura 2.4: Ejecución concurrente en un sistema de un sólo núcleo. Fuente: Silberschatz et al. (2021c).

Por otro lado, los hilos virtuales se implementaron con tecnologías como Hyper-Threading de Intel y Simultaneous Multi-Threading (SMT) de AMD. Estas tecnologías se introdujeron en procesadores a partir de la década de 2000 y permiten la creación y ejecución de múltiples hilos virtuales en un solo hilo de procesador físico. Los hilos virtuales se utilizan para aprovechar el paralelismo a nivel de hilo y mejorar la eficiencia en la ejecución de tareas paralelas y concurrentes Tanenbaum and Bos (2014b).

Es importante tener en cuenta que la implementación y disponibilidad de hilos de procesador y hilos virtuales pueden variar entre diferentes generaciones y modelos de procesadores. Cada fabricante puede tener su propia tecnología y terminología específica para referirse a estas características.

Según el libro de Drozdowski (2009) las comunicaciones entre hilos se dan cuando un hilo necesita datos de entrada producido por otro hilo para ejecutarse en una instancia de tiempo determinada. Este tipo de comunicaciones a menudo deriva en la sincronización cuando se necesita acceder a una variable global en un momento determinado. Además, las comunicaciones entre hilos de ejecución introducen un nuevo término, llamado precedencia donde un hilo A al depender de los datos que produzca otro hilo B tiene que ejecutarse lógicamente al terminar el B. Estas precedencias entre hilos suelen representarse a través de las aristas en un grafo de tareas.

2.4 Trabajos

En el entorno de la calendarización, típicamente, se le conoce a un conjunto de tareas como un trabajo, definición que se menciona en el libro de Drozdowski (2009) . Cada trabajo se divide en pequeñas tareas, que se pueden calendarizar y ejecutar en diferentes nodos del sistema. Sin embargo, es conveniente mencionar que no existe en muchos libros y artículos, una diferencia clara entre tarea y trabajo y a veces se puede utilizar indistintamente, como por ejemplo en el artículo de Bittencourt et al. (2018). Por tanto para futuros usos del término en esta tesis se va a utilizar el concepto de trabajo empleado en el libro "Scheduling for Parallel Processing"de Drozdowski (2009).

Cada trabajo puede tener una representación de sus conjuntos de tareas a través de varias formas. En un estudio de Topcuoglu et al. (2002a) se menciona que la representación de las tareas que componen al trabajo puede ser hecha mediante un grafo de tipo DAG, donde los nodos representan a las tareas, las aristas representan la relación de precedencia entre un nodo y otro, y la valencia del nodo representa el tiempo que se demora la tarea el/los recursos que se dispongan. Esta representación resulta ser útil ya que permite visualizar las dependencias entre las tareas de forma clara y concisa. De esta manera, se puede analizar la estructura de las tareas y determinar la secuencia de ejecución más adecuada para minimizar el tiempo total o maximizar el uso de recursos. Además, se pueden aplicar algoritmos de teoría de grafos para encontrar caminos críticos, identificar tareas con mayor impacto en el rendimiento del sistema y optimizar la planificación.

Un elemento importante a tener en cuenta en los grafos DAG es la predicción de los tiempos de ejecución de las tareas en los distintos dispositivos, que representa la valencia de los nodos, porque permite realizar una planificación de tareas más precisa y eficiente. Si se tiene una buena estimación de los tiempos de ejecución de las tareas en cada dispositivo, se podría realizar una asignación de tareas más equitativa y balanceada, lo que puede reducir el tiempo total de ejecución del trabajo y maximizar el uso de los recursos.

Existen diferentes formas para predecir el tiempo de ejecución de las tareas en sistemas computacionales, algunas de ellas son:

- Modelos analíticos: se utilizan fórmulas matemáticas para estimar el tiempo de ejecución de una tarea. Estos modelos se basan en la complejidad computacional de la tarea y en las características del sistema en el que se ejecutará.
- Técnicas empíricas: se basan en la observación de tiempos de ejecución pasados de tareas similares. Estas técnicas pueden ser útiles cuando se dispone de suficientes datos históricos.
- Aprendizaje automático: se pueden utilizar algoritmos de aprendizaje automático para predecir el tiempo de ejecución de una tarea en función de diferentes características de la tarea y del sistema. Estos algoritmos pueden entrenarse con datos históricos y ajustarse para mejorar su precisión en la predicción.
- Simulación: se pueden utilizar simulaciones para predecir el tiempo de ejecución de una tarea. En estos casos se simula la ejecución de la tarea en un entorno virtual, lo que puede proporcionar una estimación precisa del tiempo de ejecución.

Existen varias representaciones y enfoques para abordar problemas de calendarización además de los grafos DAG. Algunas de estas representaciones y enfoques incluyen:

1. **Matrices de adyacencia:** Las matrices de adyacencia se utilizan para representar relaciones entre tareas en un problema de calendarización. Las filas y las columnas representan las tareas, y los elementos de la matriz indican las relaciones entre ellas, como precedencias o tiempos de procesamiento.
2. **Listas de prioridad:** Las listas de prioridad ordenan las tareas según cierto criterio, como la fecha límite, el tiempo de procesamiento o la importancia. Este enfoque se utiliza a menudo en algoritmos de construcción y búsqueda local para resolver problemas de calendarización.

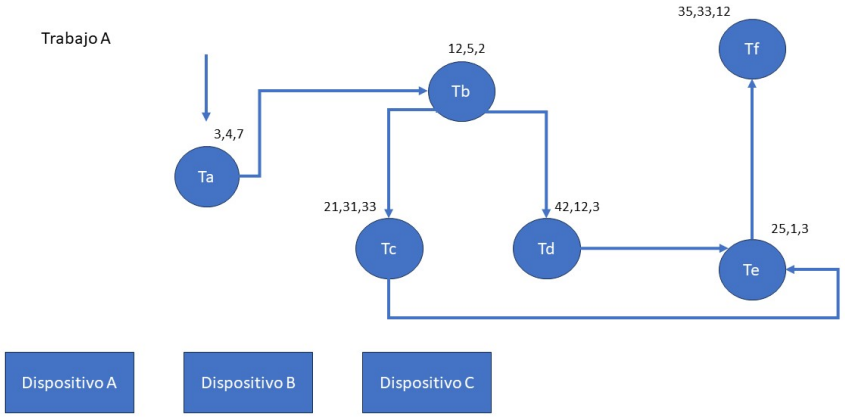

Figura 2.5: Grafo DAG de un Trabajo A

2.5 Grafo Dirigido Acíclico(DAG)

Un grafo dirigido acíclico es un grafo dirigido sin ciclos. Como se ha podido observar en secciones anteriores, los grafos DAG son una parte importante dentro de la representación del problema de calendarización. En el libro de Drozdowski (2009) se menciona que este tipo de representación son una parte ubicua en los calendarizadores. En dicho libro también dan una definición de algunos de los conceptos principales de teoría de grafos y clases de grafos que componen al problema de calendarización.

- **Grafo** Un grafo se define como un par $G = (V, E)$, donde V es un conjunto de nodos (vértices) y E es un conjunto de aristas. Una arista se representa como un par de nodos pertenecientes a V. Por ejemplo, $(i, j) \in E$ denota una arista entre los nodos i y j. Es importante destacar que en un grafo no hay un orden específico entre los elementos de un par, y las aristas no tienen una dirección definida.

- **Grafo Dirigido** Por otro lado, un grafo dirigido se define como un par $G = (V, A)$, donde V es el conjunto de nodos y A es el conjunto de arcos. Un arco se representa como un par ordenado $(i, j) \in A$. En este caso, el arco (i, j) indica que hay una dirección desde el nodo i hacia el nodo j. Un nodo sin arcos entrantes se denomina nodo de entrada, mientras que un nodo sin arcos salientes se denomina nodo de salida.

En el libro de Drozdowski (2009) además se menciona que en el contexto de la calendarización, los grafos se utilizan para representar las interconexiones de sistemas informáticos o el orden de ejecución de tareas dentro de un trabajo. Es importante tener en cuenta que el término "nodo"se utiliza en el contexto del grafo específico y puede representar una computadora en una red, o una tarea en un grafo de tareas.

- **Grado del nodo**: El grado de un nodo $i \in V$, denotado como $\deg(i)$, es el número de aristas incidentes en ese nodo. Se denota por $\deg(G)$ al máximo grado de cualquier nodo en el grafo G, es decir, $\deg(G) = \max_{i \in V} \deg(i)$.

En el caso de grafos dirigidos, es necesario distinguir entre el número de arcos que comienzan o terminan en un nodo en particular. Por lo tanto, se denota por indeg(i) al número de arcos que entran en el nodo i, y por outdeg(i) al número de arcos que comienzan desde el nodo i.

- **Camino**: Un camino en un grafo dirigido es una secuencia de nodos $(a_1, \ldots, a_k)$ tal que $a_i \in V$ para $i = 1, \ldots, k$ y $(a_i, a_{i+1}) \in A$ para $i = 1, \ldots, k-1$. Un ciclo en un grafo dirigido es un camino $(a_1, \ldots, a_k)$ tal que $(a_k, a_1) \in A$. Para grafos no dirigidos, las definiciones son análogas, pero se utilizan aristas $f(a_i, a_{i+1})$ en lugar de arcos (a_i, a_{i+1}).

Un nodo v en V se considera un nodo de salida si no existe ningún arco (i, h) en E, es decir:
$\forall u \in V, (i, j) \notin E$
De manera similar, un nodo de entrada se puede definir matemáticamente como:
Un nodo v en V se considera un nodo de entrada si no existe ningún arco (i, j) en E, es decir:
$\forall u \in V, (i, j) \notin E$
Estas definiciones indican que un nodo de salida no tiene arcos salientes y un nodo de entrada no tiene arcos entrantes en el grafo DAG.

Un grafo ponderado en los nodos es un grafo en el cual a cada nodo se le asocia un número. Un grafo ponderado en los arcos (o aristas) es un grafo en el cual a cada arco (o arista) se le asocia un número. También existen grafos ponderados en los arcos y en los nodos. Los pesos de los nodos y los arcos tienen una interpretación inmediata en el grafo de tareas. El peso de un nodo representa el tiempo de procesamiento de una tarea representada por el nodo. El peso de un arco puede representar la duración del tiempo de comunicación o la cantidad de datos transferidos entre las tareas. La suma de los pesos a lo largo de un camino puede interpretarse como una distancia. Dependiendo de la aplicación, se pueden sumar los pesos de los nodos, de los arcos o de ambos a lo largo del camino para calcular la distancia. El camino más largo sin ciclos se llama camino crítico. Es importante tener en cuenta que la definición precisa del camino crítico depende de la forma de calcular la distancia en el grafo.

Un grafo acíclico se llama árbol si no es dirigido. Un DAG es un bosque de salida si y solo si para cada nodo i existe a lo sumo un nodo j para el cual existe un arco (j, i) en A. De manera análoga, un DAG es un bosque de entrada si y solo si para cada nodo i existe a lo sumo un arco (i, j) en A. El nodo sin predecesores en el bosque de salida, o un nodo sin sucesores en el bosque de entrada, se llama raíz. Los bosques de entrada o de salida con solo una raíz se llaman árboles de entrada o árboles de salida, respectivamente.

2.6 Sistemas Distribuidos

Aunque el tema que aborda el tópico de Sistemas Distribuidos resulta ser muy extenso y podría alcanzar perfectamente para un libro, en esta sección, se presentan algunos conceptos y definiciones relacionados con este tipo de sistemas que son de interés para este trabajo. Según el libro de Coulouris et al. (2011a) se define como un **Sistema Distribuido** aquel que los componentes de hardware o software localizados en computadoras interconectadas se comunican y coordinan sus acciones únicamente mediante el intercambio de mensajes. Para ellos esa definición simple abarca toda la gama de sistemas en la

que las computadoras en red pueden ser utilizadas de manera útil. Sin embargo no es la única definición, por ejemplo, en el libro de Tanenbaum and Steen (2006a) mencionan que en la literatura se han tratado de ofrecer varias definiciones de Sistemas Distribuidos, sin embargo ninguna de ellas ha sido realmente satisfactoria y ninguna de ellas guardan relación con ninguna de las otras. Por tanto, en Tanenbaum and Steen (2006a) se define Sistemas Distribuidos, de manera general, de la siguiente forma "Un sistema distribuido es una colección de computadoras independientes que se presenta a sus usuarios como un sistema único y coherente." En el libro de Tanenbaum and Steen (2006a) se menciona que en esta definición hay varios aspectos importantes a tener en cuenta, el primero es que un Sistema Distribuido, está compuesto por componentes que son autónomos, estos componentes pueden ser, por ejemplo, computadoras. Y el segundo elemento es que los usuarios crean que están interactuando con un sistema único. Por tanto se deduce que los sistemas autónomos necesitan colaborar entre sí. Pero según el libro de Tanenbaum and Steen (2006a) y la bibliografía consultada Coulouris et al. (2011b), Arpaci-Dusseau and Arpaci-Dusseau (2018) es preferible concentrarse en las características que componen a un Sistema Distribuido.

Según el libro de Tanenbaum and Steen (2006a) una característica importante que poseen los sistemas distribuidos es que las diferencia entre las distintas computadoras y las formas de comunicarse entre sí, son ocultas para el usuario. Otra característica importante a tener en cuenta es que los usuarios que interactúan con un sistema distribuido generalmente deben tener la experiencia con el sistema de una forma consistente y uniforme, independientemente del lugar o el momento en el que la interacción tenga lugar, o sea que el sistema se comporte sin ninguna falla provocada por la región o el momento. Éste tipo de sistema, como se menciona en el libro de Tanenbaum and Steen (2006a) debe ser además fáciles de expandir o escalar, ya que representa una característica que es consecuencia de tener computadoras independientes entre sí.

Por otra parte como se menciona en Coulouris et al. (2011b) un sistema distribuido debe tener las siguientes consideraciones:

- **Concurrencia**: En una red de computadoras interconectadas, la concurrencia debe estar siempre presente. El sistema distribuido debe ser capaz de manejar distintas peticiones a la par. En este caso se puede entender concurrencia en relación al concepto dado de paralelismo en la sección de **Hilos**, lo que a nivel macro.

- **Falta de reloj global**: Cuando los programas necesitan cooperar, coordinan sus acciones intercambiando mensajes. La coordinación a menudo depende de una idea compartida del momento en que se va a realizar las acciones entre programas. Sin embargo, existen límites en la presición con las que las computadoras en una red pueden sincronizar sus relojes. No existe una noción única global de tiempo que se correcta. Esto resulta ser una consecuencia directa de que la única forma de comunicación es mediante el envío de mensajes.

- **Resiliencia al Fallo**: Todo sistema informático es propensos al fallo. Los sistemas distribuidos pueden presentar fallas en la red, lo que deriva en el aislamiento de las computadoras que están conectadas. Sin embargo, esto no significa que dejen de funcionar. Es responsabilidad del diseñador de sistemas ofrecer una robustez necesaria para seguir manteniendo todo aún cuando ocurra alguna falla determinada.

Una clase importante de **Sistemas Distribuidos** y que se relaciona directamente con este trabajo, son los usados para realizar computación de alto desempeño. En el libro de

Tanenbaum and Steen (2006a) se pueden distinguir dos subgrupos dentro de este tipo de sistemas: **Computación en Cluster** [2] y la **Computación en Malla**.

- **Computación en Cluster**: En el libro de Tanenbaum and Steen (2006a) se menciona que los sistemas en cluster empezaron a tomar auge luego de que la relación precio-rendimiento de las computadoras personales y las estaciones de trabajo mejoró. Desde una perspectiva financiera y técnica se volvió atractiva la idea de invertir en la construcción de supercomputadoras interconectadas por una red de alta velocidad para realizar procesamientos intesivos.

  La Computación en Cluster se utiliza en prácticamente todos los casos para la programación paralela en la que un programa único que requiera cálculos intensivos se ejecute en múltiples máquinas. En el libro de Tanenbaum and Steen (2006a) se menciona que un ejemplo bien conocido de Computación en Cluster es el formado por los cluster basados en Linux Beowulf. En este caso cada cluster consiste de una colección de nodos de computación que son controlados y accesados por un solo nodo maestro (ver figura 2.6). El nodo maestro es el encargado de administrar los nodos que se encargarán de ejecutar un programa paralelo especifíco, manteniendo una cola de Trabajos (ver sección 2.4) enviados y además proporcionando una interfaz para los usuarios del sistema. Es el nodo maestro el que ejecuta el software intermedio necesario para la ejecución de los programas y la gestión del cluster, mientras que los nodos de cálculo frecuentemente no necesitan más que un sistema operativo para su funcionamiento. A grandes rasgos se podría decir que el nodo maestro tiene la tarea de un calendarizador.

  Aunque en el libro de Tanenbaum and Steen (2006a) se menciona que una caracterítica importante de los cluster es su homogeneidad, debido a un conjunto de ventajas que se ofrece como la integración rápida, la administración eficiente entre otras. Sin embargo como se menciona en el artículo de Bohn and Lamont (2002) resulta muy rápido para los diseñadores de sistemas, agregar nuevo hardware que resulte más poderoso que el que se tiene, aunque esto produce al mismo tiempo un uso no óptimo de los recursos. No obstante como se menciona en Carpenter et al. (2022) los sistemas modernos de computación de alto desempeño, se están volviendo más heterogéneos, afectando a todos los componentes, como unidades de procesamiento, jerarquías de memorias, componentes de red y sistemas de almacenamiento. De hecho, un cúmulo grande de trabajos se enfocan en resolver temas puntuales de la Computación en Cluster haciendo la consideración de su heterogeneidad, ejemplo de esto son los trabajos de Goglin (2014),Fadika et al. (2012),Bohn and Lamont (2002) y Qin and Jiang (2005).

- **Computación en Malla**: En el libro de Jacob et al. (2005) se menciona que la Computación en Malla puede significar cosas diferentes para varias personas. Se explica que a menudo, se presenta como una visión grandiosa, similar a las redes eléctricas, donde los usuarios (o dispositivos eléctricos) obtienen acceso a la electricidad a través de enchufes de pared sin preocuparse ni considerar dónde ni cómo se genera realmente la electricidad. En esta visión de la computación en malla, la computación se vuelve omnipresente y los usuarios individuales (o aplicaciones clientes) obtienen acceso a recursos informáticos (procesadores, almacenamiento, datos,

[2] Cluster es un anglicismo utilizado dentro del contexto de las tecnologías de la información y significa el proceso de integrar dos o más computadoras para que trabajen simultáneamente en el procesamiento de una determinada tarea.

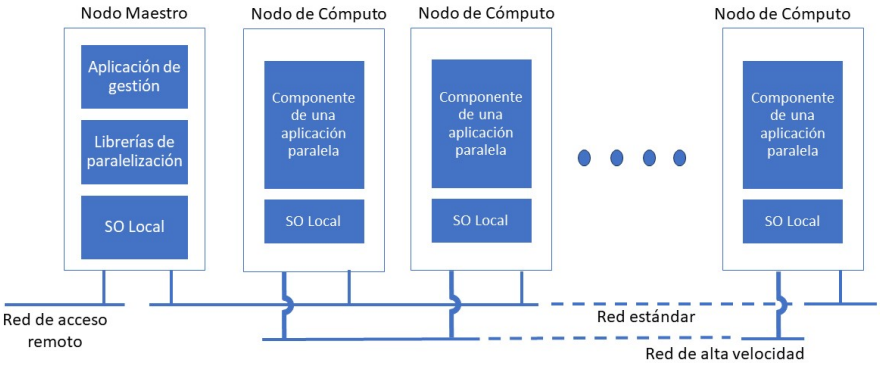

Figura 2.6: Ejemplo de un sistema de Computación en Cluster. Fuente: Tanenbaum and Steen (2006b).

aplicaciones, etc.) según sea necesario, sin tener conocimiento en dónde se encuentran esos recursos ni de las tecnologías subyacentes, hardware, sistema operativo, etc. Aunque se explica que esta visión resulta ser un poco grandiosa y utópica, debido a los grandes desafíos en cuestiones técnicas, comerciales, políticas y sociales que sigue teniendo esta área. Por otra parte, según Tanenbaum and Steen (2006a) los sistemas de Computación en Malla tienen un alto grado de heterogeneidad: no se hacen suposiciones sobre el hardware, los sistemas operativos, las redes, los dominios administrativos, las políticas de seguridad, etc. En los sistemas de Computación en Malla se reúnen recursos de diferentes organizaciones para permitir la colaboración de un grupo de personas o instituciones. Esta colaboración se realiza en forma de una organización virtual. Las personas que pertenecen a la misma organización virtual tienen derechos de acceso a los recursos que se proporcionan a esa organización. Típicamente, los recursos consisten en servidores de cómputo (incluidos supercomputadoras, posiblemente implementadas como clústeres de computadoras), instalaciones de almacenamiento y bases de datos. Además, también se pueden proporcionar dispositivos de red especiales como telescopios, sensores, etc. Una arquitectura típica de de Computación de Malla es la propuesta por Jacob et al. (2005) como se muestra en la figura 2.7.

2.7 Virtualización

La Virtualización es un tema ampliamente estudiado en los sistemas operativos modernos, sus detalles intrínsecos y alcances, son material suficiente para dedicar amplios textos que se introduzcan en sus profundidades. Sin embargo en esta sección se presenta un resumen, tomando algunos conceptos claves que lo hacen útil y que están estrechamente relacionados con criterios a tomar en cuenta durante la calendarización.

En el libro de Silberschatz et al. (2021e) se define la Virtualización como una tecnología que permite a los sistemas operativos ejecutar aplicaciones dentro de otro sistema operativo. Resulta que esta definición no contempla los matices que involucran el ¿por

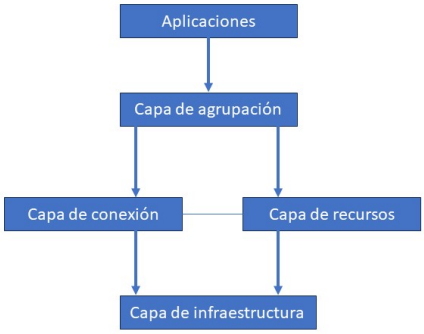

Figura 2.7: Arquitectura por capas de un Sistema de Computación en Malla. Fuente: Tanenbaum and Steen (2006c).

qué? de su uso. Por lo tanto en el libro de Tanenbaum and Bos (2014b) se brinda una forma de entender la virtualización a raíz de un ejemplo concreto. Supongamos que una empresa tiene un conjunto de computadoras alojadas en un bastidor(o rack [3]) conectadas por una red de alta velocidad, donde cada uno alberga un servicio distinto, un servidor de correo electrónico, un servidor web, un servidor FTP, algunos servidores de comercio electrónico, entre otros.

Existen varias razones para adoptar este modelo de máquinas separadas ofreciendo distintos servicios, uno de ellos podría ser, que una sola máquina no puede albergar la carga completa del procesamiento de todos los servicios, otra razón es la confiabilidad: la administración según el libro de Tanenbaum and Bos (2014b) simplemente no confía en que un sistema operativo funcione todos los días del año sin interrupciones. Entonces si se coloca los servicios en máquinas separadas, al menos se podría asumir que si uno de los servidores sufre algún bloqueo, los otros no se van a ver afectados. Esto además resulta ser beneficioso para la seguridad del sistema, ya que si un intruso o persona mal intencionada logra compremeter el servidor quizás el servidor web, no se van a ver afectados correos confindenciales alojados en el servidor de correo. Aunque esta solución de tener servidores alojados en máquinas distintas resulta ser buena, debido a su tolerancia a fallos y mantener un considerable aislamiento, es también excesivamente costosa y difícil de administrar, debido a la gran cantidad de máquinas usadas. En el libro de Tanenbaum and Bos (2014b) se menciona además que algunos empresas u organizaciones dependen de más de un sistema operativo para sus operaciones diarias: un servidor web en Linux, un servidor de correo en Windows, un servidor de comercio electrónico para clientes que se ejecuta en OS X, y algunos otros servicios que se ejecutan en varias versiones de UNIX. Entonces de aquí se desprende el problema, del gran costo que involucra tener máquinas físicas independientes y separadas para lograr obtener los beneficios que se describieron anteriormente. Una solución a este problema es la creación de máquinas virtuales. La idea de la virtualización o como se le conoce técnicamente según Tanenbaum and Bos (2014b)

[3]Rack es un término inglés que se emplea para nombrar a la estructura que permite sostener o albergar un dispositivo tecnológico. Se trata de un armazón metálico que, de acuerdo a sus características, sirve para alojar una computadora, un router u otra clase de equipo.

un Monitor de Máquinas Virtuales, es crear varias máquinas virtuales compartiendo los recursos de hardware dentro de una misma máquina física, en donde además se puedan potencialmente ejecutar varios sistemas operativos. A un Monitor de Máquinas Virtuales según Tanenbaum and Bos (2014b) se le conoce también como hipervisor.

Aunque el término de la tecnología de máquinas virtuales resulte relativamente nuevo a partir del auge tan grande que ha tomado en los último tiempos, resulta que es una tecnología un tanto antigua, que se inició en IBM, como un método para que mútliples usuarios de un mismo sistema pudieran ejecutar aplicaciones concurrentemente como se describe en el libro de Silberschatz et al. (2021e). Aunque para más historia, en el libro de Tanenbaum and Bos (2014b) se hace un descripción más profunda sobre el uso inicial de esta tecnología.

Un hipervisor es un software o firmware que se ejecuta directamente en el procesador y actúa como una capa intermedia entre el hardware físico y las máquinas virtuales. El hipervisor se encarga de gestionar y controlar los recursos del sistema, como el procesador, la memoria, el almacenamiento y los dispositivos de entrada/salida. Proporciona a cada máquina virtual un entorno virtualizado en el que puede ejecutar su propio sistema operativo y aplicaciones, como si estuvieran funcionando en una máquina física independiente. En la bibliografía se distinguen dos tipos de hipervisores Goldberg (1972):

- Hipervisor de Tipo 1 (o nativo): Se instala directamente sobre el hardware y controla directamente los recursos del sistema. Actúa como un sistema operativo independiente y las máquinas virtuales se ejecutan sobre él. Tanenbaum and Bos (2014b)
- Hipervisor de Tipo 2 (o hospedado): Se instala como una aplicación dentro de un sistema operativo existente. Requiere que el sistema operativo anfitrión esté en funcionamiento para gestionar los recursos del sistema y proporcionar soporte para las máquinas virtuales. Tanenbaum and Bos (2014b)

En la figura 2.8 se puede observar la localización de cada uno de estos tipos de hipervisores.

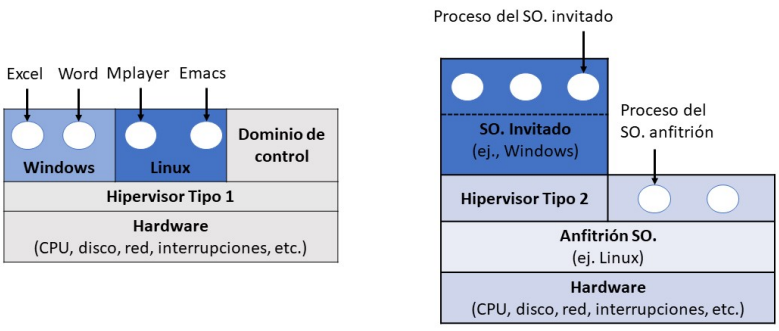

Figura 2.8: Localización de los hipervisores. Fuente: Tanenbaum and Bos (2014a).

La virtualización en los procesadores ofrece una serie de beneficios, como la consolidación de servidores, la flexibilidad en la asignación de recursos, el ahorro de costos y

la simplificación de la administración de sistemas. También permite ejecutar múltiples sistemas operativos o aplicaciones en un solo procesador, lo que mejora la eficiencia y la utilización de los recursos Tanenbaum and Bos (2014b).

Por otra parte también cabe mencionar que existe también la virtualización en la memoria, que permite crear una capa de abstracción entre el software y la memoria física de un sistema informático. Con la virtualización en la memoria, múltiples sistemas operativos o aplicaciones pueden ejecutarse de manera simultánea en un mismo sistema físico, cada uno de ellos con su propio espacio de memoria virtual.

En un entorno de virtualización en la memoria, cada sistema operativo o aplicación tiene asignado un espacio de memoria virtual que es independiente de la memoria física subyacente. Cada espacio de memoria virtual se divide en páginas, que son unidades de memoria de tamaño fijo. Estas páginas se mapean en la memoria física a través de tablas de páginas, que mantienen la correspondencia entre las direcciones virtuales y las direcciones físicas Tanenbaum and Bos (2014b).

El hipervisor o monitor de máquinas virtuales se encarga de gestionar las tablas de páginas y realizar las traducciones necesarias entre las direcciones virtuales y las direcciones físicas. Cuando un sistema operativo o aplicación accede a una dirección virtual, el hipervisor intercepta la operación y realiza la traducción correspondiente, asegurándose de que el acceso se realice a la ubicación correcta en la memoria física.

La virtualización en la memoria también permite implementar técnicas como la asignación dinámica de memoria, donde se asigna y libera memoria según las necesidades de cada sistema operativo o aplicación. Esto proporciona flexibilidad en el uso de los recursos de memoria y permite una mejor utilización de la memoria física disponible Tanenbaum and Bos (2014b).

Además, la virtualización en la memoria también ofrece mecanismos de aislamiento y protección, asegurando que cada sistema operativo o aplicación no pueda acceder ni interferir con la memoria asignada a otros sistemas. Esto garantiza la seguridad y el rendimiento de cada entorno virtualizado.

En resumen, la virtualización en la memoria utiliza técnicas de mapeo y traducción de direcciones para proporcionar a cada sistema operativo o aplicación un espacio de memoria virtual independiente. Esto permite la ejecución simultánea de múltiples sistemas operativos o aplicaciones en un mismo sistema físico, mejorando la eficiencia, la flexibilidad y la seguridad del entorno virtualizado.

Otro concepto importante a tener en cuenta dentro de la virtualización es el espacio de intercambio o swap [4]. El swap de memoria es un mecanismo utilizado en los sistemas operativos para gestionar la memoria virtual cuando se agota la memoria física disponible. Consiste en mover datos desde la memoria RAM hacia un espacio en el disco duro llamado área de intercambio (swap space), liberando así espacio en la memoria RAM para que otros procesos puedan utilizarlo Tanenbaum and Bos (2014b) Silberschatz et al. (2021e).

Cuando un sistema operativo implementa la virtualización, el swap de memoria se relaciona con la asignación de memoria para las máquinas virtuales. Cada máquina virtual tiene su propio espacio de memoria asignado, que puede ser mayor que la cantidad de memoria física disponible en el sistema. Cuando una máquina virtual necesita más memoria de la que tiene disponible en la RAM, se utiliza el swap de memoria para mover parte de su contenido a la memoria virtual en el disco Silberschatz et al. (2021e) Arpaci-Dusseau and Arpaci-Dusseau (2018).

[4]Swap es un anglicismo frecuentemente utilizado en habla hispana para referirse a la memoria de intercambio, su uso es más grande en los usuarios del sistema operativo linux

En este contexto, el hipervisor es responsable de gestionar el swap de memoria para las máquinas virtuales. Supervisa el uso de la memoria de cada máquina virtual y decide qué partes de la memoria deben ser intercambiadas hacia el área de intercambio en el disco cuando sea necesario. El hipervisor también se encarga de controlar el intercambio de datos entre la memoria física y el área de intercambio cuando se produce una falta de memoria Silberschatz et al. (2021e).

El swap de memoria en la virtualización es importante para garantizar que cada máquina virtual tenga suficiente espacio de memoria para ejecutar sus aplicaciones y procesos, incluso cuando la memoria física del sistema esté limitada. Sin el swap de memoria, sería difícil o incluso imposible ejecutar múltiples máquinas virtuales con grandes requerimientos de memoria en un sistema con recursos limitados Tanenbaum and Bos (2014b).

Es importante tener en cuenta que el swap de memoria introduce una penalización en el rendimiento, ya que el acceso a los datos en el disco es más lento que el acceso a la memoria RAM. Por lo tanto, es recomendable asignar suficiente memoria física a las máquinas virtuales para evitar un uso excesivo del swap de memoria y mantener un rendimiento óptimo Silberschatz et al. (2021e).

2.8 Descriptores del Dispositivo

Resulta de particular importancia para este trabajo, definir las principales características que describen al dispositivo o unidad de procesamiento donde se ejecutará una Tarea 2.4 que pertenezca a un Trabajo 2.4. Ya que continuamente éstas características tienen un impacto directo en los tiempos de ejecución de dichas tareas, y en la eficiencia general del sistema al momento de estar ejecutándose un Proceso2.2. Por ejemplo elementos tales como, la memoria RAM, el tipo de procesador, los dispositivos de almacenamiento, la red que se utiliza para interconectarse con otros dispositivos, etc, tienen un impacto directo en el rendimiento del sistema.

2.8.1. Discos

El almacenamiento de datos permite a los usuarios guardar información en dispositivos de almacenamiento, asegurando la retención de los datos incluso cuando la computadora está apagada. En vez de introducir los datos de forma manual, los usuarios pueden acceder a la información almacenada y emplearla en sus actividades. Asimismo, el almacenamiento de datos se emplea como una medida de seguridad para salvaguardar la información y evitar su pérdida. Siendo un componente además directamente relacionado con la velocidad de un proceso en ejecución que necesite consultar datos en un archivo. Aunque no todo proceso o tarea(antes de ser ejecutada en un procesador) requiere hacer uso de algún tipo de dispositivo de almacenamiento. En este contexto obviamos los dispositivos de almacenamiento primario (RAM,Caché, ROM, etc) y nos concentramos exclusivamente en los dispositivos de almacenamiento secundario.

Existen tres tipos principales de tecnología de almacenamiento secundario:

1. Magnético (HDD(Hard Disk Drives), cintas magnéticas): los datos se almacenan utilizando patrones de magnetización en una superficie especial.

2. Óptico (discos compactos, Blu-ray): los datos se almacenan en deformidades en una superficie circular que se pueden leer cuando se ilumina con un diodo láser.

3. Semiconductores: los datos se almacenan utilizando circuitos integrados basados en tecnología de semiconductores. Aunque este tipo de tecnología principalmente se usaba para el almacenamiento volátil [5], los Discos de Estado Sólido (SSD) emergieron como una tecnología de consumo que ofrece una opción no volátil con velocidades de acceso superiores a su contraparte magnética Blumzon and Pănescu (2019).

HDD:

Dentro de la tecnología de uso magnético se encuentra su opción más popular y ampliamente utilizada. Según IBM las unidades de disco duro (HDD) utilizan un disco magnético giratorio y un cabezal de escritura mecánico para manipular los datos. Los factores de forma más comunes son las unidades de 2,5 y 3,5 pulgadas, que se utilizan para computadoras portátiles y de escritorio, respectivamente. Si bien la mayoría de los HDD aprovechan una interfaz SATA, también conocida como Serial ATA, también puede encontrar conexiones Serial Attached SCSI (SAS) o Fibre Channel para uso especializado. En la figura 2.10 se pueden visualizar la ubicación de los componentes principales de un disco:

- Motor del giro: Este es el motor que hace girar los platos (platters) del disco duro. La velocidad a la que gira este motor se mide en revoluciones por minuto (RPM) y puede variar dependiendo del modelo del disco duro. Las velocidades comunes son 5400 RPM y 7200 RPM, aunque algunos discos duros de alto rendimiento pueden girar a velocidades aún mayores.

- Platos : Los platos son los discos metálicos que se encuentran dentro del disco duro y en los que se almacenan los datos. Están recubiertos con un material magnético que permite la lectura y escritura de datos. Un disco duro puede tener varios platos, y cada plato puede tener datos almacenados en ambas caras.

- Pivote : El pivote es el eje sobre el cual se mueve el brazo del actuador que sostiene las cabezas de lectura/escritura. Permite que las cabezas de lectura/escritura se muevan de manera precisa sobre la superficie de los platos.

- Conector IDE : IDE (Integrated Drive Electronics) es una interfaz de conexión para discos duros y otros dispositivos de almacenamiento. El conector IDE es donde se conecta el cable que lleva los datos desde y hacia el disco duro.

- Actuador VCM : VCM (Voice Coil Motor) es el motor que mueve el brazo del actuador y, por lo tanto, las cabezas de lectura/escritura. Funciona mediante el uso de campos magnéticos para mover el brazo de manera precisa sobre los platos.

- Cabeza de Lectura/Escritura : Las cabezas de lectura/escritura son las que leen y escriben datos en los platos. Hay una cabeza de lectura/escritura para cada cara de cada plato en el disco duro.

- Placa Base: La placa base es la estructura de metal que sostiene todos los componentes internos del disco duro. Proporciona rigidez y estabilidad al disco duro, y también ayuda a disipar el calor generado por el motor y la electrónica del disco duro.

[5] En el almacenamiento volátil los datos se pierden si no se suministra energía eléctrica, a diferencia del almacenamiento magnético u óptico

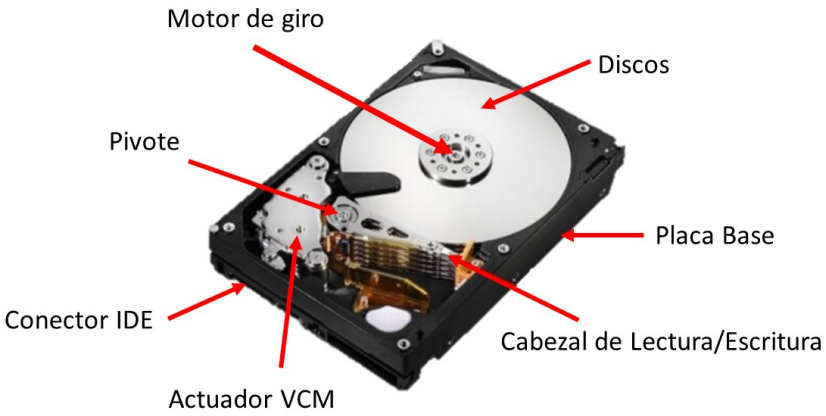

Figura 2.9: Componentes principales de un HDD. Fuente: Taktak-Meziou et al. (2015a).

Un HDD es un tipo de disco duro que utiliza tecnología de servo systems para su funcionamiento.Los servo systems son sistemas electromecánicos que controlan y posicionan con precisión los cabezales de lectura/escritura dentro de un HDD. Estos sistemas utilizan señales de retroalimentación y algoritmos de control para garantizar un posicionamiento preciso y estable de los cabezales sobre los discos magnéticos, lo que permite acceder y escribir datos de manera confiable Chen et al. (2010). En la figura 2.10 se puede observar el comportamiento de un sistema servo en un HDD

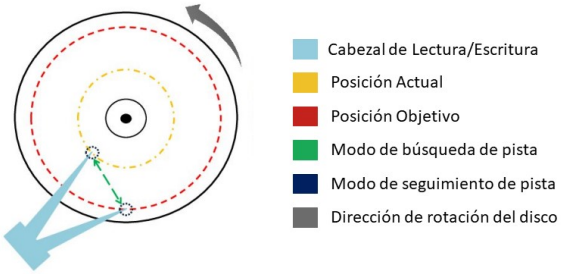

Figura 2.10: Funcionamiento de un sistema servo en un HDD. Fuente: Taktak-Meziou et al. (2015b).

Los HDD utilizan diferentes tipos de conexiones y velocidades dependiendo de la interfaz utilizada. Algunos de los tipos de conexiones más comunes son:

1. SATA (Serial ATA): Es la interfaz más común en la mayoría de los HDD modernos. Hay diferentes versiones de SATA, como SATA I, SATA II y SATA III, que ofrecen velocidades de transferencia de datos de 1.5 Gbps, 3 Gbps y 6 Gbps respectivamente.

2. SAS (Serial Attached SCSI): Es una interfaz de alta velocidad utilizada principalmente en servidores y sistemas de almacenamiento empresariales. Proporciona velocidades de transferencia de datos más rápidas que SATA, con versiones que van desde 3 Gbps hasta 12 Gbps.

3. PATA (Parallel ATA): Es una interfaz más antigua que se utilizaba en los HDD antes de la introducción de SATA. Proporciona velocidades de transferencia de datos más lentas en comparación con SATA, con una velocidad máxima de 133 Mbps. Chen et al. (2010)

las velocidades de los HDD, varían según varios factores, como la densidad de almacenamiento, la velocidad de rotación de los discos y la tecnología utilizada. Las velocidades de los HDD suelen oscilar entre 5,400 RPM (revoluciones por minuto) y 7,200 RPM, aunque existen modelos de alta velocidad que pueden alcanzar hasta 10,000 RPM.

Es importante tener en cuenta que las velocidades de transferencia de datos y las velocidades de rotación de los HDD no son directamente comparables, ya que la velocidad de rotación influye en la latencia y el tiempo de acceso a los datos, mientras que la velocidad de transferencia de datos se refiere a la tasa máxima a la que los datos pueden ser transmitidos desde y hacia el HDD Chen et al. (2010).

SSD: Por otra parte se puede identificar la opción más popular que implementa la tecnología de semiconductores, los SSD. Los SSD son dispositivos de almacenamiento de estado sólido que utilizan memoria flash para almacenar datos de forma no volátil. A diferencia de los discos duros tradicionales, que emplean discos magnéticos giratorios, los SSD no tienen partes móviles Dr. Priyankasharma (2021a). El SSD está compuesto por dos componentes principales: el controlador y la memoria flash, junto con algunos otros componentes que se encuentran en una Placa de Circuito Impreso (PCB), que está protegida dentro de una carcasa como se muestra en la figura 2.11.

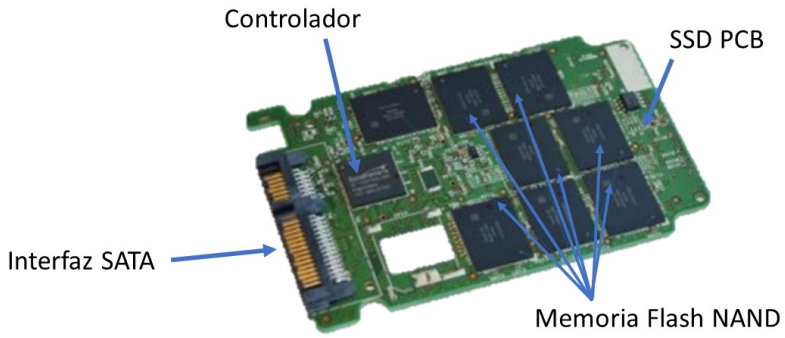

Figura 2.11: Hardware PCB del Disco de Estado Sólido. Fuente: Dr. Priyankasharma (2021b).

Los SSD están compuestos por varios componentes clave, entre ellos el controlador, la memoria flash, la memoria flash NAND y la interfaz SATA Dr. Priyankasharma (2021a).

1. Controlador: El controlador es una parte esencial del SSD, ya que se encarga de administrar el funcionamiento del dispositivo y coordinar la transferencia de datos entre la memoria flash y la interfaz de conexión. El controlador controla las operaciones de lectura, escritura y borrado de datos, así como la gestión del desgaste de la memoria flash.

2. Memoria flash: La memoria flash es donde se almacenan los datos en un SSD. A diferencia de los discos duros tradicionales que utilizan platos magnéticos, es un tipo de memoria no volátil que retiene los datos incluso cuando no hay energía eléctrica. Además se organiza en celdas de memoria y utiliza transistores de puerta flotante para almacenar los datos de forma eléctrica.

3. Memoria flash NAND: La memoria flash NAND es un tipo específico de memoria flash utilizada en los SSD. Se caracteriza por su alta densidad de almacenamiento y su capacidad para almacenar múltiples bits de datos por celda de memoria. Además es más rápida y duradera en comparación con otras tecnologías de memoria flash.

4. Interfaz SATA: La interfaz SATA (Serial ATA) es el estándar de conexión utilizado en los SSD y otros dispositivos de almacenamiento. Proporciona una conexión de alta velocidad entre el SSD y la placa madre de la computadora. La interfaz SATA se utiliza para transmitir los datos entre el SSD y la computadora, permitiendo una rápida transferencia de información.

Características notables de los SSD:

1. Velocidad: Los SSD son notablemente más rápidos en términos de acceso y transferencia de datos en comparación con los discos duros. Esto se debe a que no hay cabezales de lectura/escritura mecánicos ni platos giratorios, lo que permite un acceso más rápido a los datos almacenados en la memoria flash.

2. Tiempo de arranque y carga más rápido: Debido a su velocidad de lectura y escritura, los SSD permiten que los sistemas operativos y las aplicaciones se inicien y carguen mucho más rápido en comparación con los discos duros tradicionales.

3. Mayor resistencia: Dado que no tienen partes móviles, los SSD son más resistentes a los golpes y las vibraciones, lo que los hace ideales para su uso en dispositivos portátiles como laptops y tabletas.

4. Consumo de energía reducido: Los SSD consumen menos energía en comparación con los discos duros, lo que ayuda a prolongar la duración de la batería en dispositivos portátiles y a reducir los costos de energía en sistemas de escritorio.

5. Menor tamaño y peso: Los SSD son más compactos y livianos que los discos duros tradicionales, lo que los hace ideales para dispositivos con restricciones de espacio, como ultrabooks y computadoras de factor de forma pequeño.

6. Silencio: Debido a la ausencia de partes móviles, los SSD no generan ruido mecánico, lo que resulta en un funcionamiento más silencioso en comparación con los discos duros.

SSHD:

Por otra parte de la combinación resultante entre las tecnologías de los HDD y los SSD se pueden encontrar los Solid State Hybrid Drive (SSHD). Estos son un tipo de unidad de almacenamiento que combina las características de un disco duro tradicional (HDD) y un disco de estado sólido (SSD). Un SSHD consta de una porción de almacenamiento de disco duro convencional con una capacidad mayor y una porción de almacenamiento de memoria flash más pequeña y rápida. La idea principal detrás de un SSHD es aprovechar las ventajas de ambos tipos de almacenamiento: la capacidad de almacenamiento masivo del HDD y la velocidad y eficiencia del SSD.

El SSHD utiliza la memoria flash para almacenar los archivos y aplicaciones más utilizados y frecuentemente accedidos, lo que permite un acceso rápido y una respuesta ágil. Al mismo tiempo, los datos menos utilizados se almacenan en la porción de disco duro del SSHD.

El objetivo es ofrecer una combinación de mayor capacidad y mayor rendimiento en comparación con un HDD tradicional, a un costo más bajo que un SSD de capacidad similar. La tecnología de SSHD se utiliza en dispositivos como computadoras portátiles, consolas de juegos y sistemas de almacenamiento externo Dr. Priyankasharma (2021a).

En el estudio comparativo de Dr. Priyankasharma (2021a), se determinó que progresivamente los HDD irán siendo reemplazados por los SSD, puesto que los SSD son mejores en casi todas las métricas comparables (Para más información ver table 2.1), aunque éstos últimos tienen una vida útil mucho menos larga que los HDD, y además su recuperación de datos suele ser algo complicada. Sin embargo los clientes de estos discos desean tener lo mejor de las dos tecnologías por lo que los SSHD son la opción más fácil de usar y los más confiable en la actualidad.

Características	SSHD	SSD	HDD
Mecanismo	Magnético + memoria Flash NAND	Memoria Flash NAND NOR	Platos metálicos giratorios
Capacidad	Por encima de 2TB (laptop) Por encima de 10TB (PC escritorio)	Por encima de 1TB (laptop) Por encima de 4TB (PC escritorio)	Por encima de 2TB (laptop) Por encima de 10TB (PC escritorio)
Durabilidad	----------	Resistente al choque	Frágil
Consumo de Potencia	Promedio de 2W	Promedio de 2W	Promedio de 10W
Durabilidad	----------	MTBF > 2 millones de horas	MTBF < 700,000 horas
Ruido	----------	No tiene	Presenta
Tiempo de arranque del SO	Promedio de 10 – 15 segundos	Promedio de 10 – 15 segundos	Promedio de 30 – 40 segundos
Velocidad de apertura de archivos	Más rápido en comparación con HDD	30% más rápido en comparación con HDD	Más lento que SSD
Velocidad	Más rápido que HDD	> 200 MB/s	50 – 120 MB/s
Vibración	Mientras se utiliza HDD	No tiene vibración	Las partes móviles causan vibración
Afectaciones por el magnetismo	A veces se ve afectado	Sin efecto	Puede borrar datos
Encriptación completa del disco	Compatible	Compatible	Compatible
Costo	Económico en comparación con SSD	Costoso	Económico en comparación con SSD
Calentamiento	Moderado	Poco	Alto
Tamaño	Grande	Pequeño	Grande

Tabla 2.1: Tabla de comparación entre SSD,HDD,SSHD. Fuente: Dr. Priyankasharma (2021c).

2.8.2. Memorias

Las memorias juegan un papel fundamental en el funcionamiento de los servidores y son vitales para la ejecución eficiente de tareas. En los servidores, se requiere una cantidad considerable de memoria para manejar grandes volúmenes de datos y procesar múltiples

solicitudes simultáneamente, por lo que es una característica de vital importancia para lograr que los tiempos de ejecución de las tareas sean más rápidos. En esta sección se tratará de enfocarse únicamente en las memorias de almacenamiento primario. Las memorias de almacenamiento secundario ya fueron tratadas en la sección anterior

Las memorias de una computadora son componentes esenciales que almacenan y proporcionan acceso rápido a los datos y las instrucciones necesarias para el funcionamiento del sistema. Hay varios tipos de memorias en una computadora, cada una con diferentes características y usos específicos. Aquí hay una descripción general de algunos tipos comunes de memorias:

1. Memoria RAM (Random Access Memory): La RAM es la memoria principal de una computadora y se utiliza para almacenar datos y programas que se están utilizando activamente. Es una memoria volátil, lo que significa que su contenido se pierde cuando se apaga la computadora. La RAM proporciona un acceso rápido a los datos y permite que el procesador acceda a ellos de manera aleatoria Null and Lobur (2006a).

2. Memoria caché: La memoria caché es una memoria de alta velocidad que actúa como una capa intermedia entre la CPU y la memoria principal. Su objetivo es almacenar temporalmente los datos e instrucciones que se utilizan con mayor frecuencia, lo que permite un acceso más rápido a ellos y mejora el rendimiento general del sistema Null and Lobur (2006a).

3. Memoria ROM (Read-Only Memory): La ROM es una memoria de sólo lectura que contiene instrucciones y datos que no se pueden modificar. Al ser no volátil, su contenido se mantiene incluso cuando se apaga la computadora. La ROM almacena el firmware y programas de inicio (boot) esenciales para el arranque del sistema y la inicialización de hardware Null and Lobur (2006a).

En la figura 2.12 se puede apreciar la jerarquía de las memorias principales en una computadora. Donde el pico de la pirámide represente las de mayor costo y en la base se pueden apreciar las de menor.

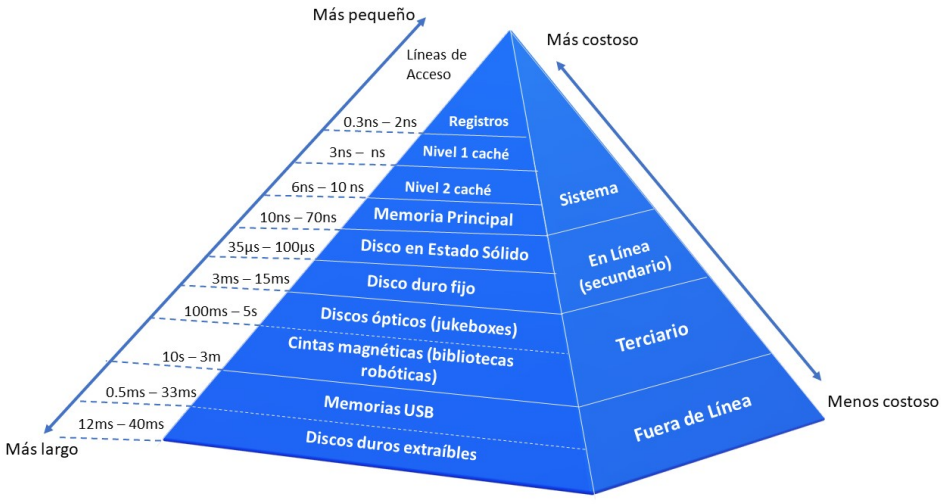

Figura 2.12: Jerarquía de Memoria en una Computadora. Fuente: Null and Lobur (2006b).

Hay dos tipos generales de chips utilizados en la construcción de la memoria RAM en las computadoras actuales: SRAM y DRAM. La DRAM utiliza condensadores que pierden carga y requiere recargas periódicas para mantener los datos. En cambio, la SRAM mantiene su contenido mientras haya energía disponible. Aunque la SRAM es más rápida y costosa, los desarrolladores sin embargo prefieren modelar utilizando la DRAM por su densidad, consumo de energía más bajo y generación de menos calor. Ambas tecnologías se utilizan en combinación, con DRAM para la memoria principal y SRAM para la memoria caché. Existen varias variantes de DRAM y SRAM, cada una con características específicas. Null and Lobur (2006a). Existen diferentes variantes de la DRAM, como la MDRAM, FPM DRAM, EDO DRAM, BEDO DRAM, SDRAM, SL DRAM, DDR SDRAM, RDRAM y DR DRAM. Por otro lado, los diferentes tipos de SRAM incluyen la SRAM asíncrona, SRAM síncrona y SRAM de ráfaga de tubería.

Dentro de el grupo de variantes de la DRAM se destaca principalmente la DDR SDRAM ya que en la últimas décadas ha tenido un uso extensivo en el mercado general de computadoras y servidores. Se introdujo como una mejora significativa sobre la memoria SDRAM original, ofreciendo un mayor rendimiento y velocidad de transferencia de datos A. Hema Singh (2012a).

La característica clave de la DDR SDRAM es su capacidad para transferir datos en ambos flancos de la señal de reloj, lo que permite duplicar la velocidad de transferencia en comparación con la SDRAM convencional. Esto significa que puede transferir datos en la parte ascendente y descendente del ciclo del reloj, lo que resulta en una mayor eficiencia y un mayor ancho de banda A. Hema Singh (2012a) .

La DDR SDRAM ha tenido un uso amplio en una variedad de sistemas y dispositivos electrónicos, como computadoras de escritorio, portátiles, servidores, consolas de juegos y dispositivos de red. Su capacidad para ofrecer un mayor rendimiento de memoria ha sido fundamental en el desarrollo de aplicaciones exigentes, como el procesamiento de gráficos, edición de video, juegos en línea y otras tareas intensivas de memoria A. Hema Singh (2012a).

La DDR SDRAM (DDR1) se introdujo por primera vez en 2000 como una mejora de la tecnología SDRAM convencional. Ofrecía una velocidad de transferencia de datos más rápida al permitir transferencias en ambos flancos del ciclo de reloj. La DDR1 fue ampliamente adoptada en computadoras de escritorio y portátiles durante ese período A. Hema Singh (2012a). Se lanzó con frecuencias de reloj que iban desde 100 MHz hasta 200 MHz. Utilizaba una señal de reloj de doble pulso, lo que permitía que los datos se transfirieran tanto en el flanco ascendente como en el descendente del ciclo de reloj. Esto proporcionaba una velocidad de transferencia de datos efectiva de hasta 400 MT/s (megatransferencias por segundo).

La DDR2 mejoró significativamente las velocidades de reloj en comparación con su predecesora. Comenzando desde 200 MHz, las frecuencias de reloj se incrementaron hasta 800 MHz y más allá. La DDR2 utilizaba un bus de datos más ancho y mejoras en la arquitectura interna para lograr velocidades de transferencia de hasta 1600 MT/s A. Hema Singh (2012a).

DDR3 SDRAM: La DDR3 llevó las frecuencias de reloj aún más alto, comenzando en el rango de 800 MHz y llegando hasta 2133 MHz y más allá. Las mejoras en la tecnología permitieron una mayor eficiencia y un menor consumo de energía en comparación con las generaciones anteriores. La DDR3 tenía una velocidad de transferencia efectiva de hasta 17000 MT/s A. Hema Singh (2012a). En el estudio comparativo de A. Hema Singh (2012a) se puede observar un análisis más detallado de la evolución de éstas tecnologías

de memoria. Donde hacen uso de comparaciones basándose en características como la densidad, velocidad, frecuencia máxima, entre otras. Para más información consultar la tabla 2.2

Características	DDR SDRAM	DDR2 SDRAM	DDR3 SDRAM	Ventajas DDR3 SDRAM
Voltaje	2.5 V	1.8 V	1.5 V	Reduce la demanda de energía del sistema de memoria DDR o DDR2 EN UN 17%.
Densidad	64MB a 1GB	256MB a 4GB	512MB a 8GB	Los componentes de alta densidad simplifican el sub – sistema de memoria.
Bancos Internos	4 (número fijo de filas y columnas)	4 y 8	8	Tiene una mayor relación de aciertos de página y un menor rendimiento máximo.
Entrelazado de bancos	----------	Permite el entrelazado de bancos	Permite el entrelazado de bancos	Es extremadamente efectiva para operaciones concurrentes y puede ocultar la sobrecarga de tiempo.
Pre – captura	2	4	8	Una velocidad más baja del núcleo de memoria resulta en una frecuencia más alta y una operación de menor consumo de energía.
Velocidad	100 a 200 MHz	200 a 533 MHz	300 a 1066 MHz	Mayor velocidad de transferencia de datos.
Frecuencia máxima	200 MHz o 400 Mbps por pin DQ	533 MHz o 1066 Mbps por pin DQ	1066 MHz o 2133 Mbps por pin DQ	Mayor velocidad de transferencia de datos.
Latencia de lectura	2, 2.5, 3, ciclos de reloj	3, 4, 5 ciclos de reloj	5, 6, 7, 8, 9,10 y 11	Eliminar el ajuste de medio ciclo de reloj permite la arquitectura de pre – captura 8n.
Latencia aditiva	----------	0, 1, 2, 3, 4	0, CL1 o CL2	Mejora la eficiencia de los comandos.

Tabla 2.2: Comparativa entre memorias DDR,DDR2,DDR3. Fuente: A. Hema Singh (2012b).

Sin embargo el crecimiento de la tecnología está en continuo ascenso, y en los último años ha incrementado las prestaciones de este tipo de tecnologías de memorias. Las memorias DDR4 y DDR5 son el tope actual en la fecha de escritura de este trabajo y sus características sorprenden por las enormes prestaciones que proveen con respecto a generaciones pasadas. En la tabla se muestra un análisis comparativo entre la memoria DDR4 y la DDR5 que se introdujo como estándar en julio del 2021 . Este estudio fue llevado a cabo por Rambus, una empresa diseñadora de chips que radica en Estados Unidos.

Características	DDR4	DDR5	Ventajas DDR5
Velocidad	1.6 a 3.2 GT/s 0.8 a 1.5 GHz ciclos de reloj	4.8 a 8.4 GT/s 1.6 a 4.2 GHz ciclos de reloj	Mayor ancho de banda.
Voltaje E/S	1.2 V	1.1 V	Menor consumo de energía.
Gestión de energía	En la placa base	En DIMM PMIC	Mejor eficiencia energética. Mejor escalabilidad.
Arquitectura de canal	Canal de datos de 72 – bit (64 datos + 8 ECC) 1 canal de datos por DIMM	Canal de datos de 40 – bit (32 de datos + 8 ECC). 2 canales de datos por DIMM	Mayor eficiencia de memoria. Menor latencia.
Longitud de bloque	BC4, BL8	BC8, BL16	Mayor eficiencia de memoria.
Densidad máxima de chip	16GB	64GB	Módulos DIMM de mayor capacidad.
Mayor inteligencia	SPD (PC)	SPD Hub y Temperatura de sensores (I°C)	Gestión mejorada del sistema. Mayor telemetría para la gestión térmica.

Tabla 2.3: Comparativa entre memorias DDR4 Y DDR5. Fuente: Press (2023).

2.9 Procesadores

Aunque en secciones anteriores se han ido tocando temas relacionados con las características de los procesadores, en esta sección se ampliarán los elementos importantes que están directamente relacionados con los objetivos de este trabajo. Se abordarán sobre los conceptos de núcleos del procesador, tipos, velocidades, etc.

La CPU (Central Unit of Processing), conocida como el "cerebro" de la computadora, se encarga de buscar instrucciones en la memoria y ejecutarlas. Cada CPU sigue un ciclo básico que implica buscar, decodificar y ejecutar dichas instrucciones sucesivamente hasta que el programa finaliza Tanenbaum and Bos (2014b).

Cada CPU tiene un conjunto de instrucciones específicas que puede ejecutar, lo que significa que no se pueden ejecutar programas diseñados para una arquitectura diferente. Por ejemplo un procesador x86 no puede ejecutar programas ARM y viceversa. Debido a que acceder a la memoria lleva más tiempo que ejecutar una instrucción, las CPUs incluyen registros internos para almacenar variables y resultados temporales. El conjunto de instrucciones también permite cargar y almacenar datos entre la memoria y los registros, así como realizar operaciones aritméticas y lógicas con los datos almacenados Tanenbaum and Bos (2014b).

Un núcleo de CPU es una unidad de procesamiento independiente dentro de un procesador. Cada núcleo de CPU tiene su propio conjunto de instrucciones y puede ejecutar tareas de manera independiente.

Un procesador puede tener uno o varios núcleos. Los procesadores de un solo núcleo solo pueden ejecutar una tarea a la vez, mientras que los procesadores de múltiples núcleos pueden ejecutar múltiples tareas simultáneamente, asignando una tarea a cada núcleo. Esto se conoce como paralelismo a nivel de hilos o multitarea en paralelo (ver sección 2.3).

Cada núcleo de CPU tiene su propia unidad de control, unidad aritmético-lógica y memoria caché. Esto permite que los núcleos de CPU realicen operaciones independientes, lo que mejora el rendimiento y la capacidad de respuesta del sistema Shen. P. John (2013).

En resumen, un núcleo de CPU es una unidad de procesamiento independiente dentro de un procesador que puede ejecutar tareas de manera simultánea y mejorar el rendimiento y la eficiencia de la computadora.

Los procesadores están compuestos por varios componentes esenciales, entre ellos:

1. Unidad de control: Es responsable de coordinar y controlar las operaciones del procesador. Se encarga de interpretar y ejecutar las instrucciones del programa Shen. P. John (2013).

2. Unidad aritmético-lógica (ALU): Es la parte encargada de realizar operaciones aritméticas y lógicas, como sumas, restas, multiplicaciones y comparaciones Tanenbaum and Bos (2014b).

3. Registros: Son pequeñas unidades de almacenamiento de alta velocidad que almacenan datos temporales durante las operaciones del procesador Shen. P. John (2013).

4. Caché: Es una memoria de acceso rápido que almacena datos e instrucciones utilizados con frecuencia para acelerar el acceso a la memoria principal Tanenbaum and Bos (2014b).

5. Unidad de punto flotante (FPU): Es responsable de realizar operaciones matemáticas más complejas, especialmente operaciones con números decimales Shen. P. John (2013).

Además de estos componentes esenciales, los procesadores pueden tener características adicionales, como múltiples núcleos (que permiten realizar varias tareas simultáneamente ver sección 2.3). tecnologías de virtualización (que permiten ejecutar múltiples sistemas operativos en una misma máquina ver sección 2.7) y tecnologías de gestión de energía para optimizar el consumo energético.

Cuando se va a hablar de arquitecturas de procesadores se tiene que hacer una distinción entre arquitecturas de diseño y arquitecturas de conjunto de instrucciones ISA (Arquitectura del Conjunto de Instrucciones).Las arquitecturas de diseño, son las que implementan los ingenieros para realizar la estructura interna, que determina cómo funcionan y se ejecutan las instrucciones en los procesadores.

A lo largo de los años, han surgido diferentes arquitecturas, con el objetivo de mejorar el rendimiento y la eficiencia de los microprocesadores. A continuación, se presenta una síntesis de algunas arquitecturas de diseños importantes:

1. Arquitectura de un solo ciclo: Es la más simple y básica, donde cada instrucción se ejecuta en un solo ciclo de reloj. Es eficiente en términos de velocidad, pero no optimiza el uso de recursos del procesador Tanenbaum and Bos (2014b).

2. Arquitectura de múltiples ciclos: Las instrucciones se dividen en varias etapas o ciclos de ejecución, lo que permite una mayor flexibilidad y eficiencia en el uso de los recursos. Cada ciclo realiza una parte específica de la instrucción, como la decodificación, la ejecución y el almacenamiento de resultados Tanenbaum and Bos (2014b).

3. Arquitectura superescalar: Utiliza múltiples procesadores para ejecutar múltiples instrucciones de forma simultánea, siempre que no existan dependencias entre ellas. Esto permite un mayor paralelismo y un mejor rendimiento general Tanenbaum and Bos (2014b) Shen. P. John (2013).

4. Arquitectura de canalización (pipeline): Divide las instrucciones en etapas y las procesa en paralelo, de manera que cada etapa se encarga de una parte específica de la instrucción. Esto permite que múltiples instrucciones se ejecuten en diferentes etapas del pipeline al mismo tiempo, mejorando así el rendimiento y la eficiencia (ver figura 2.13) Elahi (2018).

Por otro lado se encuentran las arquitecturas de tipo ISA

1. Arquitectura x86: Esta es la arquitectura predominante en las computadoras personales y servidores. Sus características incluyen:

   - Compatibilidad con una amplia gama de software y sistemas operativos.
   - Amplia disponibilidad de software y soporte de la industria.
   - Conjunto de instrucciones rico y completo.
   - Capacidad de procesamiento de datos de 32 y 64 bits.
   - Variedad de opciones de rendimiento y potencia.

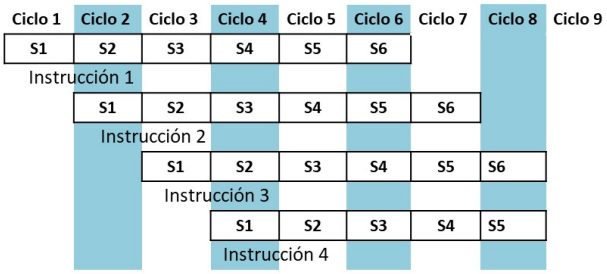

Figura 2.13: Instrucciones en Pipeline. Fuente: Null. Linda (2018).

2. Arquitectura ARM: Es ampliamente utilizada en dispositivos móviles, sistemas embebidos y dispositivos de bajo consumo energético. Sus características incluyen:
   - Eficiencia energética y baja potencia.
   - Diseño modular y escalable para adaptarse a diferentes requisitos.
   - Amplia adopción en sistemas operativos móviles, como Android e iOS.
   - Versatilidad y flexibilidad para aplicaciones de Internet de las cosas (IoT) y dispositivos embebidos.
3. Arquitectura RISC-V: Es una arquitectura de conjunto de instrucciones de código abierto que está ganando popularidad en la industria. Sus características incluyen:
   - Licencia abierta y libre de regalías, lo que permite la colaboración y personalización.
   - Modularidad y extensibilidad para adaptarse a diferentes requisitos y aplicaciones.
   - Diseño simple y limpio, lo que facilita la implementación y la depuración.
   - Soporte para diferentes configuraciones de procesadores, desde microcontroladores hasta servidores de alto rendimiento.

Estas tres arquitecturas son las más dominantes en la actualidad, pero también existen otras arquitecturas especializadas, como PowerPC, SPARC y MIPS, que se utilizan en aplicaciones específicas, como servidores de alto rendimiento, sistemas embebidos y redes. Cada arquitectura tiene sus propias fortalezas y se elige según los requisitos específicos de cada aplicación y dispositivo Tanenbaum and Bos (2014b) Null and Lobur (2006a) Jeffers et al. (2016) Hennessy and Patterson (2011).

La arquitectura x86 fue creada por en Intel con el lanzamiento del procesador Intel 8086 en 1978. Fue una evolución de la arquitectura de 8 bits utilizada en los primeros microprocesadores de Intel, como el Intel 8008 y el Intel 8080. El procesador Intel 8086 introdujo una arquitectura de 16 bits, lo que significaba que podía manejar palabras de

datos de 16 bits y direcciones de memoria de 16 bits. Esto permitió un mayor rendimiento y una mayor capacidad de direccionamiento en comparación con las arquitecturas de 8 bits anteriores Stakem (2012). La compañía de procesadores que tiene una competencia directa con Intel, AMD, también utiliza este tipo de arquitecturas en sus procesadores.

Cuando se va a evaluar un procesador en general se toman en cuenta distintas métricas como por ejemplo:

1. Frecuencia de reloj: La frecuencia de reloj, medida en gigahercios (GHz), indica la velocidad a la que el procesador ejecuta instrucciones. Una frecuencia de reloj más alta generalmente significa un rendimiento más rápido Null and Lobur (2006a).

2. Hyper-Threading [6] o Hilos: Algunos procesadores admiten la tecnología de hiperhilos (Hyper-Threading en Intel, Simultaneous Multithreading [7] en AMD), que permite que cada núcleo ejecute múltiples hilos simultáneamente. Esto puede mejorar el rendimiento en aplicaciones multitarea y paralelas.

3. Caché: La caché es una memoria de alta velocidad incorporada en el procesador que almacena datos y instrucciones utilizados con frecuencia. Una caché más grande y rápida puede reducir los tiempos de acceso a la memoria principal y mejorar el rendimiento 2.8.2.

4. TDP (Thermal Design Power): El TDP es la cantidad de energía térmica que el procesador disipa y se mide en vatios. Un TDP más bajo indica una mayor eficiencia energética y una generación de calor reducida Hennessy and Patterson (2011).

5. Benchmarks: Los benchmarks son pruebas estandarizadas que se utilizan para medir el rendimiento del procesador en diferentes tareas y escenarios. Algunos ejemplos de benchmarks populares son el rendimiento en juegos, pruebas de rendimiento de CPU y pruebas de codificación .

Existen numerosos estudios y análisis comparativos de tipos de procesadores. Por ejemplo en la tabla 2.4 se muestra un estudio hecho por una revista especializada en tecnología llamada AEC Corke (2021), que se encarga de emitir, entre otras cosas, reportes y comparaciones entre distintas marcas de procesadores. En su publicación del 2021 emitió una comparación entre un conjunto de procesadores de la marca Intel y AMD.

[6] Hyperthreading es una tecnología desarrollada por la compañía Intel que permite la ejecución de múltiples hilos virtuales por cada procesador, haciendo que las tareas se ejecuten más rápido.

[7] Es un diseño tecnológico que combina el multihilo con la tecnología de un procesador superescalar

	Amd Ryzen 5 5600X	Amd Ryzen 7 5800X	Amd Ryzen 9 5900X	Amd Ryzen 9 5950X	Intel Core i5 – 11600	Intel Core i5 – 11600K	Intel Core i7 – 11700	Intel Core i7 – 11700K	Intel Core i9 – 11900	Intel Core i9 – 11900K
# de núcleos de CPU	6	8	12	16	6	6	8	8	8	8
# de hilos de CPU	12	16	24	32	12	12	16	16	16	16
Frecuencia Base	3.70 GHz	3.80 GHz	3.7. GHz	3.40GHz	2.80 GHz	3.90 GHz	2.50 GHz	3.60 GHz	2.50 GHz	3.50 GHz
Frecuencia máxima de aumento	4.60 GHz	4.70 GHz	4.80 GHz	4.90 GHz	4.80 GHz	4.90 GHz	4.90 GHz	5.00 GHz	5.20 GHz	5.30 GHz
Caché	L2 caché 3MB L3 caché 32MB	L2 caché 4MB L3 caché 32MB	L2 caché 6MB L3 caché 64MB	L2 caché 8MB L3 caché 64MB	12MB Intel Smart Caché	12MB Intel Smart Caché	16MB Intel Smart Caché	16MB Intel Smart Caché	16MB Intel Smart Caché	16MB Intel Smart Caché
Por Defecto TDP	65 W	105 W	105 W	105 W	65 W	125 W	65 W	125 W	65 W	125 W
Precio (sin IVA)	£229	£317	£424	£574	£176	£192	£262	£292	£344	£441

Tabla 2.4: Comparativa procesadores. Fuente: Corke (2021).

2.10 Paralelismo

Existen principalmente dos tipos de paralelismo de manera general en las aplicaciones:

- Paralelización a nivel de tarea: Se basa en separar una tarea un varias sub-tareas independientes entre sí, pudiendo ejecutarse simultáneamente en diferentes núcleos, procesadores o nodos de ejecución en general Hennessy and Patterson (2011).
- Paralelización a nivel de datos: Se basa en la división de conjuntos de datos en pequeñas partes para procesarlas en paralelo. De la misma forma que las tareas, esas pequeñas partes de datos se van asignando a nodos de ejecución o núcleos para que sea procesadas Hennessy and Patterson (2011).

El hardware de una computadora puede aprovechar estos dos tipo de paralelismo de cuatro formas fundamentales:

- Paralelismo a nivel de instrucciones: A través de este tipo de paralelismo se trata de explotar el paralelismo a nivel de datos a un nivel modesto, intentando usar la ayuda del compilador y la idea del pipeline, además a nivel medio se usa la ejecución expeculativa [8]
- Las arquitecturas vectoriales y las unidades de procesamiento gráfico (GPUs): Se trata de explotar el paralelismo de datos aplicando una sola instrucción a una colección o conjunto de datos. Estas arquitecturas están optimizadas para realizar operaciones en conjuntos de datos más grandes mediante instrucciones SIMD (Single Instruction, Multiple Data). Las unidades de procesamiento gráfico (GPUs) son particularmente efectivas en tareas que pueden aprovechar la paralelización masiva,

[8]La ejecución especulativa es una técnica utilizada en el diseño de procesadores para mejorar el rendimiento mediante la anticipación y ejecución de instrucciones de manera especulativa, incluso antes de conocer los resultados de instrucciones anteriores. Una de las técnicas comunes utilizadas en la ejecución especulativa es la predicción de ramas (branch prediction), donde se intenta predecir la dirección de las ramas condicionales antes de conocer el resultado real de la condición

como el procesamiento de gráficos, la simulación física y el aprendizaje automático Hennessy and Patterson (2011).

- Paralelismo a nivel de hilo 2.3: Trata de explotar tanto el paralelismo de datos como el paralelismo de tareas en modelos de hardware que están estrechamente acoplados permitiendo la ejecución de hilos en paralelo. Esto se logra mediante la asignación de hilos a núcleos de procesador o a diferentes procesadores, lo que permite que los hilos se ejecuten de forma concurrente y se aproveche el paralelismo a nivel de hilo en aplicaciones que pueden dividirse en tareas independientes Hennessy and Patterson (2011).

- Paralelismo a nivel de solicitud: Explota el paralelismo de tareas desacopladas que se especifican tanto por el programador como por el sistema operativo. Es la capacidad de procesar múltiples solicitudes o tareas simultáneamente en un sistema. Esto se aplica a sistemas de memoria distribuida, redes y otros entornos donde múltiples solicitudes pueden procesarse en paralelo para mejorar el rendimiento y la eficiencia Hennessy and Patterson (2011).

Michael Flynn en 1966 Flynn and Luk (2011), econtró una simple clasificación con abreviaciones que es usado para definir los sistemas modernos. En su trabajo examinó las secuencias de instrucciones y los datos que se necesitan dentro de los componentes más restringidos de los multiprocesadores, clasificando todas las computadoras dentro de las cuatro categorías siguientes:

- Single instruction stream, single data stream (SISD): Se refiere a una arquitectura de computadora secuencial o uniprocesador convencional. En este tipo de arquitectura, una sola instrucción se ejecuta en un único procesador o núcleo, y opera en un único elemento de datos a la vez. Es el modelo más básico y secuencial de procesamiento Hennessy and Patterson (2011).

- SIMD (Single Instruction, Multiple Data): Se refiere a una arquitectura en la que una sola instrucción se ejecuta en múltiples procesadores o núcleos de forma simultánea, pero cada procesador opera en un conjunto de datos diferente. Todos los procesadores ejecutan la misma instrucción al mismo tiempo, pero con diferentes datos. Esto permite un alto grado de paralelismo en operaciones que pueden aplicarse de manera independiente a múltiples elementos de datos, como operaciones matriciales o de procesamiento de imágenes Hennessy and Patterson (2011).

- MISD (Multiple Instruction, Single Data): Es una categoría teórica que no ha sido ampliamente implementada en la práctica. En esta arquitectura, múltiples instrucciones se ejecutarían en paralelo en un solo dato. Sin embargo, hasta la fecha, no se ha construido ningún multiprocesador comercial de este tipo Hennessy and Patterson (2011).

- MIMD (Multiple Instruction, Multiple Data): Se refiere a una arquitectura en la que múltiples procesadores o núcleos ejecutan instrucciones independientes en diferentes conjuntos de datos de forma simultánea. Cada procesador tiene su propio flujo de instrucciones y su propio conjunto de datos, lo que permite una ejecución paralela de tareas independientes. Esta es la clase más común de arquitecturas paralelas y se utiliza en sistemas multiprocesador y en clústeres de computadoras Hennessy and Patterson (2011).

Aunque típicamente estas clasificaciones corresponden a un modelo general y en la práctica se usan combinaciones de SISD, SIMD y MIMD.

El conjunto de instrucciones SIMD es una de los más usados y populares en la actualidad. Dentro de este conjunto de instrucciones se encuentra la extensión SSE(Streaming SIMD Extensions), desarrollado por Intel para mejorar el rendimiento en operaciones vectoriales en procesadores x86 Hennessy and Patterson (2011).

Dentro de este conjunto de instrucciones se encuentra la extensión AVX (Advanced Vector Extensions). Las AVX son un conjunto de extensiones de instrucciones diseñadas por Intel (y también adoptadas por AMD) que amplían el tamaño de los registros vectoriales y agregan nuevas instrucciones para realizar operaciones en paralelo en un conjunto más amplio de datos. Las instrucciones AVX permiten aprovechar el paralelismo a nivel de datos mediante el uso de registros vectoriales más amplios, lo que resulta en un procesamiento más rápido de datos vectoriales. Las instrucciones SSE permiten realizar operaciones en paralelo en conjuntos de datos más grandes utilizando registros vectoriales de 128 bits. Estas instrucciones se utilizan principalmente en aplicaciones que realizan operaciones intensivas en datos, como procesamiento de imágenes, procesamiento de señales, simulaciones científicas y multimedia Hennessy and Patterson (2011).

Otra extensión muy popular son las AVX. Estas extensiones se introdujeron por primera vez con la familia de procesadores Intel Sandy Bridge en 2011. Las AVX permiten realizar operaciones en paralelo en registros vectoriales más amplios, lo que resulta en un mayor rendimiento en aplicaciones que pueden aprovechar el paralelismo a nivel de datos. Estas extensiones amplían el tamaño de los registros vectoriales de 128 bits a 256 bits (en la actualidad ya llegan a 512 bits), lo que permite realizar operaciones en un conjunto más grande de datos en una sola instrucción. Al utilizar instrucciones AVX, se pueden realizar operaciones simultáneas en múltiples elementos de datos, lo que acelera el procesamiento en aplicaciones que se benefician de la paralelización de datos, como el procesamiento de imágenes, el procesamiento de señales, la simulación física y más Hennessy and Patterson (2011).

2.10.1. Programación Híbrida Heterogénea

La programación híbrida heterogénea ha surgido como resultado de la evolución de los sistemas de computación paralela y distribuida, así como de los avances en la diversidad de dispositivos de procesamiento.

La programación híbrida heterogénea se refiere a la combinación de diferentes modelos de programación y dispositivos de procesamiento heterogéneos para abordar problemas de manera eficiente en sistemas paralelos o distribuidos.

El desarrollo de arquitecturas de procesadores especializados, como las GPUs (Unidades de Procesamiento Gráfico), FPGAs (Field-Programmable Gate Arrays) y otros aceleradores, ha brindado nuevas oportunidades para el paralelismo y la aceleración de aplicaciones. Estos dispositivos ofrecen capacidades de procesamiento masivo y paralelismo a nivel de datos que difieren de las CPU tradicionales Dongarra and Lastovetsky (2009).

Estas arquitecturas involucran el diseño y desarrollo de algoritmos y aplicaciones que pueden aprovechar al máximo los recursos disponibles en el sistema heterogéneo. Esto implica dividir la carga de trabajo en tareas que se pueden ejecutar en diferentes dispositivos y coordinar la comunicación y la sincronización entre ellos Dongarra and Lastovetsky (2009).

Este tipo de programación es especialmente útil en aplicaciones que requieren un alto rendimiento y eficiencia computacional, como simulaciones científicas, aprendizaje automático, procesamiento de imágenes y más. Al combinar los diferentes dispositivos de procesamiento de manera eficiente, se puede lograr un mayor rendimiento y aceleración en comparación con la programación en un solo tipo de dispositivo Dongarra and Lastovetsky (2009).

Si bien no hay un evento o lugar específico donde haya nacido, la programación híbrida heterogénea, se ha desarrollado y evolucionado a medida que la tecnología de procesamiento paralelo ha avanzado y se han utilizado diferentes modelos de programación para optimizar el rendimiento en sistemas heterogéneos.

Algunas de las tecnologías y herramientas comunes utilizadas en la programación híbrida heterogénea incluyen OpenMP, CUDA, OpenCL, MPI y DirectCompute, CUDA . Estas herramientas permiten a los desarrolladores aprovechar las capacidades de procesamiento paralelo de diferentes dispositivos y coordinar su uso de manera efectiva.

A continuación se muestra una breve descripción de algunas de éstas herramienta:

- CUDA (Compute Unified Device Architecture): CUDA es un modelo de programación desarrollado por NVIDIA para aprovechar el potencial de procesamiento de las GPUs (Unidades de Procesamiento Gráfico). Permite a los desarrolladores escribir código en lenguaje CUDA y ejecutarlo en las GPU compatibles. CUDA proporciona un entorno de programación paralela que permite el procesamiento masivo en paralelo y la aceleración de tareas computacionalmente intensivas Kirk (2007).

- OpenMP (Open Multi-Processing): OpenMP es un modelo de programación paralela de hilos compartidos utilizado para aprovechar el paralelismo en sistemas multiprocesador o multinúcleo. Permite a los desarrolladores escribir código en C, C++ o Fortran utilizando directivas de compilador y funciones específicas para distribuir tareas entre múltiples hilos. OpenMP es ampliamente utilizado para paralelizar bucles y secciones de código en aplicaciones que se benefician del procesamiento paralelo Mattson (2001).

- MPI (Message Passing Interface): MPI es una biblioteca de programación utilizada para implementar la comunicación y sincronización entre procesos en sistemas distribuidos o de memoria compartida. Proporciona funciones y directivas para enviar y recibir mensajes entre procesos, lo que permite la colaboración y el intercambio de datos en paralelo. MPI es comúnmente utilizado en aplicaciones científicas y de alto rendimiento que requieren comunicación entre nodos de un clúster o supercomputadora Nielsen (2016).

Cuando se utilizan este tipo de bibliotecas en un entorno de programación híbrida, generalmente se hacen las siguientes suposiciones:

- Suposición de hardware heterogéneo: Se asume que el sistema de computación tiene una combinación de dispositivos de procesamiento heterogéneos disponibles, como CPUs y GPUs. OpenMP se utiliza para aprovechar el paralelismo en la CPU, CUDA se utiliza para aprovechar el paralelismo en la GPU, y MPI se utiliza para la comunicación y sincronización entre nodos en un entorno distribuido.

- Suposición de paralelismo a nivel de datos: Se asume que las tareas o bucles del programa se pueden dividir en subprocesos independientes que se pueden ejecutar

simultáneamente en diferentes hilos o dispositivos de procesamiento. OpenMP se utiliza para paralelizar bucles en la CPU, CUDA se utiliza para paralelizar cálculos en la GPU y MPI se utiliza para distribuir tareas entre nodos.

- Suposición de comunicación y sincronización: Se asume que las tareas paralelas pueden requerir comunicación y sincronización entre hilos, dispositivos o nodos. MPI se utiliza para facilitar la comunicación entre nodos en un entorno distribuido, mientras que OpenMP y CUDA se utilizan para la sincronización y la comunicación entre hilos o dispositivos en el mismo nodo.

- Suposición de particionamiento de tareas: Se asume que las tareas del programa se pueden dividir en tareas más pequeñas que se pueden asignar a diferentes dispositivos o nodos para su procesamiento en paralelo. OpenMP se utiliza para particionar tareas en hilos de la CPU, CUDA se utiliza para particionar tareas en hilos de la GPU y MPI se utiliza para particionar tareas entre nodos.

Estas suposiciones son fundamentales para el diseño y la implementación de programas híbridos que utilizan combinaciones de OpenMP, CUDA y MPI. Sin embargo, es importante tener en cuenta que la implementación exacta y el rendimiento pueden variar según la aplicación y las características del hardware y el entorno específico.

Por ejemplo la multiplicación de matrices usando el algoritmo de Strassen es un algoritmo eficiente para la multiplicación de matrices que aprovecha el paralelismo y la estrategia de divide y vencerás. Este algoritmo puede ser utilizado en combinación con estrategias como OpenMP, CUDA y MPI para mejorar aún más su rendimiento. En la figura 2.14

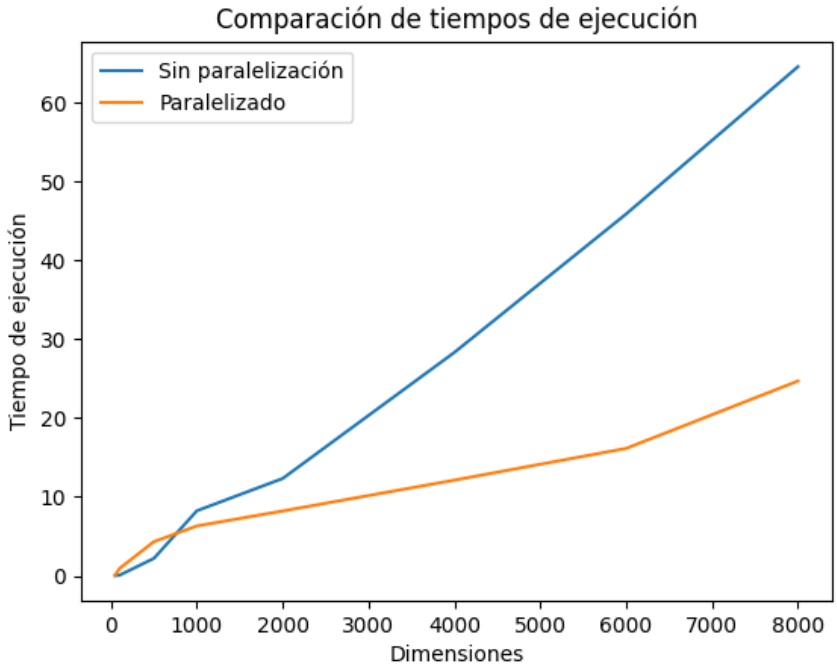

Figura 2.14: Gráfica comparativa de tiempos usando el algoritmo de Strassen sin estrategia de paralelización y usando la estrategia de paralelización con CUDA.

Capítulo 3

Sistemas Heterogéneos y Calendarizadores

En este capítulo se abordan algunos conceptos sobre la estructura de los sistemas heterogéneos y su relación con la calendarización. El objetivo es brindar una buena comprensión sobre algunos términos intrínsecos y nomenclaturas que se utilizan. Algunos de estos conceptos involucran el conocimiento de qué es un calendarizador, qué son los sistemas heterogéneos, qué tipos de calendarizadores existen, entre otros. Además se pretende enfocar al lector, en un marco específico de calendarizadores, los calendarizadores estáticos heterogéneos, y como este tipo de acercamiento provee soluciones interesantes para la optimización de indicadores tales como, consumo de energía, tiempo de completitud de calendarización, entre otros. Por otra parte se explica la complejidad del problema del problema de calendarización y algunos algoritmos utilizados para atacarlo.

3.1 Sistemas Heterogéneos

La calendarización en computadoras implica el análisis del sistema que va a procesar las tareas. Por lo que es necesario, para el diseño de un buen calendarizador, tener en cuenta el criterio de qué tipo de sistema es el que se tiene, si es homogéneo o heterogéneo. Los sistemas heterogéneos incluyen procesadores con diferentes arquitecturas, conjuntos de instrucciones y representaciones de datos. También incluyen elementos de procesamiento con diferentes características de rendimiento, como CPUs de propósito general, unidades de procesamiento gráfico (GPU) y Matriz de puerta programable en campo (FPGA) Dongarra et al. (2011). Por otra parte un sistema homogéneo consiste en procesadores o unidades de procesamiento que son idénticos en arquitectura y especificaciones Yang et al. (2019) . De manera general se podría decir que los sistemas de cómputo homogéneo están compuesto de recursos de cómputo idénticos (procesadores, memorias, etc.), mientras que un sistema de cómputo heterogéneo está compuesto de diferentes tipos de recursos, con capacidades de procesamiento y memoria distintas Lavaei et al. (2020).

En este estudio, se parte del supuesto de que el sistema en el que se realizará la calendarización es heterogéneo, basándose en características que poseen los entornos de trabajo de los participantes en este proyecto, además de investigaciones previas que indican que la mayoría de los sistemas de supercomputadoras que presentan características heterogéneas suelen tener un mejor rendimiento energético. Por otra parte, según un estudio de MarketsandMarkets (2018), se indica que la tendencia hacia la heterogeneidad está en aumento en el mercado de supercomputadoras.

Los sistemas heterogéneos suelen brindar un mejor rendimiento y eficiencia energética para aplicaciones que pueden aprovechar los recursos de hardware especializados, como las GPU o los FPGA Gao and Zhang (2016). La utilización de hardware especializado puede mejorar significativamente el rendimiento y la eficiencia energética de las aplicaciones en comparación con la utilización de hardware generalizado. Por tanto, debido a la naturaleza de las supercomputadoras que fueron confeccionadas para trabajar con aplicaciones altamente demandantes, se puede entender que la utilización de sistemas heterogéneos resulta ser una buena opción.

Por otra parte, los sistemas homogéneos presentan varias ventajas en cuanto a diferentes aspectos:

- Mayor simplicidad y facilidad de administración: Los sistemas homogéneos son más fáciles de administrar y mantener debido a su arquitectura uniforme con componentes idénticos o similares en todos los nodos de procesamiento.
- Compatibilidad y portabilidad de aplicaciones: Las aplicaciones desarrolladas para un nodo o procesador en un sistema homogéneo pueden ejecutarse sin problemas en otros nodos o procesadores del sistema, lo que facilita la compatibilidad y portabilidad.
- Menor latencia y mayor ancho de banda de la comunicación: En los sistemas homogéneos, la comunicación entre los nodos de procesamiento se puede lograr con una menor latencia y un mayor ancho de banda, ya que los componentes del sistema están optimizados para trabajar juntos.
- Mayor disponibilidad y confiabilidad: Los sistemas homogéneos tienden a ser más confiables y tener una mayor disponibilidad, ya que utilizan componentes conocidos y probados en el tiempo, reduciendo la posibilidad de fallas o incompatibilidades inesperadas.
- Menor complejidad de programación: La programación en sistemas homogéneos puede ser más sencilla en comparación con los sistemas heterogéneos, ya que no es necesario gestionar diferentes tipos de procesadores o aceleradores, lo que simplifica el desarrollo de software.

Sin embargo, no están excentos de problemas que los hacen falibles al momento de utilizarlos, como por ejemplo:

- Limitaciones de rendimiento en ciertos tipos de aplicaciones, que podrían beneficiarse de arquitecturas especializadas o aceleradores específicos.
- Menor flexibilidad en comparación con los sistemas heterogéneos, ya que todos los nodos de procesamiento son similares y carecen de la capacidad de adaptarse a requisitos específicos de carga de trabajo.
- Mayor consumo de energía en comparación con los sistemas heterogéneos, ya que no se pueden aprovechar completamente las eficiencias energéticas de dispositivos especializados.
- Posibilidad de menor escalabilidad, en comparación con los sistemas heterogéneos, ya que los componentes idénticos pueden limitar el potencial de crecimiento y expansión del sistema.

- Posibles limitaciones en la capacidad de procesamiento, en comparación con los sistemas heterogéneos que utilizan aceleradores especializados, ya que no se pueden aprovechar plenamente las capacidades de cómputo paralelo.

Existen varios ejemplos de supercomputadoras con sistemas homogéneos, por ejemplo:

- IBM Blue Gene: La serie Blue Gene de IBM ha sido conocida por su arquitectura homogénea y altamente escalable. Estos sistemas utilizan procesadores PowerPC y están diseñados para aplicaciones científicas y de simulación a gran escala.
- Cray XT: Los sistemas Cray XT (antes conocidos como Cray XT5 o Cray XE) también se consideran heterogéneo. Estos sistemas están basados en arquitecturas x86 de alto rendimiento, utilizando procesadores Intel Xeon en múltiples nodos interconectados. Aunque también se han configurado con diferentes tipos de nodos de procesamiento, incluyendo nodos con múltiples CPUs y nodos con GPUs.

En el artículo de Gao and Zhang (2016) se usan para el análisis los datos proporcionados por "TOP500" y "GREEN500" que son listas y clasificaciones que se utilizan para evaluar y comparar el rendimiento y la eficiencia energética de las supercomputadoras en todo el mundo.

La lista TOP500 es una clasificación que se publica dos veces al año y enumera las 500 supercomputadoras más potentes del mundo. Esta lista se basa en una prueba de referencia llamada Linpack, que mide el rendimiento en cálculos de punto flotante utilizando la biblioteca de álgebra lineal LINPACK. El ranking se basa en la capacidad de procesamiento en términos de operaciones de punto flotante por segundo (FLOPS). La lista TOP500 proporciona una visión general de las supercomputadoras más poderosas y se utiliza como un punto de referencia en el campo de la computación de alto rendimiento.

La lista GREEN500 es una clasificación que se publica junto con la lista TOP500 y se centra en la eficiencia energética de las supercomputadoras. Esta lista evalúa el rendimiento por vatio (FLOPS/Watt) para medir qué tan eficientemente una supercomputadora realiza cálculos en relación con la energía consumida. El objetivo de la lista GREEN500 es fomentar el desarrollo y la adopción de supercomputadoras más eficientes desde el punto de vista energético, promoviendo así la sostenibilidad y la reducción del impacto ambiental.

Además, usan la métrica de Linpack para analizar estos listados de datos y clasificaciones proporcionados por TOP500 y GREEN500.

El indicador Linpack efficiency, también conocido como eficiencia de Linpack, es una métrica utilizada en el campo de la computación de alto rendimiento para evaluar el rendimiento y la eficiencia de las supercomputadoras.

La eficiencia de Linpack se calcula dividiendo el rendimiento medido por el benchmark Linpack (medido en FLOPS u operaciones de punto flotante por segundo) por la capacidad teórica máxima de procesamiento de la supercomputadora. El resultado se expresa como un porcentaje Gao and Zhang (2016).

El benchmark Linpack es una prueba de referencia que mide el rendimiento en cálculos de álgebra lineal numérica, utilizando la biblioteca de álgebra lineal LINPACK. Este benchmark es ampliamente utilizado para evaluar y clasificar el rendimiento de las supercomputadoras en la lista TOP500 y TOP100.

La eficiencia de Linpack proporciona una medida de qué tan eficientemente una supercomputadora puede utilizar su capacidad de procesamiento en comparación con su

capacidad teórica máxima. Una eficiencia de Linpack alta (generalmente se considera deseable un valor cercano al 100 por ciento) indica que la supercomputadora está utilizando de manera eficiente su capacidad de procesamiento y obteniendo un rendimiento cercano a su capacidad máxima.

Es importante destacar que la eficiencia de Linpack es solo una métrica y no representa necesariamente el rendimiento real en aplicaciones específicas. Sin embargo, es una medida comúnmente utilizada para comparar el rendimiento relativo de las supercomputadoras y evaluar su eficiencia en términos de procesamiento de cálculos de punto flotante.

Gao and Zhang (2016) llegan a la conclusión de que los sistemas heterogéneos se caracterizan por ser altamente eficientes en términos de consumo de energía, aunque presentan una eficiencia relativamente baja en Linpack. Por otro lado, los sistemas homogéneos muestran una baja eficiencia energética, pero se destacan por su buen desempeño en Linpack. En segundo lugar, se observa que los sistemas heterogéneos con interconexión Ethernet tienen una baja eficiencia en Linpack, pero sobresalen en cuanto al consumo de energía. Por otro lado, los sistemas que utilizan InfiniBand logran un equilibrio favorable en ambas eficiencias. En tercer lugar, los procesadores Intel se destacan como los más eficientes en términos energéticos en la lista GREEN500. Por último, se han evidenciado mejoras generales en la eficiencia energética de Linpack.

Sin embargo la calendarización en sistemas heterogéneos presenta varios desafíos, debido a la presencia de múltiples tipos de recursos de hardware. Lo que complica la toma de decisiones óptimas sobre qué tarea se debe ejecutar en qué recurso y cuándo. Según Nikoletseas and Rolim (2013) la calendarización en sistemas heterogéneos es un desafío debido a la variabilidad de los recursos, las capacidades de los nodos, la necesidad de optimizar el rendimiento de la aplicación y la eficiencia energética. Por tanto los calendarizadores en sistemas heterogéneos resultan ser una opción que permite optimizar los indicadores mencionados anteriormente

3.2 Calendarizador

Un calendarizador, también conocido como planificador, es un componente crítico en sistemas de computación, que se encarga de programar y gestionar la utilización de los recursos de hardware y software disponibles para realizar tareas de manera eficiente.
Según Coulouris et al. (2011b) "Un calendarizador es un módulo que implementa políticas de planificación de procesos y les asigna recursos en función de estas políticas". En este contexto, los recursos pueden incluir CPU, memoria, dispositivos de entrada/salida, entre otros.

En Silberschatz et al. (2012) se define el calendarizador como "el módulo responsable de seleccionar, entre los procesos en ejecución, aquel que debería recibir el próximo tiempo de CPU". En este caso, el enfoque está en la asignación de tiempo de CPU en un sistema operativo.

Sin embargo, para este trabajo se necesita vincular directamente el concepto de calendarización dentro del marco de las supercomputadoras. Un servidor de computación de alto rendimiento (HPC) es un sistema de computación de alta capacidad diseñado para procesar grandes cantidades de datos y realizar cálculos complejos de manera rápida y eficiente. Éste, puede ser considerado también un servidor de supercomputadoras aunque tenga algunas diferencias con el enfoque tradicional.

En Sterling et al. (2018) se define el calendarizador como "el componente del sistema HPC, que decide cuáles son los trabajos más importantes que deben ejecutarse a

continuación, y en qué recurso de hardware deben ejecutarse". Aquí, el enfoque está en la asignación de recursos en un entorno de HPC, con el objetivo de lograr el máximo rendimiento y la eficiencia energética.

En Jeannot and Zilinskas (2019), se define el calendarizador en sistemas HPC como "el componente que gestiona la planificación de tareas y la asignación de recursos en función de los requisitos de rendimiento y la disponibilidad de los recursos, con el objetivo de maximizar el rendimiento y la utilización de los recursos". Aquí, el enfoque está en la gestión de la carga de trabajo en sistemas HPC, con el objetivo de lograr un rendimiento óptimo y una utilización eficiente de los recursos. Según Quinn (2004) "Los calendarizadores son esenciales en los sistemas HPC y se utilizan para asignar recursos de manera eficiente y reducir los tiempos de espera".

El problema de la calendarización ha existido desde hace mucho tiempo, ya que la necesidad de organizar y programar actividades ha sido una preocupación constante en diversas áreas. Sin embargo, en el ámbito de la investigación operativa y la optimización, el problema de calendarización ha sido objeto de estudio desde mediados del siglo XX Pinedo (2016).

El desarrollo de los primeros modelos y algoritmos para la calendarización se produjo en el contexto de la planificación de la producción en la industria manufacturera. A medida que las empresas buscaban mejorar la eficiencia y maximizar el uso de los recursos, surgieron desafíos relacionados con la asignación óptima de tareas y la programación de actividades en el tiempo Pinedo (2016).

A lo largo de los años, el problema de calendarización ha evolucionado y se ha adaptado a diferentes dominios y aplicaciones. Ha encontrado aplicaciones en áreas como la programación de proyectos, la gestión de inventarios, la planificación de horarios de transporte, la asignación de personal, entre otros.

Con el avance de la tecnología y la creciente complejidad de los sistemas, el problema de calendarización se ha vuelto más desafiante. Se han desarrollado modelos y algoritmos más sofisticados para abordar aspectos como la incertidumbre, los plazos ajustados, la asignación de recursos heterogéneos y la optimización de múltiples objetivos.

3.3 Calendarizadores Estáticos Heterogéneos

Existen dos grandes grupos de calendarización, calendarizadores estáticos y calendarizadores dinámicos. Según Lavaei et al. (2018), los calendarizadores estáticos son aquellos que asignan las tareas a los recursos antes de la ejecución de las mismas, mientras que los calendarizadores dinámicos toman decisiones de planificación en tiempo de ejecución en función del estado actual del sistema, la carga de trabajo y las prioridades de las tareas.

Ambos enfoques de calendarización tienen sus ventajas, en Buyya et al. (2013) se expone que el enfoque estático suele ser mejor para cargas de trabajo de alta demanda y para entornos en los que el hardware subyacente no cambia con frecuencia. En Sotiriades et al. (2015) se señala que un calendarizador estático permite un control más preciso sobre los recursos y el tiempo de ejecución y en Dastjerdi et al. (2018) se dice que la calendarización estática es más adecuada para aplicaciones con requisitos de tiempo real y para cargas de trabajo con una alta carga de procesamiento.

Por otra parte Sotiriades et al. (2015) menciona que un calendarizador dinámico puede adaptarse mejor a los cambios en las condiciones de la carga de trabajo y del hardware, lo que permite un mejor rendimiento y una mejor utilización de los recursos, Al-Khateeb et al. (2018) argumenta que la calendarización dinámica puede mejorar el rendimiento

y la eficiencia energética en cargas de trabajo variables y en entornos de hardware heterogéneos. Según Al-Khateeb et al. (2018) la calendarización estática no es capaz de adaptarse a los cambios en la carga de trabajo o en las características de los recursos, lo que puede conducir a un uso ineficiente de los recursos y una falta de equilibrio en la carga de trabajo. En Sotiriades et al. (2015) se evidencia que la calendarización estática puede no ser capaz de adaptarse a los requisitos de tiempo real de las aplicaciones y puede conducir a la violación de los plazos y a la pérdida de calidad de servicio. En Al-Khateeb et al. (2018) también se expone que la calendarización dinámica puede introducir un mayor costo computacional debido a la necesidad de tomar decisiones en tiempo real basadas en la información actualizada, en Liu et al. (2019) se argumenta que la calendarización dinámica puede ser susceptible a la sobrecarga de comunicación y puede generar cuellos de botella en la red.

Calendarizadores	Ventajas	Desventajas
Estáticos	Mejor control sobre los recursos y el tiempo de ejecución	No puede adaptarse a los cambios en la carga de trabajo o en las características de los recursos.
	Adecuado para aplicaciones con requisitos de tiempo real y alta carga de procesamiento.	Puede conducir a la violación de los plazos y pérdida de calidad de servicio.
	Más preciso en la asignación de recursos.	No puede ajustarse a las demandas cambiantes de la carga de trabajo.
Dinámico	Capacidad de adaptarse a los cambios en las condiciones de la carga de trabajo y del hardware.	Mayor costo computacional debido a la toma de decisiones en tiempo real.
	Mejor rendimiento y utilización de los recursos en entornos variables.	Puede generar sobrecarga de comunicación y cuellos de botella en la red.
	Adecuado para cargas de trabajo variables y entornos de hardware heterogéneos.	Puede haber falta de equilibrio en la carga de trabajo.

Tabla 3.1: Tabla comparativa de calendarizadores estáticos y dinámicos

En la tabla 3.1, se condensan las ventajas y desventajas descritas anteriormente para cada uno de estos enfoques de calendarización.

La elección de un calendarizador estático o dinámico en servidores HPC depende de varios factores, como la naturaleza de las aplicaciones y las tareas que se ejecutarán, las características del hardware disponible y los requisitos de tiempo real de la aplicación. En general, los calendarizadores estáticos se utilizan con mayor frecuencia en aplicaciones con requisitos de tiempo real y en tareas críticas que necesitan un tiempo de respuesta predecible, mientras que los calendarizadores dinámicos se utilizan con mayor frecuencia en aplicaciones que tienen requisitos de tiempo menos restrictivos y que pueden tolerar cierta variabilidad en el tiempo de respuesta.

Hay algunos estudios que han examinado el uso de calendarizadores estáticos y dinámicos en supercomputadoras, pero no hay un consenso claro sobre cuál es el enfoque más utilizado.

Un estudio publicado en Soft Computing: Theories and Applications (SoCTA) en 2018 observó que los algoritmos dinámicos, EDF (Earliest Deadline First) y LST (Least Slack Time First), funcionan de manera eficiente en situaciones de baja carga (underload), pero no logran un rendimiento óptimo en situaciones de alta carga (overload). Por otro

lado, los algoritmos estáticos, RM (Rate Monotonic) y SJF (Shortest Job First), no son capaces de planificar de manera efectiva un proceso específico en situaciones de baja carga, pero muestran un mejor desempeño en situaciones de alta carga en comparación con los algoritmos dinámicos Teraiya et al. (2020).

El estudio se llevó a cabo utilizando un conjunto de datos amplio que consta de más de 7000 conjuntos de Trabajos, cada uno de ellos compuesto por uno a nueve tareas. La carga varió entre 0.5 y 5. Se realizaron pruebas durante 500 unidades de tiempo para asegurar la precisión de los resultados obtenidos.

En resumen, este estudio comparó los algoritmos de calendarización dinámica y estática en un RTOS (Real-Time Operating System) para aplicaciones de tiempo real suaves, evaluando su tasa de éxito y la utilización efectiva de la CPU en diferentes escenarios de carga. Los algoritmos dinámicos mostraron un buen rendimiento en situaciones de baja carga, mientras que los algoritmos estáticos fueron más efectivos en situaciones de alta carga.

Por otra parte en el estudio de Oki et al. (2021) se compara el rendimiento de los esquemas de calendarización dinámica y estática del compilador OSCAR en un programa de control de motor real ejecutado en un procesador multicore embebido en un FPGA. Para sistemas embebidos en tiempo real, como los programas de control de motores de automóviles, la calendarización dinámica se considera la única opción viable debido a las ramas condicionales, tareas de granularidad pequeña y estructuras de programa con pocos bucles. Sin embargo, mediante el uso de métodos de retroalimentación del perfil de ejecución, la calendarización estática también puede ser efectiva en estos sistemas. La evaluación se realizó en el procesador multicore RH850. Los resultados mostraron que la calendarización estática logró mejoras significativas en la velocidad de ejecución en comparación con la ejecución secuencial, mientras que la calendarización dinámica obtuvo mejoras más pequeñas. Esto indica que la calendarización estática, junto con los métodos de retroalimentación del perfil de ejecución, es una opción eficaz para reducir los tiempos de ejecución en sistemas de control embebidos en tiempo real con procesadores multicore Oki et al. (2021).

En resumen, no hay un enfoque universalmente aceptado para la elección de un calendarizador en servidores HPC, y se debe evaluar cuidadosamente en función de las necesidades específicas de la aplicación y los recursos disponibles. Sin embargo para este trabajo se decidió escoger la calendarización estática heterogénea como enfoque a trabajar.

3.3.1. Elementos a considerar en la calendarización

En el libro de Pinedo (2016) se exponen conceptos importantes referidos a la calendarización en general muy interesantes. Sin embargo su marco de estudio abarca definiciones usadas en el entorno de la calendarización en la industria manufactera y servicios entre otras. Elementos sobre los cuales no se concentra este trabajo, por lo que no resulta muy conveniente incluirlos.

Dada la extensividad de los distintos escenarios y restricciones que se pueden encontrar en la literatura, se va a tratar de enfocar solamente en una parte del problema. Por eso resulta más conveniente la definición que da Topcuoglu et al. (2002a) en donde se expone un marco de trabajo específico y que se ajusta bien al entorno de definiciones de éste trabajo.

De manera más amplia y genérica en el libro de Drozdowski (2009) se exponen algunos conceptos pero enfocados en el área de computación, a diferencia del libro de Pinedo

(2016), aunque sus definiciones y escenarios exceden el alcance de éste proyecto.

Sin embargo se tratarán de vincular los dos enfoques, encontrando puntos en común y mencionando ciertos elementos que resultan importante conocer.

Notaciones importantes:

Para un problema de calendarización estática se supone, donde conocen de antemano, las características del trabajo que va a ser descompuesto en tareas para que cada una sea calendarizada. Entonces, es posible conocer sus tiempos de ejecución, en las distintas unidades de procesamiento que componen al sistema heterogéneo. Aunque esto resulta ser un tema controversial, puesto que tampoco existe un consenso general de como determinar o predecir esos tiempos de ejecución. Sin embargo, esos temas se van a discutir en capítulos posteriores.

Sea $P = P_1, P_2, ..., P_m$ el conjunto de unidades de procesamiento. En la teoría clásica, se distinguen dos tipos de entornos de procesamiento: paralelo y dedicado. En general, las unidades de procesamiento paralelas son capaces de ejecutar cualquier tarea. Las unidades de procesamiento paralelas tienen tres subtipos: idénticos ($P_i = P_j$ para todo i, j). Los uniformes (con velocidades diferentes b_i), lo que significa que el tiempo de ejecución de la tarea que se ejecute en esas unidades dependerá únicamente de la velocidad b_i. Los no relacionados son tipos de unidades que no guardan relación unas con otras, por lo que los tiempos de ejecución de las tarea dependerán en gran medida de las características de los dispositivos en donde se ejecuten. Para nuestra definición se va a asumir, como estamos abordando un entorno heterogéneo, que el conjunto de unidades de procesamiento son paralelos de tipo no relacionado.

Restricciones de precedencia: Para algún conjunto T de tareas que consta de las tareas $T = T_1, T_2, ..., T_n$ si alguna tarea T_i debe completarse antes de que otra tarea T_j pueda comenzar, lo cual se denota como $T_i \rightarrow T_j$, se dice que existen restricciones de precedencia en el conjunto T. Las precedencias se representan como un grafo dirigido acíclico (DAG) llamado grafo de tareas o grafo de precedencia de tareas. Por lo general, las tareas son los nodos del grafo y las aristas son las precedencias. Así, un grafo de tareas se representa como un par (T, A), donde A es el conjunto de restricciones de precedencia Drozdowski (2009). En la figura 3.1 se puede visualizar un grafo DAG compuesta por cinco tareas con precedencias entre ellas.

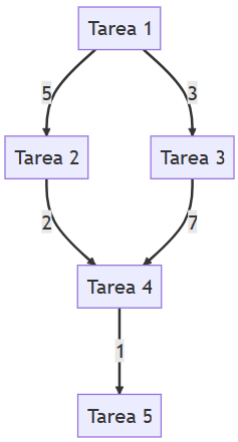

Figura 3.1: Grafo DAG compuesto por cinco tareas

Sea R_{ij} la cantidad de datos necesarios para que se ejecute la Tarea T_i que son entregados por la Tarea T_j. Esto se entiende a través de las precedencias de ejecución, ya que,

para que una tarea se ejecute tiene que haberse ejecutado su conjunto de predecesoras y además haber mandado todos los datos las predecesoras hasta la tarea T_i. En ésta representación no se especifica en qué unidades se van a trabajar los datos de transferencia entre tareas.

Sea RT_{ij} la tasa de transferencia entre unidades de procesamiento. Esto se entiende como la tasa de transferencia para enviar datos desde un P_i hasta un procesador P_j. Se podría representar como una matriz de $P \times P$. La métrica de la tasa de transferencia es expresada en *datos/segundos*, y la métrica de datos es tomada a partir de qué métrica se utilizó para representar los datos necesarios que necesita una Tarea T_i para ejecutarse, proporcionados por una tarea T_j. Por ejemplo si esos datos se representan en bytes, la tasa de transferencia sería *bytes/segundos*.

Entonces el costo de comunicación de la arista (i, j), que corresponde a la transferencia de datos desde la tarea T_i (programada en P_m) a la tarea T_j (programada en P_n), se define como:

$$C_{ij} = \frac{\text{datos}(T_i, T_j)}{\text{tasa}(P_m, P_n)}$$

Donde:

- C_{ij} es el costo de comunicación entre las tareas T_i y T_j.
- datos(T_i, T_j) es la cantidad de datos necesarios para que se ejecute la Tarea T_i que son entregados por la Tarea T_j.
- tasa(P_m, P_n) es la tasa de transferencia para enviar datos desde un procesador P_m hasta un procesador P_n Drozdowski (2009).

Los costos de ejecución de la Tarea T_i en el procesador P_n en Topcuoglu et al. (2002a) son representados como W_{ij} y para obtener los costos de ejecución W_{ij}, no se consideró un entorno real. El conjunto de unidades de procesamiento que se consideraron fueron compuestos en base a la aleatoriedad de distintos parámetros.

Por ejemplo, para asignar velocidades diferentes a las distintas unidades de procesamiento, los pesos de potencia de cómputo de cada unidad se establecieron aleatoriamente dentro de un rango dado. El peso de potencia de cómputo de cada procesador P_j, ϕ^i, es un número que muestra la relación entre la velocidad de la unidad de procesamiento de P_j y la velocidad de la unidad más rápido del sistema.

Para establecer el costo de cálculo de cada tarea en los procesadores, se establece el costo de cálculo del nodo en un procesador base (P_{base}) de forma analítica o aleatoria. Luego, se calcula el costo de cálculo para cada otro procesador P_i mediante

$$W(T_j, P_k) = \frac{\phi^{\text{base}}}{\phi^k} \cdot W(T_j, P_{\text{base}})$$

Donde:

$W(T_j, P_k)$ es el costo de cálculo de la tarea T_j en el procesador P_k.

ϕ^{base} es el peso de potencia de cómputo del procesador base P_{base}, que muestra la relación entre la velocidad de la unidad de procesamiento de P_{base} y la velocidad de la unidad más rápida del sistema.

ϕ^k es el peso de potencia de cómputo del procesador P_k, que muestra la relación entre la velocidad de la unidad de procesamiento de P_k y la velocidad de la unidad más rápida del sistema.

$W(T_j, P_{\text{base}})$ es el costo de cálculo de la tarea T_j en el procesador base P_{base}.
Por lo tanto, la fórmula calcula el costo de cálculo de una tarea en un procesador específico basándose en el costo de cálculo de la tarea en el procesador base y la relación de las velocidades de las unidades de procesamiento.

Esta forma de calcular el costo de cómputo de una tarea en un procesador resulta ser efectiva de manera experimental, sin embargo, en la práctica es mucho más complejo el análisis de los costos de cómputo, ya que la forma mencionada anteriormente no toma en cuenta las características de las tareas y las unidades de procesamiento en donde se van a ejecutar dichas tareas. En capítulos posteriores se tratará diseñar una forma eficiente que sirva para predecir dichos costos de ejecución.

El $EST(m, pj)$ (Tiempo Más Temprano de Inicio de Ejecución) y $EFT(n, pj)$ (Tiempo Más Temprano de Finalización de Ejecución) representan los tiempos más tempranos posibles de inicio y finalización de un nodo n en el procesador P_j, respectivamente. Estos valores se determinan en base al algoritmo de calendarización que se utilice y a las dependencias entre nodos.

El $EST(T_m, P_j)$ es el tiempo más temprano en el cual la tarea T_m puede comenzar su ejecución en el procesador pj considerando las restricciones de precedencia y la disponibilidad de recursos. Toma en cuenta los tiempos de finalización de las tareas predecesoras y la disponibilidad de los recursos requeridos por la tarea T_m.

El $EFT(n, pj)$ es el tiempo más temprano en el cual se espera que el nodo n complete su ejecución en el procesador pj. Se calcula sumando el tiempo de ejecución de la tarea T_i a su $EST(m, pj)$.

Dependiendo del tipo de algoritmo que se esté utilizando para resolver el problema de calendarizar, se utilizan en muchas ocasiones funciones para establecer las prioridades. Por ejemplo en Topcuoglu et al. (2002a) se propone una heurística para resolver este tipo de problema y ellos definen una función de prioridad que se calcula recorriendo el camino crítico en un grafo DAG, y en su caso toman en cuenta los costos de comunicaciones y los costos de ejecución. Sin embargo no necesariamente debería ser un criterio para establecer una prioridad, ya que se podrían tomar en cuenta otros factores como los costos energéticos que producen la ejecución de una tarea en procesador determinado, entre otros, siempre dependiendo del tipo de problema en el entorno específico que se quiera resolver.

Entonces la función objetivo de la calendarización es encontrar una asignación de tareas de un Trabajo 2.4 a unidades de procesamiento o recursos de manera que se minimice la duración total de la ejecución o makespan [1] cumpliéndose todas las condiciones de precedencia Topcuoglu et al. (2002a), del Trabajo .

3.4 ¿Por qué es tan difícil la calendarización?

La calendarización ha sido tradicionalmente considerada como un problema de optimización, en el cual se busca encontrar la mejor asignación de tareas a los recursos disponibles con el objetivo de optimizar algún criterio de desempeño Pinedo (2016). Por esta razón, muchos de los primeros enfoques para resolver problemas de calendarización se basaron en técnicas de optimización matemática, como programación lineal, programación entera y programación dinámica Leung et al. (2000).

[1] Es un término comúnmente utilizado en la calendarización que hace referencia a la longitud total o el tiempo de finalización más tardío de un conjunto de tareas o procesos

La complejidad matemática de la calendarización se puede entender a través de su formulación matemática, en la programación lineal, la calendarización se puede representar como un problema de asignación de recursos sujeto a restricciones lineales como se muestra a continuación:

$$\text{mín} \sum_{i=1}^{n} \sum_{j=1}^{m} t_{ij} x_{ij}$$

Sujeto a:

$$\begin{aligned}
&\sum_{j=1}^{m} x_{ij} = 1 && \forall i \in \{1, \ldots, n\} \\
&\sum_{i=1}^{n} x_{ij} \leq c_j && \forall j \in \{1, \ldots, m\} \\
&x_{ij} \in \{0, 1\} && \forall i \in \{1, \ldots, n\}, j \in \{1, \ldots, m\}
\end{aligned}$$

Donde:

n es el número de tareas a calendarizar. m es el número de recursos disponibles. t_{ij} es el tiempo de procesamiento de la tarea i en el recurso j. x_{ij} es una variable binaria que indica si la tarea i se asigna al recurso j ($x_{ij} = 1$) o no ($x_{ij} = 0$). El primer conjunto de restricciones garantiza que cada tarea se asigna a exactamente un recurso. El segundo conjunto de restricciones garantiza que cada recurso no se sobrecarga más allá de su capacidad máxima c_j. Las variables binarias x_{ij} permiten modelar la asignación de tareas a recursos.

Sin embargo, incluso en este caso simplificado, el problema sigue siendo NP-difícil y no tiene solución en tiempo polinómico. Además, la programación lineal no es capaz de modelar las restricciones no lineales, como la disponibilidad variable de recursos y los requisitos de tiempo real.

En el caso de la programación dinámica, aunque es más general que la programación lineal, su complejidad también aumenta exponencialmente con el tamaño del problema Nemhauser (1967). Sin embargo, la programación dinámica no es adecuada para tratar con recursos heterogéneos y la necesidad de equilibrar la carga de trabajo entre ellos. Se sabe que la programación dinámica puede resolver problemas de optimización de manera eficiente si el número de subproblemas es polinomial en el tamaño de la entrada. Sin embargo, para muchos problemas de calendarización, el número de subproblemas crece exponencialmente con el tamaño de la entrada, lo que resulta en una complejidad exponencial Drozdowski (2009).

Un ejemplo clásico para demostrar que la calendarización es un problema NP (No Determinista Polinómico) es el Problema del Flujo de Tiempo (Job Shop Scheduling Problem). En este problema, se tienen n trabajos y m máquinas, y cada trabajo debe ser procesado en un orden específico en diferentes máquinas, cumpliendo con ciertas restricciones de precedencia.

La tarea consiste en encontrar una secuencia de calendarización que minimice el *makespan*, es decir, el tiempo total requerido para completar todos los trabajos. Este problema es conocido por ser NP-duro, lo que implica que no se ha encontrado un algoritmo eficiente que pueda resolverlo en tiempo polinómico para todos los casos.

Supongamos que tenemos n tareas y m máquinas. Para cada tarea, hay m posibles máquinas en las que se puede procesar. Por lo tanto, el número total de posibles asignaciones para la primera tarea es m.

Ahora, para la segunda tarea, nuevamente hay m posibles máquinas disponibles. Sin embargo, esta vez debemos considerar todas las posibles asignaciones para la primera tarea. Por lo tanto, el número total de posibles asignaciones para la segunda tarea es m^2.

Continuando este proceso para las n tareas, cada vez multiplicando por m, obtendremos el número total de posibles secuencias de calendarización:

$$m \cdot m^2 \cdot m^3 \cdot \ldots \cdot m^n$$

Simplificando esta expresión, obtenemos:

$$m^{1+2+3+\ldots+n}$$

La suma de los números naturales del 1 al n se puede expresar como $n \cdot (n+1)/2$. Por lo tanto, la expresión se convierte en:

$$m^{n \cdot (n+1)/2}$$

Podemos ver que esta expresión crece exponencialmente con el número de tareas y máquinas. A medida que n y m aumentan, el número total de posibles secuencias de calendarización se vuelve cada vez más grande, lo que demuestra que el problema del flujo de tiempo tiene una complejidad exponencial.

La complejidad NP se debe a que el número de posibles secuencias de calendarización crece exponencialmente con el número de tareas y máquinas. Para encontrar la secuencia óptima que minimice el *makespan*, es necesario explorar todas las posibles combinaciones, lo que requiere un tiempo de ejecución exponencial.

Por lo tanto, la calendarización es considerada un problema NP, lo que implica que no se ha encontrado un algoritmo eficiente que pueda resolverlo en tiempo polinómico para todos los casos.

Sin embargo, en la práctica existen otras cuestiones a tomar en cuenta que pueden aumentar o disminuir el espacio de búsqueda, como la restricciones de ejecución de las tareas, las disponibilidades de los recursos, o la cantidad de objetivos que se intenten minimizar o maximizar.

Por otra parte la calendarización en servidores HPC (High-Performance Computing) y centros de datos es un problema complejo debido a la gran cantidad de tareas que deben ejecutarse y a los recursos limitados disponibles.

La calendarización en supercomputadoras implica la asignación de recursos, como CPU, memoria y almacenamiento, a las tareas para lograr los objetivos de rendimiento, tiempo de respuesta, energía. Además, también deben tenerse en cuenta las restricciones de los recursos, como los recursos compartidos, las prioridades y las dependencias entre tareas Topcuoglu et al. (2002b).

La complejidad aumenta más cuando se consideran sistemas con múltiples usuarios y múltiples objetivos, como la maximización del rendimiento, la minimización del tiempo de respuesta y la maximización del uso de recursos. La calendarización supercomputadoras también puede implicar la optimización de múltiples objetivos simultáneamente, como la maximización del rendimiento y la minimización del costo energético Beloglazov et al. (2012). Por tanto se hace difícil encontrar una solución óptima manualmente en un tiempo razonable. Por lo que es necesario encontrar métodos eficientes que puedan proporcionar

soluciones precisas y óptimas en un tiempo razonable. La búsqueda de enfoques efectivos para la calendarización ha impulsado la investigación y desarrollo de diversas estrategias, tales como algoritmos heurísticos y técnicas de optimización. En Sukhpal Singh (2016) presentan una revisión exhaustiva de la asignación de recursos y la calendarización en la computación en la nube. Discuten los métodos tradicionales, como la programación lineal y la programación de redes, que pueden ser eficientes para proyectos pequeños y simples. Sin embargo, estos métodos pueden no ser lo suficientemente eficientes para proyectos más grandes y complejos como se pudo ver anteriormente. Los métodos heurísticos, como los Algoritmos Genéticos, los algoritmos basados en reglas y método de inteligencia artificial como el Aprendizaje Automático pueden proporcionar soluciones rápidas y cercanas a lo óptimo, pero no garantizan una solución óptima en todos los casos.

Capítulo 4

Algoritmos en la Calendarización Estática Heterogénea

De manera general se pueden visualizar en la bibliografía dos grandes grupos de algoritmos en la calendarización estática, el primero basado en heurísticas y el segundo basado en búsquedas aleatorias guiadas, como se muestra en la Figura 4.1

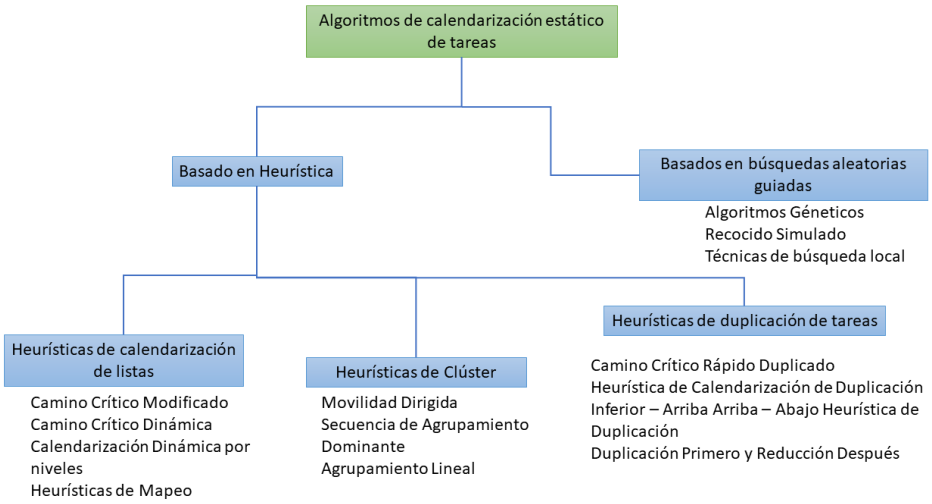

Figura 4.1: Algoritmos de calendarización estática. Fuente: Topcuoglu et al. (2002c).

En el artículo de Topcuoglu et al. (2002a) se menciona que existen dentro de los algoritmos basados en heurísticas tres grupos más, el primero heurísticas de calendarización de listas o heurísticas de programación de listas, el segundo heurísticas de agrupamiento, y el tercero heurísticas de duplicación de tareas. Además se menciona que la mayoría de las heurísticas que componen a estos grupos fueron pensadas para resolver el problema de la calendarización estática con la consideración de que los recursos computacionales fueran homogéneos.

- **Heurísticas de calendarización de listas**: Según la revisión que hace Topcuoglu et al. (2002a) Una calendarización de listas mantiene una lista de todas las tareas de un grafo según sus prioridades. Esta heurística posee dos fases, la primera es la priorización de tareas o selección en donde se escogen las tareas que estén listas con mayor prioridad, y la segunda es la de selección del nodo o procesador, en donde se

escoge el procesador más adecuado que minimice una función de costo predefinida con anterioridad. Esta función de costo puede ser el tiempo de ejecución inicial. Existen varios ejemplos de este tipo de heurísticas en la bibliografía, como son el Camino Crítico Modificado (MCP)Wu and Gajski (1990),Heurísticas de Mapeo El-Rewini and Lewis (1990),Heurísticas de Inserción Lewis and Kruatrachue (1988). Este tipo de heurísticas según Topcuoglu et al. (2002b) tienen muy buen resultado para nodos homogéneos, además aclara que se suelen obtener mejores resultados que en los demás grupos. Para un análisis más técnico y una definición más detallada del uso de este tipo de heurísticas se puede consultar el artículo de Schutten (1996).

- **Heuristicas de Clúster**: Según el artículo de Topcuoglu et al. (2002a) se dice que un algoritmo que pertenece a este grupo mapea las tareas de un grafo dado a un número no limitado de agrupaciones. En cada paso las tareas seleccionadas para agrupar pueden ser cualquier tipo de tareas, no necesariamente las tareas que estén listas para ser ejecutadas. En cada iteración del algoritmo se refinan las agrupaciones anteriores mezclando con otras. Si dos tareas son asignadas al mismo agrupamiento entonces serán asignadas al mismo procesador o nodo. Se prosigue con una serie de pasos adicionales para lograr la calendarización final, primeramente se hace una mezcla sucesiva de los grupos para lograr igualar éstos con la cantidad de procesadores o nodos que se dispongan. Luego, se pasaría a el mapeo de grupos en donde se asginaría cada grupo a los procesadores disponibles. Por último vendría el ordenamiento de tareas en donde el grupo de tareas que fueron asignadas a cada nodo o procesador, serían organizadas para realizar la ejecución en dicho nodo. Una explicación más extensiva se puede encontrar en el artículo de Palis and Liou (1997). Existen varios ejemplos de heurísticas diseñadas en este grupo, como pueden ser la Agrupación de Secuencia Dominante Yang and Gerasoulis (1994), el Método Lineal de Agrupación Palis and Liou (1997), Heurística de Mobilidad Dirigida Wu and Gajski (1990).

- **Heurísticas de duplicación de tareas**: En la revisión que hace Topcuoglu et al. (2002a) en su artículo mencionan que la idea detrás de este tipo de heurísticas, es calendarizar un grafo de tareas asignando algunas de sus tareas de manera redundante. Lo que hace que se reduzca el costo adicional requerido en la comunicación entre procesos como se puede verificar en los artículos de Ahmad and Yu-Kwong Kwok (1994), Lewis and Kruatrachue (1988) y Park et al. (1997). Sin embargo dentro de este grupo de heurísticas existen muchas diferencias entre sus implementaciones ya que en muchos casos varía la estrategia de selección de tareas para la duplicación. Este tipo de algoritmos según Topcuoglu et al. (2002a) son usualmente utilizados para un número ilimitado de nodos o procesadores homogéneos, además se menciona que tiene un valores de complejidad mucho más alto que los otros grupos de heurísticas.

Como se mencionó anteriormente, estos grupos de algoritmos basados en heurísticas en un principio fueron diseñados para procesadores o nodos homogéneos, sin embargo, al pasar del tiempo han ido sufriendo modificaciones debido al auge que han tenido la implementación de sistemas heterogéneos. Aunque resulta ser mucho más complicada la calendarización en sistemas heterogéneos debido a las diferentes tasas de ejecución entre los procesadores o nodos que componen al sistema, como se menciona en el artículo de Arabnejad (2012) Por ejemplo para el caso de las Heurísticas de calendarización de listas se han propuesto varias alternativas para procesadores o nodos heterogéneos:

- **Algoritmo HEFT:** Este algoritmo se basa en dos etapas clave: la Etapa de Priorización de Tareas y la Etapa de Asignación de Procesador.

  - **Etapa de Priorización de Tareas:** En esta etapa, se calculan los valores de prioridad para cada nodo, conocidos como upward rank. Este valor se basa en el promedio del tiempo de ejecución de las tareas y el promedio de los costos de comunicación entre tareas. Al final de esta etapa, se genera una lista de tareas ordenadas por su prioridad, de mayor a menor. En caso de empates, se selecciona una tarea de manera aleatoria. Aunque existen otras estrategias de programación para resolver empates, como seleccionar nodos cuyos hijos directos tengan mayores upward rank, estas estrategias suelen aumentar la complejidad temporal, por lo que se prefiere la selección aleatoria. El upward rank se calcula con la siguiente fórmula:

$$\text{ranku}(n_i) = w_i + \text{máx}(c_{jj} + \text{ranku}(r_{ij}))$$

  - **Etapa de Asignación de Procesador:** En esta etapa, se procesan los nodos en orden de prioridad, de mayor a menor, según su upward rank. Para cada nodo, se busca minimizar el valor del tiempo de finalización más temprano en el procesador asignado. EST(ni, pj) y EFT(ni, pj) son el tiempo de inicio de ejecución más temprano y el tiempo de finalización de ejecución más temprano del nodo ni en el procesador pj, respectivamente. Se definen de la siguiente manera:

$$\text{EST}(n_i, p_j) = \text{máx}(\text{TAvailable}[p_j], \underset{n_m \in \text{pred}(n_i)}{\text{máx}}(\text{EFT}(n_m, p_k) + c_{mij}))$$

$$\text{EFT}(n_i, p_j) = W_{ij} + \text{EST}(n_i, p_i)$$

    El algoritmo tiene una complejidad temporal de $O(e \times p)$, donde e representa las aristas y p los procesadores. Sin embargo, para grafos densos donde el número de tareas sea proporcional a $O(v^2)$, entonces la complejidad está en el orden de $O(v^2 \times p)$.

- **Calendarización de tareas de alto rendimiento**: Este algoritmo fue propuesto por Ilavarasan et al. (2005) y consta de tres fases: clasificación por niveles, priorización de tareas y selección de procesador. En la fase de clasificación por niveles, el grafo DAG dado se recorre de arriba hacia abajo para ordenar las tareas en cada nivel y agrupar aquellas que son independientes entre sí. Como resultado, las tareas en el mismo nivel pueden ejecutarse en paralelo. En la fase de priorización de tareas, se calcula y asigna una prioridad a cada tarea utilizando los atributos Costo de Enlace Descendente (DLC), Costo de Enlace Ascendente (ULC) y Costo de Enlace (LC) de la tarea. El DLC de una tarea es el costo de comunicación máximo entre todos los predecesores inmediatos de la tarea. El DLC de todas las tareas en el nivel 0 es 0. El ULC de una tarea es el costo de comunicación máximo entre todos los sucesores inmediatos de la tarea. El ULC para una tarea de salida es 0. El LC de una tarea es la suma de DLC, ULC y el LC máximo de todas sus tareas predecesoras inmediatas. En cada nivel, en función de los valores de LC, la tarea con el valor de LC más alto recibe la mayor prioridad, seguida de la tarea con el siguiente valor de LC más alto, y así sucesivamente en el mismo nivel. En la fase de

selección de procesador, se selecciona el procesador que proporciona el tiempo de finalización mínimo para una tarea y así poder ejecutarla. El algoritmo utiliza una política basada en inserción, que considera la inserción de una tarea en un intervalo de tiempo inactivo más temprano entre dos tareas ya programadas en un procesador tal y como se menciona en Arabnejad (2012).

Por otro lado como se mostró en la figura 4.1 existen otro conjunto de algoritmos o técnicas que reciben el nombre de **Búsquedas aleatorias guiadas**, en donde según Topcuoglu et al. (2002a) se usan decisiones aleatorias para ir guiándose dentro del espacio de búsqueda, lo cual no significa realizar pasos aleatorios simples como en los métodos tradicionales de búsqueda aleatoria. Estas técnicas combinan el conocimientos obtenido de los resultados de las búsquedas anteriores con algo de aleatorización de características para generar nuevos resultados. Los Algoritmos Genéticos resultan ser la técnica más popular y ampliamente estudiada dentro de este grupo, como se puede ver en los artículos de Edwin S.Hou and Ren (1994),H. Singh (1996) y Li et al. (2023). Los algoritmos genéticos suelen suelen dar salidas con buena calidad de calendarización, sin embargo sus tiempos para calendarizar resultan ser mucho mayores que los grupos que son basados en heurísticas. Un ejemplo de este tipo de algoritmos es:

El **Algoritmo genético de cola de prioridad múltiple (MPQGA)** que fue propuesto por Xu et al. (2012). En la revisión que hace Gilda (2013b) se menciona que éste consta de dos componentes claves: (1) un algoritmo genético para generar colas de prioridad múltiples para la planificación de tareas en sistemas de cómputo heterogéneos distribuidos y paralelos, y (2) un enfoque basado en heurística de tiempo de finalización más temprano heterogéneo (HEFT) para buscar una solución para asignar tareas a procesadores. Este utiliza una colección de soluciones que evolucionan a través de operadores genéticos para obtener mejores soluciones. Las nuevas soluciones (descendencia) para la siguiente generación se obtienen aplicando los siguientes dos operadores genéticos:

- **Cruzamiento**: que tiene como objetivo tomar las mejores características de cada padre y combinar las características restantes para formar la descendencia.
- **Mutación**: que tiene como objetivo introducir variaciones en los individuos.

El valor de aptitud juega un papel importante en decidir qué individuos se utilizarán para generar la población de la siguiente generación, mientras que los operadores genéticos realizan la evolución concreta.

El tiempo total de procesamiento o de ejecución es el tiempo de finalización más largo entre todas las subtareas, que es equivalente al tiempo de finalización real del nodo de salida T_{exit}. Para el problema de la planificación de tareas, el objetivo es obtener asignaciones de subtareas que garanticen un *makespan* mínimo para asegurar que no se viole la precedencia de las subtareas. Por lo tanto, el valor de la función de aptitud se define como: $Value_{\text{fitness}} = \text{TiempoTotalEjecución} = EFT(T_{\text{exit}})$.

El algoritmo HEFT se utiliza para asignar las subtareas a los procesadores. Las subtareas se han asignado a los procesadores en orden de prioridad. En cada paso de la asignación, el procesador seleccionado proporciona el tiempo de finalización más temprano para la subtarea en consideración, teniendo en cuenta todas las comunicaciones de los padres de la subtarea.

Sin embargo, a raíz del auge que está teniendo la inteligencia artificial en los último años, se han intentado proponer enfoques donde se mezclan heurísticas, con algoritmos de aprendizaje automático, brindando soluciones altamente eficientes.

4.1 Definición de algoritmo de aprendizaje automático

Un algoritmo de aprendizaje automático es un conjunto de técnicas y modelos matemáticos que permiten a los sistemas informáticos aprender a través de la experiencia y los datos sin ser programados explícitamente. En lugar de seguir un conjunto de reglas predefinidas, los algoritmos de Aprendizaje Automático utilizan datos de entrada y un proceso iterativo de prueba y error para mejorar su capacidad para predecir y tomar decisiones en situaciones futuras, como se menciona en Alpaydin (2010).

4.2 Uso de algoritmos de aprendizaje automático en la calendarización

El aprendizaje automático es una herramienta útil para la calendarización debido a su capacidad para analizar grandes conjuntos de datos y detectar patrones complejos. Dado lo expuesto en el capítulo anterior acerca de la complejidad del problema resulta conveniente el uso de este tipo de algoritmos para brindar soluciones cercanas a lo óptimo. En Cai and Chen (2019) se presenta una revisión exhaustiva del uso del aprendizaje automático para la gestión de recursos en la computación en la nube, incluyendo la calendarización de tareas y la asignación de recursos. Los autores discuten las técnicas de aprendizaje automático más comúnmente utilizadas, como las redes neuronales y los árboles de decisión, y cómo se pueden aplicar a la calendarización para mejorar la eficiencia y la precisión de la asignación de recursos.

Los modelos de aprendizaje profundo, como las Redes Neuronales Artificiales (ANN), pueden aprender a predecir la duración de las tareas y estimar el tiempo necesario para completar un proyecto. Esto puede ayudar a planificar mejor el uso de los recursos y programar las tareas en consecuencia. Además, los Algoritmos Genéticos (GA) pueden encontrar la combinación más eficiente de tareas y recursos para un proyecto, lo que puede ayudar a minimizar los costos y maximizar la utilización de los recursos. En Goodfellow et al. (2016) se proporciona una descripción completa de los modelos de aprendizaje profundo, incluyendo las Redes Neuronales Artificiales, mientras que en Goldberg (1989) se proporciona una descripción detallada de los Algoritmos Genéticos y su aplicación en la optimización.

4.3 Tipos de algoritmos usados en la calendarización estática heterogénea

Los algoritmos de Aprendizaje Automático más usados para resolver el problema de la calendarización estática heterogénea son:

1. Redes Neuronales Artificiales (ANN): En la calendarización de tareas, las redes neuronales artificiales se utilizan para predecir la duración de las tareas y estimar el tiempo necesario para completar un proceso. Estos algoritmos pueden aprender

a partir de grandes conjuntos de datos y detectar patrones complejos en los datos de entrada.

2. Árboles de Decisión (DT): Los árboles de decisión son algoritmos de aprendizaje supervisado que se utilizan para clasificar tareas según sus características y asignarlas a los recursos adecuados. Los DT son fáciles de entender e interpretar, lo que los hace útiles para la toma de decisiones.

3. Algoritmos Genéticos (GA): En la calendarización estática heterogénea, los Algoritmos Genéticos pueden encontrar la combinación más eficiente de tareas y recursos para un proyecto, lo que puede ayudar a minimizar los costos y maximizar la utilización de los recursos.

4. Máquinas de Vectores de Soporte (SVM): Las máquinas de vectores de soporte se utilizan para clasificar tareas según sus características y asignarlas a los recursos adecuados. Las SVM son útiles cuando los datos son no lineales y las relaciones entre las variables son complejas.

5. Algoritmos de Agrupamiento (Clustering): Los algoritmos de agrupamiento se utilizan para agrupar tareas según sus características y asignarlas a los recursos adecuados. Estos algoritmos pueden identificar patrones en los datos de entrada y agrupar tareas similares.

En Huang and Chen (2013) se presenta una estrategia para la calendarización estática heterogénea basada en redes neuronales artificiales y Algoritmos Genéticos. Los resultados experimentales muestran que la estrategia propuesta es efectiva en la reducción del tiempo de ejecución de las tareas. Comparándose con otros métodos de programación lineal y algoritmos de planificación basado en prioridad. Sus resultados muestran que su método supera a los otros dos en términos de tiempo de ejecución.

En Błaszczyk and Proficz (2017) se revisan algoritmos de calendarización estática para sistemas multiprocesador en tiempo real que son conscientes del consumo de energía. Los autores analizan diferentes técnicas y enfoques, incluyendo algoritmos de búsqueda exhaustiva, heurísticos, metaheurísticos y de Aprendizaje Automático. A través de esta revisión, los autores identifican las ventajas y desventajas de los diferentes enfoques y destacan áreas de investigación futura en la calendarización estática consciente de la energía.

En Guo and Wang (2012) se presenta un enfoque de calendarización estática basado en Algoritmos Genéticos para mejorar la eficiencia energética en la ejecución de aplicaciones paralelas en clústeres heterogéneos. El algoritmo propuesto busca minimizar el tiempo total de ejecución y el consumo de energía al mismo tiempo. Los autores llevan a cabo experimentos utilizando aplicaciones paralelas reales y muestran que su enfoque basado en Algoritmos Genéticos logra un rendimiento y una eficiencia energética significativamente mejores en comparación con otros enfoques existentes.

En Abdollahi and Rajabi (2019) los autores proponen un algoritmo de calendarización estática basado en un enfoque genético híbrido para tareas con restricciones de precedencia en sistemas informáticos heterogéneos. El enfoque híbrido combina Algoritmos Genéticos con heurísticas de calendarización clásicas, como HEFT (Heterogeneous Earliest Finish

Time) y CPOP (Critical Path on a Processor). Los autores realizan experimentos utilizando conjuntos de tareas sintéticas y reales y demuestran que su algoritmo híbrido supera a otros algoritmos.

En Zhang and Zhong (2014) se presenta un algoritmo híbrido basado en técnicas genéticas para la calendarización estática de tareas en sistemas heterogéneos con múltiples objetivos, incluyendo el rendimiento y la eficiencia energética. El algoritmo híbrido combina Algoritmos Genéticos con estrategias de búsqueda local para mejorar la convergencia y la diversidad de las soluciones. Los autores llevan a cabo experimentos utilizando casos de prueba sintéticos y demuestran que su enfoque híbrido supera a otros algoritmos de Aprendizaje Automático en términos de rendimiento y eficiencia energética.

En Guo et al. (2012) se presenta un enfoque heurístico para la calendarización estática de tareas en computación en la nube, considerando factores como la heterogeneidad de los recursos y la eficiencia energética. El algoritmo propuesto, llamado Heuristic Task Scheduling Algorithm (HTSA), utiliza técnicas heurísticas y de Aprendizaje Automático para asignar tareas a recursos de cómputo de manera eficiente. Los autores comparan el rendimiento del HTSA con otros algoritmos de calendarización, como Min-Min y Max-Min, utilizando experimentos basados en conjuntos de tareas sintéticas. Los resultados muestran que el HTSA logra una mejor eficiencia en la asignación de tareas y una reducción en el tiempo total de ejecución en comparación con otros enfoques.

Capítulo 5

Predicción del tiempo de las tareas

La predicción de tiempo conecta directamente las tareas con la/las unidades de procesamiento que cuente el sistema, para poder obtener un estimado del tiempo de ejecución de las tareas en las distintas unidades de procesamiento que existan. Ésto resulta útil, ya que muchas heurísticas o algoritmos de Aprendizaje Automático utilizan como entrada, para realizar su proceso de calendarización, una matriz de costo de ejecución. En esta matriz las filas indican los tiempos de ejecución de una tarea en específico, y las columnas los tiempos de ejecución de distintas tareas en un procesador determinado
.

5.1 Escenarios de ejecución de tareas en recurso

En un sistema donde se utiliza la calendarización, como en la nube, servidores HPC entre otros, se pueden dar tres escenarios donde resulte fundamental la predicción de tiempo.

- Escenario 1- (Muchas recursos-una sola tarea): En este escenario la idea es verificar o estimar los tiempos de ejecución de la única tarea existente perteneciente al Trabajo (ver sección 2.4) asignado al sistema, en todas las unidades de procesamiento con las que se cuente. Este proceso generaría una matriz de costo que contaría solamente con una fila y varias columnas. En la figura 5.1 se puede observar el proceso donde se trata de predecir el tiempo de ejecución de una tarea en cada unidad de procesamiento, y en la figura 5.2 la matriz de costo resultante de dichas predicciones. En este caso la predicción del tiempo de ejecución puede resolver directamente la calendarización, ya que bastaría con saber la unidad de procesamiento donde la tarea se ejecute más rápido y asignarla ahí. Aunque también depende del objetivo que se esté buscando optimizar, porque se podría calendarizar en función del gasto energético que consuma la tarea y en qué unidad de procesamiento se podría obtener un menor gasto.
- Escenario 2 - (Muchas tareas-un solo recurso): En este escenario se busca obtener un orden de ejecución de las tareas. Por lo tanto, se tendría que predecir las distintas ejecuciones de las tareas en la única unidad de procesamiento con la que se cuente, como se muestra en la figura 5.3. Para este proceso se tendría una matriz de costo que contaría con varias filas en dependencia de la cantidad de tareas que contenga el Trabajo (ver sección 2.4), y una sola columna que representaría los

tiempos estimados de ejecución de las tareas en la unidad de procesamiento que se disponga, como se muestra en la figura 5.4. Si el Trabajo asignado a la única unidad de procesamiento cuenta con tareas que están interrelacionadas entre sí y se comparten datos, entonces habría que respetar ese orden, y encontrar una criterio de organización para que la unidad de procesamiento las pueda ejecutar. Este tipo tareas, que están interrelacionadas, típicamente se pueden representar a través de un grafo DAG (ver sección 2.5) donde los nodos representan a las tareas y las aristas las dependencias comunicacionales entre ellas. Las arista dirigidas representa qué tarea depende de otra en un sentido jerárquico. Una forma típica de organización de las tareas en un grafo DAG, es a través de un *upward rank*, como se propone en Topcuoglu et al. (2002b). Este *upward rank* se establece de manera recursiva para cada tarea en un grafo DAG de la siguiente forma:

$$\text{ranku}(n_i) = w_i + \text{máx}\left(c_{ij} + \text{ranku}(r_{ij})\right)_{r_{ij} \in \text{succ}(n_i)}$$

donde n_i es la tarea-iésima, w_i representa el costo de ejecución de la tarea en la unidad de procesamiento i, aunque en este escenario, como se está contemplando una sola unidad de procesamiento, sería el costo o el tiempo de ejecutar a la tarea en esa sola unidad, c_ij sería el costo de transferir datos de la tarea i a la tarea j, sin embargo los costos de transferencia de datos de una tarea con otra dentro de la misma unidad de procesamiento son 0, y r_ij sería el conjunto de nodos o tareas que son sucesores directos en el nodo n_i.

Por tanto, para saber el *upward rank* de una tarea es necesario conocer el *upward rank* de sus sucesores directos. Se puede entender que habría que recorrer el grafo para encontrar el nodo que no tuviera sucesores, ya que su *upwardrank* sería igual solamente a su costo de ejecución. Por esta razón, es que tiene un comportamiento recursivo. En la figura 5.5 se puede verificar que para encontrar el *upward rank* del nodo 1 habría que encontrar los *upward rank* de cada conjunto de nodos en el nivel jerárquico hacia abajo recursivamente, y como el último nivel jerárquico que se tiene contiene solamente al nodo 10, entonces el primer nodo a calcular su *upward rank* sería dicho nodo 10. Luego se construyen los *upward rank* de los nodos en el nivel jerárquico superior que dependen directamente del nodo 10, hasta llegar al nodo 1.

En el caso de que no se tuviera una interrelación entre los nodos o tareas, se podría buscar otra forma de organización de las tareas que se fueran a ejecutar en esa unidad. Podría ser, por ejemplo, encontrar o predecir, únicamente su tiempo de ejecución, para luego ordenarlas bajo ese criterio.

- Escenario 3- (Muchos recursos-muchas tareas): Este representa el escenario más difícil, ya que resulta computacionalmente complejo encontrar todas las combinaciones posibles de ejecución entre tareas y unidades de procesamiento. Para luego quedarse con la combinación que menos tiempo total de calendarización produzca. Hacer una búsqueda exhaustiva del espacio de soluciones para muchos casos de grafos DAG es computacionalmente costoso, como se mencionó en la sección 3.4. Por ejemplo, en la figura 5.6 se representa un Trabajo compuesto por 9 tareas, la figura 5.7 representa la matriz de costos de ejecución de dicho grafo, obtenida luego de realizar el proceso de predicción de tiempo de cada tarea en las unidades de procesamiento con que se cuente, como se muestra en la figura 5.8. La figura 5.9

representa los costos de comunicación entre tareas, donde cada elemento de la matriz representa el costo de comunicar datos desde el nodo padre (fila) al nodo hijo (columna). En este caso, es donde se hace necesario encontrar métodos computables como heurísticas, algoritmos de aprendizaje profundo, aprendizaje por refuerzo, para lograr obtener soluciones de calendarización buenas. En la figura 5.10, utilizando la heurística HEFT propuesta por Topcuoglu et al. (2002b) y descrita en el capítulo 4, se logra calendarizar el grafo DAG representado en la figura 5.6. Por otra parte, es fácilmente comprobable que si hay variaciones en las estimaciones de los tiempos de ejecución de las tareas en las distintas unidades de procesamiento, la calendarización generada por la heurística HEFT y otros algoritmos, sufre modificaciones, ya que estos algoritmos para calendarizar asumen *a priori* que se cuenta con la matriz de costos. Por lo que se requiere ser lo más preciso posible al momento de estimar dichos costos.

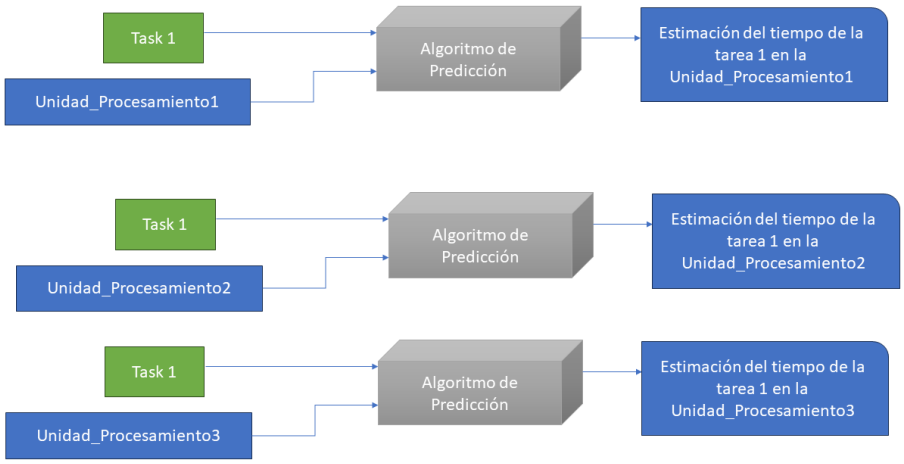

Figura 5.1: Predicción de una tarea en varias unidades de procesamiento.

	Tiempo Unidad 1	Tiempo Unidad 2	Tiempo Unidad 3
Tarea 1	23.334	29.341	17.543

Figura 5.2: Matriz de costo de una tarea en distintas unidades de procesamiento.

5.2 Técnicas de predicción de tiempo

Dentro de los Sistemas Heterogéneos (ver sección 3.1) existen tres tipos de técnicas utilizadas para realizar predicciones de tiempo **Análisis de Código**, **Evaluación Analítica de Rendimiento/ Perfilado de Código** y **Predicción Estadística** Iverson et al. (1999).

- **Análisis de código**: Éste tipo de técnica implica examinar el código fuente de la tarea sin ejecutarlo y realizar diferentes tipos de análisis, por ejemplo el análisis

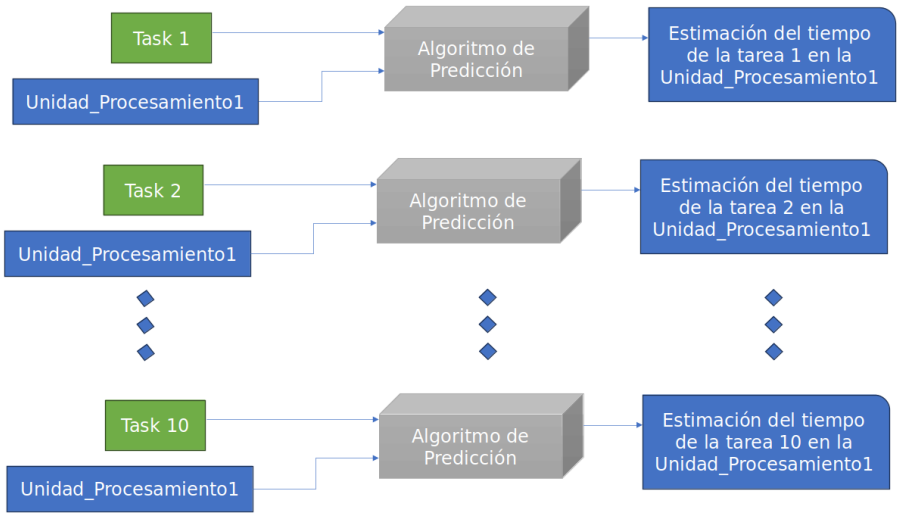

Figura 5.3: Predicción de varias tareas en una unidad de procesamiento.

	Unidad de procesamiento_1
Tarea 1	12.245
Tarea 2	11.234
Tarea 3	10.432
Tarea 4	13.912
Tarea 5	11.912
Tarea 6	14.321
Tarea 7	14.999
Tarea 8	15.321
Tarea 9	16.342
Tarea 10	17.432

Figura 5.4: Matriz de costo con diez tareas y una unidad de procesamiento.

de complejidad algorítmica, el seguimiento de dependencias de datos y el análisis de flujo de control. Estos análisis permiten identificar posibles cuellos de botella, tiempos de ejecución dominantes y patrones de comportamiento que afectan el rendimiento de la tarea. Las partes del código analizadas son evaluadas por métricas como *NPath Complexity* Bang et al. (2015) y *Cyclomatic Complexity* Bang et al. (2015). A partir de estos análisis, se pueden hacer estimaciones razonables sobre el tiempo de ejecución de la tarea en función de su código. Es importante destacar que estas estimaciones son aproximadas y pueden variar según las características específicas del entorno de ejecución. Éste tipo de análisis suele estar limitado a un tipo de código específico o a una clase limitada de arquitecturas Iverson et al. (1999).

- **Perfilado de código**: El perfilado de código es una técnica utilizada para analizar el código fuente con el objetivo de identificar el potencial de paralelismo que puede existir en él. Consiste en dividir el código en segmentos más pequeños. Luego, a

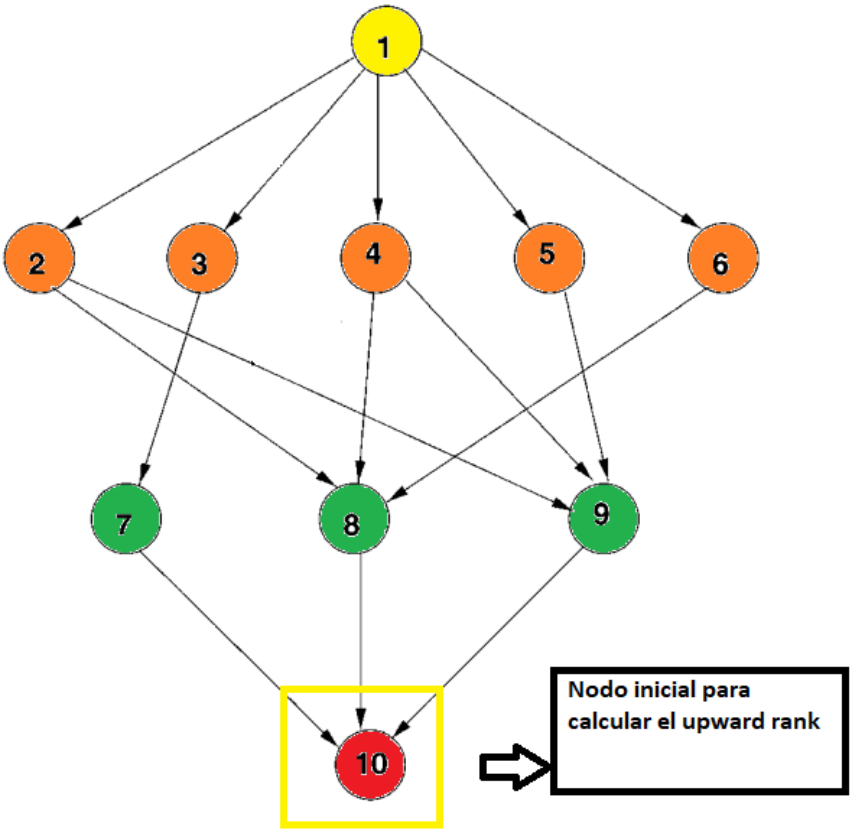

Figura 5.5: Grafo de tareas de un trabajo. Primer nodo a calcular *upwardrank*.

cada segmento se le asigna un procesador óptimo, que pueda ofrecer un rendimiento mejorado para ese tipo de segmento. Una vez que se ha dividido el código en segmentos y se han asignado los procesadores adecuados, se utiliza un perfilador de tipo de código, para analizar cada segmento y determinar su naturaleza. Los diferentes tipos de código que se pueden identificar incluyen **descompuestos en vectores**, **no descompuestos en vectores**, **paralelismo de grano fino/grueso**, **paralelismo SIMD/MIMD**, **escalar**, **especiales con propósitos específicos**. En resumen, el perfilado de código es una técnica que descompone el código en segmentos más pequeños y asigna procesadores óptimos a cada segmento. El análisis del tipo de código permite identificar el potencial de paralelismo y determinar la naturaleza de cada segmento. Esto ayuda a aprovechar de manera eficiente los recursos de procesamiento en entornos de computación heterogéneos y mejorar el rendimiento de las aplicaciones. Más adelante se darán detalles sobre éstos tipos de perfilados Freund (1989) Shanthini and Shankarkumar (2012).

- **Evaluación analítica de rendimiento**: El *benchmarking* analítico es un proceso que se utiliza para analizar el rendimiento de las unidades de procesamiento disponibles en diferentes tipos de código. Su objetivo es identificar los procesadores que son más adecuados para cada tipo de código (identificado en el perfilado de código) en términos de eficiencia y capacidad de procesamiento. En lugar de simplemente ejecutar en un procesador un código de referencia o conjunto de bucles y observar los resultados generales, la evaluación analítica de rendimiento busca realizar un análisis más profundo de cómo los factores relevantes contribuyen al rendimiento

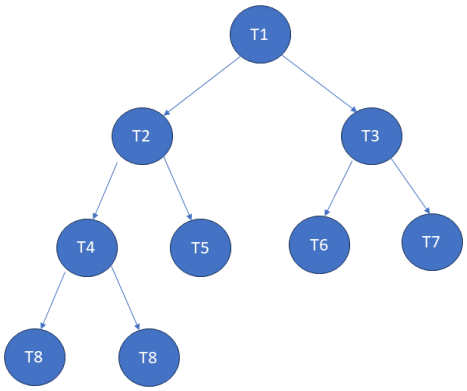

Figura 5.6: Grafo DAG 9 tareas.

	Unidad 1	Unidad 2	Unidad 3
Tarea 1	3	6	4
Tarea 2	5	6	9
Tarea 3	11	12	15
Tarea 4	17	43	12
Tarea 5	32	21	35
Tarea 6	20	22	32
Tarea 7	32	21	27
Tarea 8	45	32	21
Tarea 9	46	56	43

Figura 5.7: Matriz de costo de la figura 5.6.

del procesador. Esto implica considerar aspectos como la arquitectura del procesador, el uso de memoria y otros factores que pueden afectar el rendimiento Freund (1989).

- **Predicción estadística**: En este enfoque, se utiliza el historial de ejecuciones pasadas para predecir el tiempo de ejecución de una tarea. El método de predicción se basa en los datos de entrada de la tarea, por lo que no se necesita ningún conocimiento adicional. El proceso consta de dos iteraciones: en la primera, se buscan trabajos similares utilizando parámetros como el usuario, el sistema operativo, el número de nodos utilizados y la arquitectura de la máquina. Una vez que se encuentran trabajos similares, se agrupan juntos. En la segunda iteración, se realiza la predicción del tiempo de ejecución considerando la media, utilizando funciones de probabilidad de tiempos de ejecución anteriores o mediante un modelo de análisis de regresión Shanthini and Shankarkumar (2012).

Para el caso del **Análisis del código** Iverson et al. (1999) señala que muy difícil llevarlo a cabo en entornos de sistemas heterogéneos, debido a las limitaciones en cuanto a la

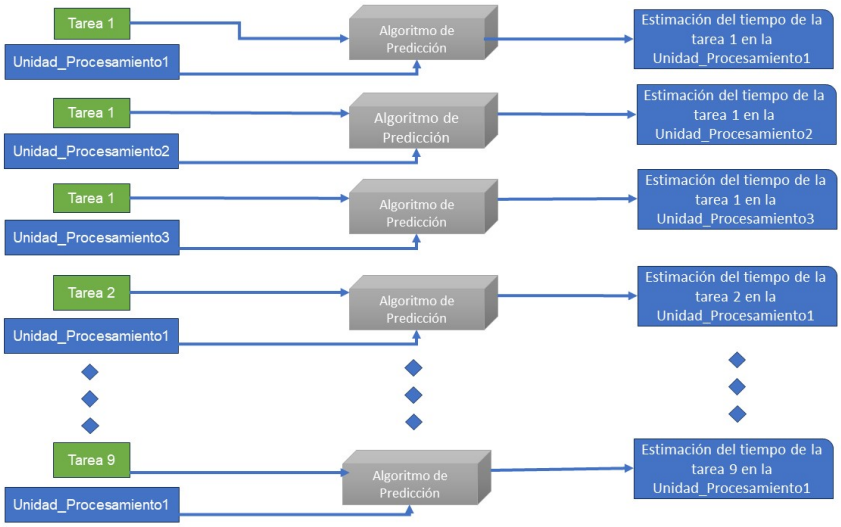

Figura 5.8: Predicción de tiempo de muchas tareas en varias unidades de procesamiento.

	Tarea 1	Tarea 2	Tarea 3	Tarea 4	Tarea 5	Tarea 6	Tarea 7	Tarea 8	Tarea9
Tarea 1	0	2	3	0	0	0	0	0	0
Tarea 2	0	0	0	2	4	0	0	0	0
Tarea 3	0	0	0	0	0	3	1	0	0
Tarea 4	0	0	0	0	0	0	0	4	6
Tarea 5	0	0	0	0	0	0	0	0	0
Tarea 6	0	0	0	0	0	0	0	0	0
Tarea 7	0	0	0	0	0	0	0	0	0
Tarea 8	0	0	0	0	0	0	0	0	0
Tarea 9	0	0	0	0	0	0	0	0	0

Figura 5.9: Matriz de costos de comunicación de la figura 5.6.

especificidad que brindan sus predicciones, ya que se limita a determinadas arquitecturas y determinados tipos de código. En este trabajo no se toma en consideración este tipo de técnicas debido a esas limitaciones.

El **perfilado de código** y la **evaluación análitica de rendimiento** son dos técnicas relacionadas, sin embargo, implican un análisis detallado del código de ejecución para identificar las regiones que pertenecen a un perfilado específico. Aunque resultan técnicas muy poderosas, a veces resulta complicado obtener el código fuente de ciertas tareas que se ejecutan en sistemas reales, debido a problemas de licencias Bielecki and Śmiałek (2022). En este trabajo no se considerará su uso .

Por otro lado se tiene a la **predicción estadística**, que utiliza observaciones pasadas para realizar predicciones. Esta técnica trata al código como una caja negra, abstrayéndose de elementos intrínsecos del código fuente Shanthini and Shankarkumar (2012) Bielecki

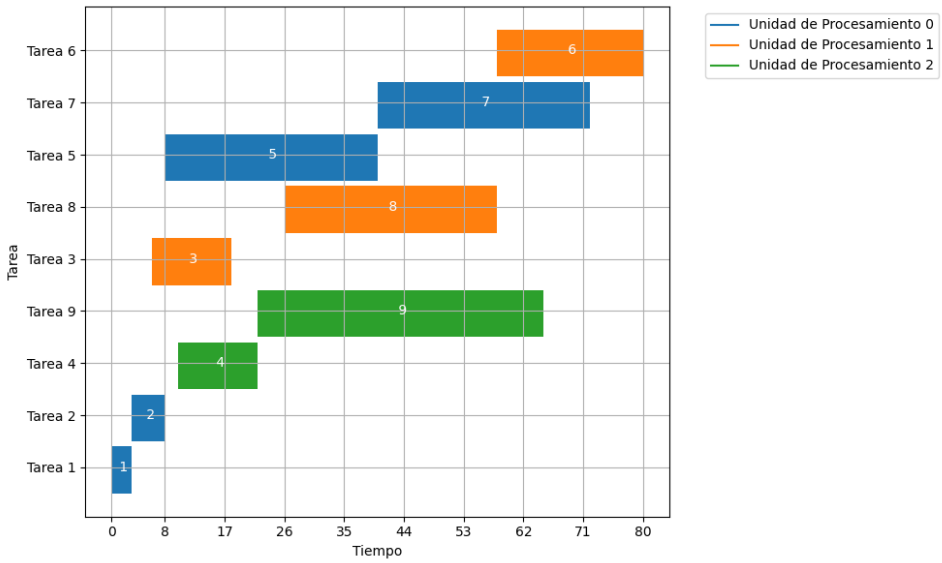

Figura 5.10: Calendarización del Trabajo representado por el grafo DAG en la figura 5.6.

and Śmiałek (2022). Para este trabajo, esta técnica resulta muy importante, pues permite tomar en cuenta características de la tarea que no asumen la estructura interna del código fuente. Lo que hace que se resuelva el problema de no contar en muchas ocasiones con las licencias de los códigos para poderlos examinar.

Dentro de esta técnica se utilizan métodos como la media de las ejecuciones pasadas, el uso de funciones de probabilidad y modelos de análisis de regresión. A raíz del auge que ha tenido la Inteligencia Artificial y el Aprendizaje Automático, se han propuesto muchos modelos de regresión para intentar hacer estimaciones precisas del tiempo de ejecución de las tareas en entornos específicos.

Por tanto para relacionar los métodos anteriores con la calendarización habría que realizar una caracterización conjunta de tareas y recursos, a partir de la necesidad de tener en cuenta tanto las características de las tareas como las de los recursos al mismo tiempo. En muchos casos, la predicción de una tarea específica puede depender, no solo de sus propias características, sino también de las características del recurso en el que se ejecuta. Por lo tanto, al considerar las tareas y los recursos en conjunto, se puede obtener una visión más completa y precisa de predecir su tiempo de ejecución en un recurso determinado. Lo que aumentaría la precisión de las predicciones para generar las matrices de costo correspondientes en dependencia de los escenarios descritos anteriormente.

Matemáticamente, esta idea se puede representar mediante el concepto de producto cartesiano. Dados dos conjuntos, el conjunto de tareas T y el conjunto de recursos R, el producto cartesiano $T \times R$ es el conjunto de todos los pares ordenados (t, r) donde t pertenece a T y r pertenece a R. Cada par ordenado representa una combinación específica de tarea y recurso.

En este contexto, cada elemento del producto cartesiano $T \times R$ representa una combinación específica de tarea y recurso, y el valor de este elemento sería el indicador a optimizar, en este caso, el tiempo de ejecución. Es decir, para cada par (t, r) en $T \times R$, se tiene un tiempo de ejecución asociado que se desea minimizar.

En Pham et al. (2020)se presenta una nueva forma de predecir los tiempos de ejecución

de las tareas en flujos de trabajo en la nube, considerando diferentes conjuntos de datos de entrada. El enfoque propuesto utiliza técnicas de Aprendizaje Automático en dos etapas, teniendo en cuenta parámetros que reflejan información sobre el rendimiento de la tarea en un recurso, en tiempo real. Tomando en cuenta los factores de la tarea y el dispositivo donde se ejecuta.

En la primera etapa, se crean modelos para cada uno de los parámetros en tiempo de ejecución. Estos modelos se generan utilizando técnicas de Aprendizaje Automático y tienen como objetivo capturar información sobre el hardware asignado por una nube específica y luego ejecutar la tarea en una máquina virtual. Esto se logra mediante un proceso de correlación entre los datos de entrada del flujo de trabajo y la información disponible sobre el tipo de máquina virtual y sus parámetros en tiempo de ejecución. En la segunda etapa, se utilizan estos modelos de parámetros en tiempo de ejecución para predecir el tiempo final de una tarea.

Los resultados empíricos obtenidos muestran que se logra un error de estimación mínimo de 1.6 % y un error máximo de 12.2 %, mientras que los métodos existentes tuvieron errores superiores al 20 % en la mayoría de las tareas evaluadas Pham et al. (2020). Además, estos modelos son fácilmente transferibles a nuevas nubes con errores mínimos y requieren pocas ejecuciones para adaptarse.

En Kiran et al. (2009) se propone un sistema de predicción del tiempo de ejecución de trabajos para mejorar la eficiencia en un entorno en malla. El sistema se enfoca en tareas de paradigma imperativo y utiliza una metodología innovadora que combina análisis estático, *benchmarking* analítico y enfoque basado en compiladores. Mediante el análisis del programa en segmentos, se obtiene el tiempo de ejecución de cada segmento y se combinan para obtener el tiempo total de ejecución. Los resultados experimentales demuestran una precisión de predicción superior al 80 % Pham et al. (2020). Además, se exploran posibles mejoras, como el manejo de otros paradigmas y la integración del sistema en un entorno de Grid real.

En el estudio de Ali et al. (2000), se analiza un entorno de Computación Heterogénea (HC) simulado, utilizando los tiempos de ejecución esperados de las tareas asignadas a diferentes máquinas. Estos tiempos se organizan en una matriz de "tiempo esperado de cómputo" (ETC), que representa el modelo del sistema HC. El objetivo es simular diversos entornos HC y evaluar el rendimiento relativo de diferentes heurísticas de asignación bajo distintas circunstancias. El modelo ETC se utiliza para expresar la heterogeneidad en los tiempos de ejecución de las tareas y entre las máquinas en el sistema HC. Se describe una técnica existente basada en rangos para generar las matrices ETC, y se propone una técnica basada en el coeficiente de variación que se compara con la técnica basada en rangos. La generación basada en el coeficiente de variación brinda un mayor control sobre la dispersión de los valores en una fila o columna específica de la matriz ETC, en comparación con el método basado en rangos.

5.3 Regresores

En el campo del aprendizaje automático y la estadística, los regresores son modelos matemáticos utilizados para realizar tareas de regresión. La regresión es una técnica es-

tadística que se emplea para predecir valores continuos basados en datos previos. En otras palabras, los regresores son algoritmos que buscan encontrar una relación funcional entre un conjunto de variables de entrada (características) y una variable de salida (objetivo) con valores numéricos Sarstedt and Mooi (2014).

El objetivo principal de un regresor es encontrar una función que mapee las variables de entrada a la variable de salida de manera óptima, de acuerdo con ciertos criterios de rendimiento y precisión. En el contexto de la regresión, esta función se conoce como la "función de regresión"Sarstedt and Mooi (2014).

Los regresores pueden variar en complejidad y capacidad para modelar relaciones entre las variables. Algunos regresores son lineales, lo que significa que asumen que la relación entre las variables de entrada y la variable de salida es aproximadamente lineal. Otros regresores, como las redes neuronales, son capaces de capturar relaciones no lineales más complejas.

En el campo de la regresión, existen dos enfoques principales para modelar la relación entre las variables de entrada y la variable de salida: la regresión paramétrica y la regresión no paramétrica.

5.3.1. Regresión paramétrica

La regresión paramétrica es una técnica estadística utilizada para modelar la relación entre una variable dependiente y una o más variables independientes. En esta técnica, se asume una forma funcional específica para esta relación, como una línea recta en el caso de la regresión lineal.

El objetivo de la regresión paramétrica es encontrar los valores óptimos para los parámetros del modelo, que representan las pendientes y los interceptos de las variables independientes, se busca que el modelo se ajuste de la mejor manera posible a los datos observados. Estos parámetros fijos permiten describir y predecir cómo cambia la variable dependiente, en función de las variables independientes Robinson (2008).

Para encontrar los valores óptimos de los parámetros, se utiliza un método de estimación, como el método de mínimos cuadrados, que minimiza la discrepancia entre los valores predichos por el modelo y los valores observados en los datos de entrenamiento. Una vez que se han estimado los parámetros, se puede utilizar el modelo para hacer predicciones sobre nuevos datos o para analizar la importancia relativa de las variables independientes en la variable dependiente Mahmoud (2019).

Un modelo de regresión paramétrica se puede expresar de la siguiente forma:

$$Y = f_1(x_1, \beta_1) + f_2(x_2, \beta_2) + ...f_n(x_n, \beta_n) + \epsilon$$

donde x_1, x_2, x_n son las variables independiente, Y es la variable dependiente, $\beta_1, \beta_2, \beta_n$ son los coeficientes de las variables independientes, ϵ es el término del error aleatorio y

f_1, f_2, f_n representan las funciones que describen las relaciones.

En los modelos paramétricos, se asume que los parámetros son vectores p-dimensionales, donde p es el número de parámetros del modelo. Cada elemento del vector de parámetros representa una característica o coeficiente específico que determina la relación entre las variables Mahmoud (2019).

Las funciones subyacentes f_n del modelo y el término de error ϵ ya son conocidas. Estas funciones pueden ser cualquier forma funcional conocida, como una función lineal, cuadrática o incluso no lineal, dependiendo del contexto y la naturaleza de los datos.

El término de error representa la diferencia entre el valor observado de la variable dependiente Y y el valor predicho por el modelo. Se asume que este término de error sigue una distribución conocida a priori, lo que significa que se tiene información previa sobre su distribución antes de realizar cualquier inferencia.

La importancia de conocer las funciones y la distribución del término de error ϵ radica en la capacidad de realizar inferencias sobre los parámetros del modelo y realizar predicciones. Al tener información sobre estas funciones y la distribución del error, se pueden estimar los parámetros óptimos del modelo y realizar pruebas estadísticas para evaluar la significancia de las variables independientes.

Modelos lineales:

En los modelos lineales, la relación entre la variable dependiente y las variables independientes se establece de forma lineal mediante la combinación lineal de los coeficientes de regresión. Esto implica que la respuesta esperada es una función lineal de las variables independientes, multiplicadas por los respectivos coeficientes.

Un ejemplo común de un modelo lineal es la regresión lineal múltiple, donde se asume una relación lineal entre la variable dependiente y múltiples variables independientes. La ecuación del modelo se expresa como:

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \varepsilon$$

donde Y es la variable dependiente, x_1 y x_2 son las variables independientes, β_0, β_1 y β_2 son los coeficientes de regresión, y ε representa el término de error.

Otro ejemplo es la regresión polinómica, que permite modelar relaciones no lineales mediante el uso de términos de orden superior de las variables independientes. En este caso, el modelo sigue siendo lineal en los coeficientes de regresión, pero la combinación lineal incluye términos polinómicos, como x_2^2, para capturar la curvatura en los datos
.

La regresión de Poisson es otro ejemplo de un modelo lineal, pero en este caso se utiliza una función de enlace logarítmico para relacionar las variables independientes con una variable de conteo. El modelo se expresa como:

$$\log(\mu) = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

donde μ representa la media de la variable de conteo.

Estos modelos lineales son paramétricos, lo que significa que se asume una forma funcional específica para la relación entre las variables. Los parámetros β_0, β_1, β_2, etc., se estiman utilizando métodos como el de mínimos cuadrados ordinarios o el de máxima verosimilitud, y el objetivo es encontrar los valores óptimos de los coeficientes que mejor se ajusten a los datos de entrenamiento Mahmoud (2019).

Modelos no lineales:

Los modelos de regresión no lineal son modelos paramétricos en los que la función es conocida pero no lineal en los parámetros. A continuación se muestran algunos ejemplos:

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 e^{\beta_3 x_2} + \varepsilon$$
$$Y = \frac{e^{\beta_0 + \beta_1 x_1}}{1 + e^{\beta_0 + \beta_1 x_1}} + \varepsilon$$
$$Y = \beta_0 + (0{,}4 - \beta_0) e^{-\beta_1 (x_1 - 5)} + \beta_2 x_2 + \varepsilon$$

Para estimar los parámetros de estos modelos, se puede utilizar el método de mínimos cuadrados. Hay varios algoritmos disponibles para la estimación de mínimos cuadrados, incluyendo el método de Newton, el algoritmo de Gauss-Newton y el método de Levenberg-Marquardt.

A veces, los modelos no lineales se pueden transformar en modelos lineales mediante una transformación simple. Por ejemplo, la ecuación $Y = \beta_0 x^{\beta_1 + x}$ se puede escribir como $Y^* = \theta_0 + \theta_1 x^*$ donde $Y^* = \frac{1}{Y}$ y $x^* = \frac{1}{x}$. En esta forma transformada, el modelo se vuelve lineal en los parámetros.

En algunos casos, si la relación entre la variable dependiente y las variables independientes no es lineal, se pueden aplicar soluciones simples antes de recurrir a modelos no lineales o no paramétricos. Estas soluciones incluyen el uso de transformaciones no lineales de las variables (por ejemplo, transformación logarítmica, raíz cuadrada o transformación potencial) o agregar variables adicionales que sean funciones no lineales de las variables existentes Robinson (2008).

Sin embargo, es importante tener en cuenta que los modelos paramétricos no siempre son adecuados para ajustar datos en muchas aplicaciones del mundo real, y pueden ser necesarios enfoques más flexibles como modelos no lineales o no paramétricos Mahmoud (2019) .

Las redes neuronales, incluyendo las redes neuronales secuenciales y las redes LSTM(Long Short Term Memory), pueden ser confusas para clasificar dentro de la regresión paramétrica, debido a su naturaleza híbrida y versátil. A continuación se explica por qué se incluyen dentro de esta clasificación.

Caso especial: Red Neuronal Secuencial (Feedforward Neural Network)

Una red neuronal secuencial, también conocida como red neuronal feedforward, es un tipo de red neuronal en la que la información fluye en una sola dirección, de la entrada a la capa de salida. Cada capa de la red está compuesta por unidades (neuronas) conectadas a todas las unidades de la capa anterior y siguiente Aggarwal (2018).

Razones de Parametrización

La red neuronal secuencial es paramétrica porque tiene una estructura fija con una cantidad finita de parámetros ajustables, es decir, los pesos y sesgos de las conexiones entre las unidades. Estos parámetros son esenciales para que la red neuronal pueda aprender y hacer predicciones precisas.

La matemática detrás de la red neuronal secuencial se basa en el cómputo de las salidas de cada unidad, que se obtienen a través de una función de activación aplicada a la suma ponderada de las entradas y los pesos de las conexiones.

La salida $y_i^{(l)}$ de la unidad i en la capa l se calcula mediante la siguiente fórmula:

$$y_i^{(l)} = \sigma\left(\sum_{j=1}^{n_{l-1}} w_{ij}^{(l)} \cdot x_j^{(l-1)} + b_i^{(l)}\right)$$

Donde:
-$y_i^{(l)}$ es la salida de la unidad i en la capa l.
-σ es la función de activación, que introduce la no linealidad en el modelo.
-$w_{ij}^{(l)}$ es el peso de la conexión entre la unidad j en la capa $l-1$ y la unidad i en la capa l.
-$x_j^{(l-1)}$ es la entrada proveniente de la unidad j en la capa $l-1$.
-$b_i^{(l)}$ es el sesgo (bias) de la unidad i en la capa l.

El objetivo del entrenamiento es ajustar los pesos y sesgos ($w_{ij}^{(l)}$ y $b_i^{(l)}$) de manera que la red neuronal pueda hacer predicciones precisas en el conjunto de entrenamiento. Sin embargo, el proceso de entrenamiento se va a profundizar más adelante

Caso especial: Red LSTM (Long Short-Term Memory)

Una red LSTM es un tipo especial de red neuronal recurrente (RNN) que está diseñada para manejar secuencias de datos, como series temporales. Las LSTM tienen una arquitectura más compleja que las redes neuronales secuenciales y utilizan compuertas para controlar el flujo de información a través del tiempo, lo que les permite aprender patrones a largo plazo en secuencias Van Houdt et al. (2020).

Razones de parametrización

Al igual que la red neuronal secuencial, la red LSTM también es paramétrica porque tiene una estructura fija con una cantidad finita de parámetros ajustables, que son los pesos y sesgos de las conexiones entre las unidades y las compuertas de la LSTM.

La matemática detrás de una celda LSTM implica el cálculo de las salidas y estados de las compuertas. A continuación, se muestra la fórmula para calcular el estado oculto h_t y la celda de memoria c_t en el instante de tiempo t en una celda LSTM:

$$
\begin{aligned}
f_t &= \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right) \\
i_t &= \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right) \\
\tilde{c}_t &= \tanh\left(W_c \cdot [h_{t-1}, x_t] + b_c\right) \\
c_t &= f_t \cdot c_{t-1} + i_t \cdot \tilde{c}_t \\
o_t &= \sigma\left(W_o \cdot [h_{t-1}, x_t] + b_o\right) \\
h_t &= o_t \cdot \tanh(c_t)
\end{aligned}
$$

Donde:
- h_t es el estado oculto en el instante de tiempo t.
- c_t es la celda de memoria en el instante de tiempo t.
- x_t es la entrada en el instante de tiempo t.
- σ es la función sigmoide, que produce valores en el rango $[0, 1]$.
- W_f, W_i, W_c, W_o son las matrices de pesos de las compuertas.
- b_f, b_i, b_c, b_o son los sesgos de las compuertas.

- f_t: Compuerta de olvido. Controla cuánta información de la celda de memoria anterior c_{t-1} se descartará.

- i_t: Compuerta de entrada. Controla cuánta información nueva debe agregarse a la celda de memoria actual.

- $\tilde{c}_t$: Celda de memoria candidata. Es una candidata a ser agregada a la celda de memoria c_t.

- o_t: Compuerta de salida. Controla cuánta información de la celda de memoria c_t se usará para calcular la salida h_t.

El proceso de entrenamiento de una red LSTM implica ajustar los pesos y sesgos (W_f, W_i, W_c, W_o y b_f, b_i, b_c, b_o) para que la red pueda capturar patrones a largo plazo en las secuencias y hacer predicciones precisas en el conjunto de entrenamiento. Sin embargo el proceso de entrenamiento se va a profundizar más adelante

5.3.2. Regresión no paramétrica

La regresión no paramétrica es un enfoque más flexible y menos restrictivo en comparación con la regresión paramétrica, ya que no asume una forma funcional específica para la relación entre las variables. En lugar de ello, se permite que el modelo se adapte a la estructura de los datos, sin imponer restricciones predefinidas Robinson (2008).

En términos matemáticos, la regresión no paramétrica puede ser expresada de la siguiente manera general:

$$y = f(x) + \varepsilon$$

Donde y es la variable dependiente, x es el vector de variables independientes, $f(x)$ es una función desconocida, que representa la relación entre las variables, y ε es un término de error aleatorio. En lugar de asumir una forma funcional específica para $f(x)$, el objetivo es

estimar esta función directamente a partir de los datos de entrenamiento sin restricciones predefinidas Robinson (2008).

El término "no paramétrico"puede ser confuso porque no significa que los modelos carezcan completamente de parámetros. En realidad, se refiere a la flexibilidad en el número de parámetros y a la capacidad de adaptarse a los datos.

En un enfoque no paramétrico, el número de parámetros no está fijo de antemano. En cambio, la cantidad de parámetros puede variar y ajustarse según las características de los datos Mahmoud (2019).

Regresión por K vecinos más cercanos (k-NN Regression):

En el método k-NN para regresión, se hace una predicción para un punto de prueba, basándose en los valores de la variable objetivo (dependiente) de los **k** vecinos, más cercanos a ese punto en el espacio de las características.

Formalmente, supongamos que tenemos un conjunto de puntos de entrenamiento con sus respectivas características, que denotaremos como X, y un vector correspondiente de valores de la variable objetivo, que denotaremos como Y. Dado un punto de prueba x, queremos predecir el valor de la variable objetivo, que denotaremos como $\hat{y}$. La predicción $\hat{y}$ se calcula de la siguiente manera: Bielecki and Śmiałek (2022)

$$\hat{y} = \frac{1}{k} \sum_{i=1}^{k} Y_i$$

Aquí, Y_i representa los valores de la variable objetivo para los **k** vecinos más cercanos a x. En otras palabras, se encuentra los k puntos en X que están más cerca de x, se observa sus correspondientes valores en Y, y luego se promedia estos valores. Este promedio es la predicción $\hat{y}$ para el punto de prueba x.

Este método se basa en la idea de que los puntos que están cerca en el espacio de las características también deberían estar cerca en términos de su variable objetivo. Es decir, si dos puntos tienen características muy similares, también deberían tener valores similares de la variable objetivo.

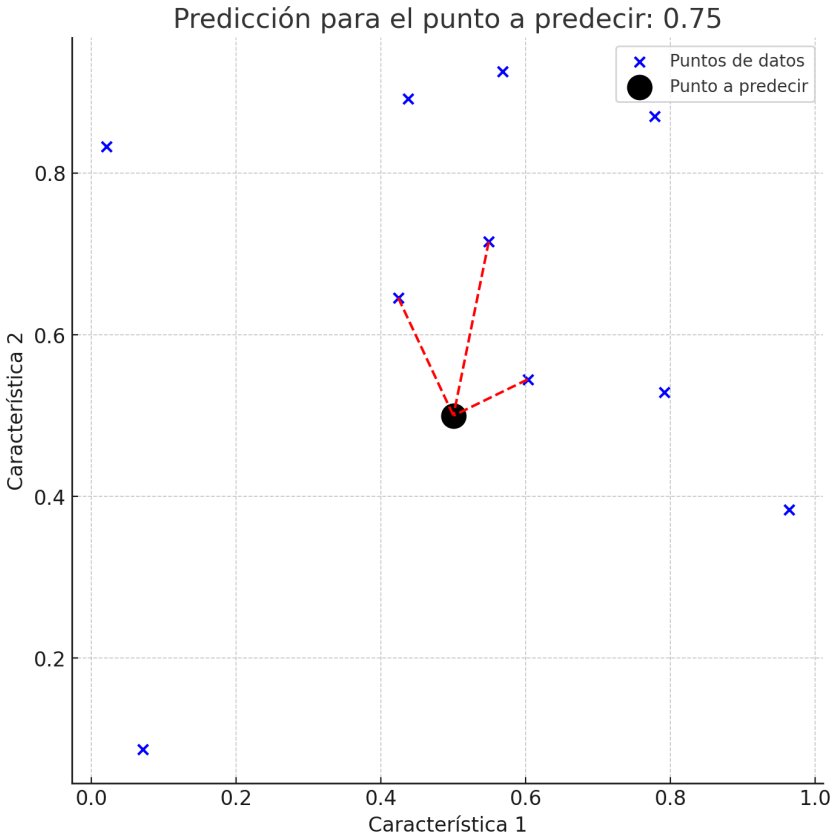

Figura 5.11: Ejemplo de una predicción gráfica con KNN

Por ejemplo en la figura 5.11, cada x representa una observación en el conjunto de datos. Las coordenadas de cada punto en el espacio de dos dimensiones corresponden a los valores de dos características diferentes (Característica 1 y Característica 2).

El punto negro representa la nueva observación para la cual queremos hacer una predicción. Sus coordenadas también se basan en los valores de las dos características.

Las líneas punteadas conectan el nuevo punto con sus tres vecinos más cercanos, que son los tres puntos que están más cerca del nuevo punto, en términos de las dos características. En otras palabras, estos son los tres puntos cuyos valores de Característica 1 y Característica 2 son más similares a los del nuevo punto.

El algoritmo de k-NN para regresión determina estos vecinos más cercanos y luego promedia sus valores de la variable objetivo (no mostrados en el gráfico) para hacer la predicción para el nuevo punto. En este caso, la predicción para el nuevo punto es aproximadamente 0.75, como se indica en el título del gráfico.

Por lo tanto, este gráfico nos muestra cómo el algoritmo de k-NN utiliza las distancias en el espacio de las características para determinar qué puntos de datos son "similares" entre sí, y cómo utiliza esta información para hacer predicciones.

Regresión por suavizado de kernel (Kernel Regression):

La regresión con kernel es un método de aprendizaje automático no paramétrico que se utiliza en este trabajo para predecir un valor continuo para un punto de prueba. Este método emplea una función de ponderación, conocida como kernel, para asignar un peso a cada punto de entrenamiento basado en su distancia al punto de prueba.

Se considera un conjunto de puntos de entrenamiento con sus respectivas características, denotadas como X, junto con un vector correspondiente de valores de la variable objetivo, denotado como Y. Para un punto de prueba dado, denotado como x, el objetivo es predecir el valor de la variable objetivo, denotado como $\hat{y}$.

La predicción $\hat{y}$ en la regresión con kernel se calcula de la siguiente manera:

$$\hat{y} = \frac{\sum_{i=1}^{n} K\left(\frac{x-X_i}{h}\right) Y_i}{\sum_{i=1}^{n} K\left(\frac{x-X_i}{h}\right)}$$

Aquí, K representa la función de kernel, que asigna una ponderación a cada punto de entrenamiento en función de su distancia al punto de prueba. La variable h es el parámetro de suavizado o ancho de banda, que controla el grado de influencia de los puntos más distantes en la predicción. X_i y Y_i son los valores de las características y de la variable objetivo para el i-ésimo punto de entrenamiento, respectivamente.

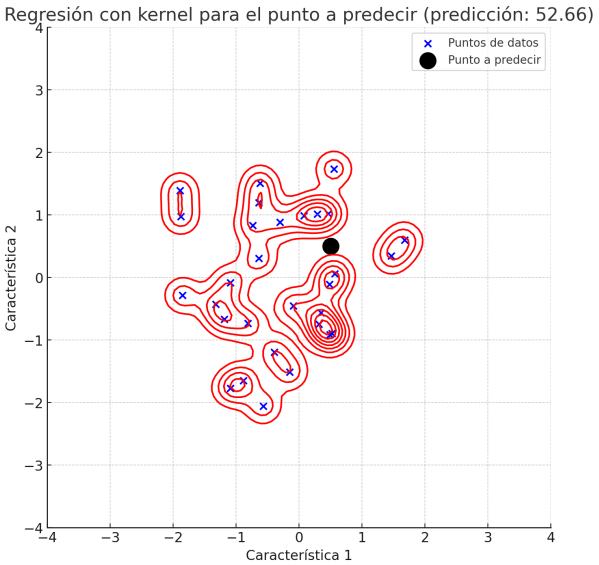

Figura 5.12: Ejemplo de una predicción utilizando Regresión por suavizado de kernel

En la figura 5.12, las líneas rojas representan las curvas de nivel de la función de densidad de kernel. Esta función de densidad se calcula utilizando los puntos de datos y la función de kernel (en este caso, una función de kernel gaussiano).

El punto negro representa la nueva observación para la que queremos hacer una predicción.

Las x representan las observaciones en el conjunto de datos. Las coordenadas de cada punto en el espacio de dos dimensiones corresponden a los valores de dos características diferentes.

La idea principal de la regresión con kernel es que el valor de la variable objetivo para el punto de prueba (el punto negro) se estima como un promedio ponderado de los valores de la variable objetivo de los puntos de datos, donde las ponderaciones son proporcionadas por la función de densidad de kernel.

Los puntos que están más cerca del punto de prueba (es decir, los puntos que están dentro de las curvas de nivel más internas) tendrán una mayor influencia en el valor predicho para el punto de prueba, mientras que los puntos que están más lejos (es decir, los puntos que están en las curvas de nivel más externas) tendrán una influencia menor.

El título del gráfico muestra la predicción de la variable objetivo para el punto de prueba. En este caso, la predicción es aproximadamente 52.66.

Regresión local:

La Regresión Local, también conocida como LOESS o LOWESS (LOcal regrESSion), es un método de regresión que busca ajustar una función localmente a cada punto de prueba utilizando un modelo polinómico. Al igual que la regresión con kernel, la regresión local asigna un peso a cada punto de entrenamiento basado en su proximidad al punto de prueba, pero a diferencia de la regresión con kernel, la regresión local ajusta un modelo distinto en cada punto de prueba.

Formalmente, se considera un conjunto de puntos de entrenamiento con sus respectivas características, denotadas como X, junto con un vector correspondiente de valores de la variable objetivo, denotado como Y. Para un punto de prueba dado, denotado como x, el objetivo es predecir el valor de la variable objetivo, denotado como $\hat{y}$. Se busca ajustar un modelo polinómico de la forma:

$$f(x) = \beta_0 + \beta_1 \cdot x + \beta_2 \cdot x^2 + \ldots + \beta_d \cdot x^d$$

Los coeficientes β se determinan minimizando la suma de los cuadrados residuales ponderados, donde las ponderaciones dependen de la distancia entre el punto de prueba y cada punto de entrenamiento. La predicción $\hat{y}$ se obtiene evaluando el modelo ajustado en el punto de prueba x.

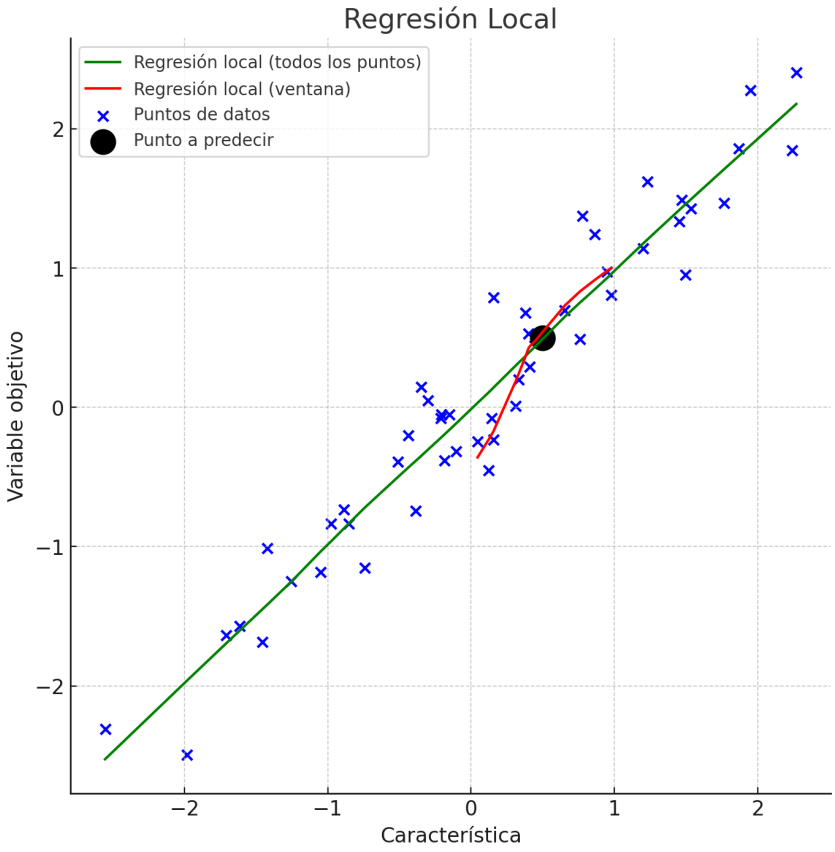

Figura 5.13: Ejemplo de una predicción utilizando Regresión Local

En la figura 5.13, se tiene puntos de datos representados con x de color azul, y un punto de prueba representado como un punto negro grande.

Además, se tiene dos líneas de regresión local. La primera línea (verde) se calcularía utilizando todos los puntos de datos. Esta línea brinda una idea general de la tendencia de los datos, pero podría no captar las variaciones locales con precisión.

La segunda línea de regresión (roja) se calcularía utilizando solo los puntos de datos dentro de la "ventana" alrededor del punto de prueba. Esta línea sería potencialmente más precisa para predecir el valor de la variable objetivo en el punto de prueba, ya que se basaría en datos que son localmente relevantes para el punto de prueba.

Por lo tanto, al observar el gráfico, se puede ver cómo la regresión local se ajusta más estrechamente a los puntos de datos locales, lo que podría dar como resultado una predicción más precisa para el punto de prueba.

Regresión por árboles de decisión (Decision Tree Regression):

En este método, se construye un árbol de decisión dividiendo el espacio de características en regiones. Cada región representa una hoja del árbol y se asigna un valor de respuesta promedio a los puntos de entrenamiento que caen en algunas de esas regiones.

Se construye un árbol de decisión mediante particiones recursivas del espacio de características. En cada nodo interno, se elige una variable y un valor de corte para dividir el espacio en dos subespacios. En cada hoja, se asigna un valor de respuesta promedio basado en los puntos de entrenamiento que caen en esa hoja Loh (2011).

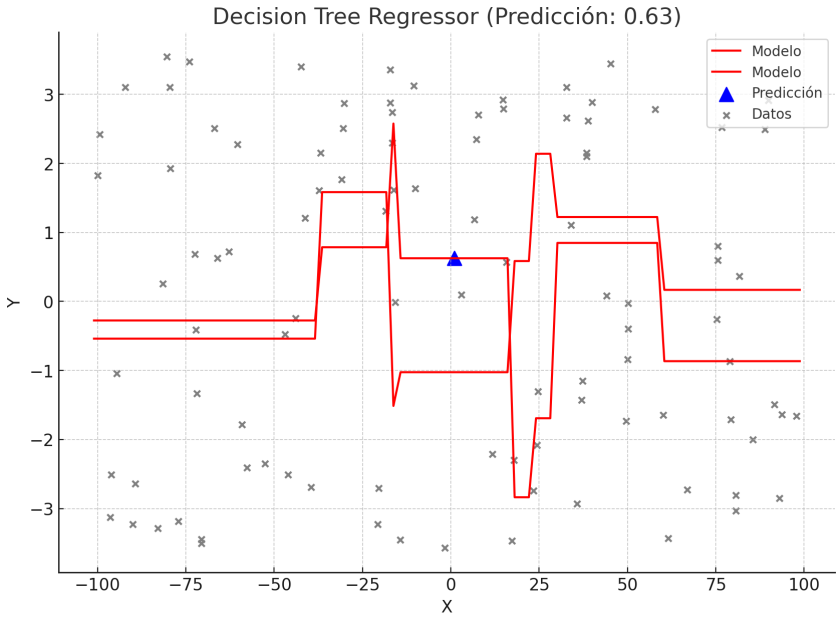

Figura 5.14: Ejemplo de una predicción utilizando Árboles de decisión

En el gráfico 5.14 se muestran los datos de entrenamiento (puntos grises), el modelo de regresión del árbol de decisión (línea roja), y la predicción para un nuevo punto (punto azul).

Los puntos grises representan los datos de entrenamiento. Cada uno es un par de características (X, Y). La línea roja es el modelo de regresión del árbol de decisión. Representa la predicción del modelo para cualquier valor dado de X. El punto azul representa la

predicción del modelo para un nuevo punto. La posición en el eje Y de este punto corresponde al valor de la predicción. El título del gráfico muestra el valor de la predicción para el nuevo punto.

Es importante tener en cuenta que la línea roja no es suave porque un árbol de decisión realiza predicciones basadas en divisiones binarias de los datos. Por lo tanto, la predicción cambia abruptamente en los valores de X donde el modelo hace una nueva división.

Además, dado que se trata de un modelo de regresión, las predicciones pueden ser cualquier número real, en lugar de estar limitadas a un conjunto de categorías como en un árbol de decisión de clasificación.

***Random forest* como extensión de los árboles de decisión:**

El *Random Forest* es una potente extensión de los árboles de decisión que busca mejorar el rendimiento y precisión del modelo. A diferencia de un solo árbol de decisión, el *Random Forest* utiliza el concepto de "ensamblado de modelos" para lograr una mayor generalización y prevenir el sobreajuste.

En lugar de depender de un único árbol, el *Random Forest* crea un conjunto de árboles independientes, cada uno entrenado con muestras de datos seleccionadas aleatoriamente y con una selección aleatoria de atributos. La predicción final del *Random Forest* se obtiene mediante el promedio (para regresión) o la votación (para clasificación) de las salidas de todos los árboles individuales.

La idea clave es que aunque cada árbol puede cometer errores, en promedio, los errores se compensan entre sí, lo que lleva a una mayor capacidad de generalización. Esta técnica reduce la varianza y mejora la estabilidad del modelo en comparación con un solo árbol de decisión.

El *Random Forest* es especialmente útil para problemas con datos ruidosos o características altamente correlacionadas. También es capaz de manejar grandes conjuntos de datos y proporcionar medidas de importancia de atributos, que ayudan a identificar las características más influyentes en las predicciones.

Máquinas de vectores de soporte para regresión (Support Vector Machines Regression):

En la regresión SVM, se busca encontrar un hiperplano en el espacio de características que se ajuste mejor a los puntos de entrenamiento.

Formalmente, se considera un conjunto de puntos de entrenamiento con sus respectivas características, denotadas como X, junto con un vector correspondiente de valores de la variable objetivo, denotado como Y. Para un punto de prueba dado, denotado como x, el objetivo es predecir el valor de la variable objetivo, denotado como $\hat{y}$.

La función objetivo que se busca minimizar es:

$$\text{mín} \left(\sum_{i=1}^{N} \epsilon_i + C \sum_{i=1}^{N} \text{máx} \left(0, |y_i - f(x_i)| - \epsilon_i\right) \right)$$

sujeto a $y_i - f(x_i) \leq \epsilon_i$ y $f(x_i) - y_i \leq \epsilon_i$ para todos los puntos de entrenamiento (x_i, y_i).

Aquí, ϵ_i es la tolerancia de error permitida para cada predicción, C es el parámetro de regularización que controla el equilibrio entre el ajuste del modelo y la complejidad del

modelo, y $f(x_i)$ es la predicción del modelo para el punto de entrenamiento x_i. La función máx$(0, |y_i - f(x_i)| - \epsilon_i)$ es la función de pérdida insensible al épsilon, que no penaliza errores menores que ϵ_i.

La predicción $\hat{y}$ se obtiene evaluando $f(x)$ en el punto de prueba x.

En resumen, la regresión SVM busca encontrar el hiperplano en el espacio de características que minimiza la suma de las desviaciones de las predicciones del modelo de los valores reales, donde las desviaciones menores a una cierta tolerancia no son penalizadas. Mammone et al. (2009).

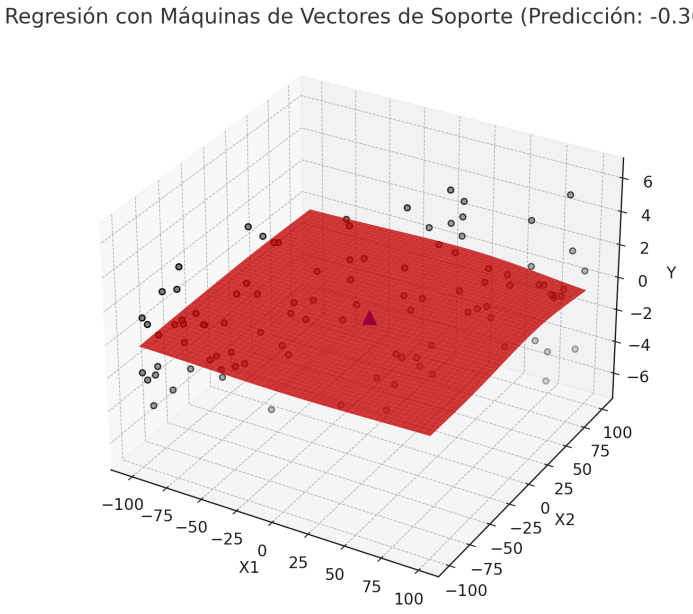

Figura 5.15: Ejemplo de una predicción utilizando SVM

La figura 5.15 representa un gráfico tridimensional que muestra una regresión SVM en dos dimensiones de entrada (*X1* y *X2*) y una dimensión de salida (*Y*).

Los puntos grises representan los datos de entrenamiento. Cada uno es un trío de características (*X1*, *X2*, *Y*). La superficie roja es el modelo de regresión SVM. Representa la predicción del modelo para cualquier par de valores dados de (*X1*, *X2*). El punto azul representa la predicción del modelo para un nuevo punto. Su posición en el eje Y corresponde al valor de la predicción. El título del gráfico muestra el valor de la predicción para el nuevo punto.

Es importante tener en cuenta que la superficie roja no es plana porque SVM es capaz de modelar relaciones no lineales entre las características de entrada y la variable objetivo. En este caso, se ha utilizado un kernel RBF (Radial Basis Function), que permite a SVM aprender relaciones no lineales complejas.

Además, dado que este es un modelo de regresión, las predicciones pueden ser cualquier número real, en lugar de estar limitadas a un conjunto de categorías como en SVM para clasificación.

5.4 Selección de los algoritmos de regresión a utilizar y justificación

En este trabajo se decidió usar cuatro tipos de regresores, un Regresor Lineal, *Random Forest regresor*, una Red Neuronal Secuencial y una Red Neuronal LSTM. Esta elección se basó principalmente en las fortalezas que tiene cada uno de estos regresores. Para luego evaluar a partir de métricas que se explicarán más adelante, cuál de ellos presenta mejores resultados.

Regresión lineal:

1. **Interpretabilidad:** La regresión lineal proporciona coeficientes de regresión que indican la relación lineal entre las variables predictoras y la variable objetivo Olive (2017). Esto permite interpretar el impacto relativo de cada variable en la predicción.

2. **Eficiencia computacional:** La regresión lineal es computacionalmente eficiente, especialmente en conjuntos de datos más pequeños, lo que la hace adecuada para aplicaciones con recursos limitados Olive (2017).

3. **Interpretación de efectos marginales:** La regresión lineal permite analizar los efectos marginales, es decir, cómo un cambio en una variable predictora afecta a la variable objetivo, manteniendo las demás variables constantes Olive (2017).

***Random forest*:**

1. **Flexibilidad y robustez:** El *Random Forest* puede manejar tanto variables categóricas como numéricas, y es menos sensible a los valores atípicos y a los problemas de multicolinealidad en comparación con otros modelos de regresión. En este contexto, como se describió anteriormente existen algunas variables categóricas como el tipo de procesador y el sistema operativo Liu et al. (2012).

2. **No linealidad:** El *Random Forest* puede capturar relaciones no lineales entre las variables predictoras y la variable objetivo mediante la construcción de múltiples árboles de decisión y su combinación Liu et al. (2012).

3. **Capacidad de manejo de alta dimensionalidad:** El *Random Forest* puede manejar conjuntos de datos con un gran número de variables predictoras sin necesidad de realizar una selección manual de características Liu et al. (2012).

Red neuronal secuencial:

1. **Capacidad de modelado no lineal:** Las redes neuronales secuenciales son capaces de aprender y modelar relaciones no lineales complejas entre las variables predictoras y la variable objetivo Aggarwal (2018).

2. **Adaptabilidad y flexibilidad:** Las redes neuronales secuenciales pueden adaptarse a diferentes tipos de datos y problemas, y pueden ajustar sus parámetros internos durante el entrenamiento para mejorar el rendimiento del modelo Aggarwal (2018).

3. **Captura de interacciones complejas:** Las redes neuronales secuenciales pueden aprender y representar interacciones complejas entre las variables predictoras, incluso cuando no se conocen *a priori*. Aggarwal (2018).

Red neuronal LSTM

- **Memoria a largo plazo:** Las LSTM son capaces de aprender y recordar información a lo largo de secuencias largas, lo que puede ser útil si el tiempo de ejecución de una tarea depende de las características de la tarea y del recurso en pasos de tiempo anteriores Van Houdt et al. (2020).
- **Capacidad para manejar secuencias de longitud variable:** Las LSTM pueden manejar secuencias de longitud variable, lo que es útil si el número de pasos de tiempo anteriores que son relevantes para predecir el tiempo de ejecución puede variar Van Houdt et al. (2020).
- **Modelado de dependencias temporales complejas:** Las LSTM pueden modelar dependencias temporales complejas, incluyendo patrones que pueden ser difíciles de capturar con modelos de regresión tradicionales Van Houdt et al. (2020).
- **Capacidad para manejar grandes cantidades de datos:** Al igual que otras Redes Neuronales Secuenciales, las LSTM pueden manejar grandes cantidades de datos de entrenamiento y pueden ser capaces de extraer patrones complejos a partir de estos datos Van Houdt et al. (2020).

En Iverson et al. (1999) se utiliza una regresión no paramétrica para determinar el tiempo de ejecución, sin embargo, a raíz de que es computacionalmente complejo considerar todas las variables independientes en el modelo, optan por sólo considerar algunas.

Estas variables independientes, sin embargo, pueden afectar directamente la medida del error estimado. En este artículo se decide utilizar el método KNN como regresor no paramétrico para realizar las predicciones. Este regresor es muy bueno debido a su simplicidad y fácil interpretación. Sin embargo, debido a su enfoque basado en la distancia, k-NN puede tener dificultades para capturar relaciones complejas entre las variables.

Para abordar estas limitaciones, es posible aplicar técnicas de preprocesamiento de datos, como la normalización de variables o la reducción de dimensionalidad, pero éstos elementos no se describen en el artículo.
El método de validación utilizado a grandes rasgos en el estudio, implica realizar experimentos utilizando datos reales y analizar el rendimiento de los algoritmos en diferentes configuraciones y condiciones.

Luego se evalúa el rendimiento en términos de error de predicción, eficiencia computacional y otras métricas relevantes para determinar la capacidad de los algoritmos para estimar el tiempo de ejecución en un entorno de cómputo distribuido heterogéneo.

Sin embargo, el uso del método de validación en el estudio presenta algunas críticas y limitaciones que vale la pena mencionar:

- **Generalización limitada**: Los resultados obtenidos a partir de los conjuntos de datos específicos utilizados en el estudio pueden no ser generalizables a otras situaciones o dominios. La validez de los resultados puede depender en gran medida de la selección de los conjuntos de datos y las condiciones de los experimentos.

- **Sesgo en la selección de conjuntos de datos**: La selección de los conjuntos de datos utilizados en el estudio puede estar sesgada hacia ciertos tipos de problemas o escenarios. Esto puede limitar la aplicabilidad de los resultados a situaciones diferentes a las que se han estudiado.
- **Simplificaciones y suposiciones**: El estudio puede hacer suposiciones y simplificaciones que podrían no reflejar completamente la complejidad del mundo real. Por ejemplo, los algoritmos pueden basarse en ciertas premisas o modelos que podrían no ser aplicables en todos los casos.
- **Dependencia del contexto**: El rendimiento de los algoritmos puede variar dependiendo del contexto específico en el que se utilicen. Factores como la configuración del sistema, el conjunto de datos y las condiciones de ejecución pueden influir en los resultados y no se pueden generalizar fácilmente a otros contextos.
- **Limitaciones en la evaluación del rendimiento**: El estudio puede centrarse en métricas específicas de rendimiento, como el error de predicción o el tiempo de CPU, pero puede no abordar otros aspectos importantes, como la robustez, la escalabilidad o la interpretabilidad de los algoritmos.

Otras técnicas de validación como, la validación cruzada no se implementan. La validación cruzada es una técnica de evaluación más robusta y recomendada en muchos casos, en lugar de la validación simple mencionada anteriormente. La validación cruzada aborda algunas de las limitaciones de la validación simple al proporcionar una estimación más precisa del rendimiento del modelo Refaeilzadeh et al. (2009).

En la validación cruzada, se divide el conjunto de datos en k subconjuntos más pequeños llamados "folds". Luego, se realiza el proceso de entrenamiento y evaluación k veces, cada vez utilizando un *fold* diferente como conjunto de prueba y los restantes $k-1$ *folds* como conjunto de entrenamiento. Los resultados se promedian para obtener una medida general del rendimiento del modelo.

La validación cruzada proporciona una estimación más robusta del rendimiento del modelo al utilizar todos los datos disponibles tanto para entrenamiento como para prueba. Esto ayuda a reducir la dependencia en una única partición de los datos y a obtener una evaluación más representativa del rendimiento general del modelo Refaeilzadeh et al. (2009).

Sin embargo, también hay algunas críticas y consideraciones en el uso de la validación cruzada, como el aumento del costo computacional al realizar k iteraciones y la posibilidad de sobreajuste si no se utiliza correctamente Refaeilzadeh et al. (2009).

En Iverson et al. (1999) se describe además, que hay relaciones no lineales en los paramétros o características predictoras, que pueden afectar directamente el error a la hora de predecir el tiempo. Por tanto, como se explicó anteriormente se decidió utilizar en uno de los modelos un regresor no parámetrico como el *Random Forest*, y tres regresores paramétricos: Redes Neuronales Secuenciales Simples, Redes LSTM, y un Regresor Lineal Múltiple. Las Redes Neuronales Secuenciales Simples y la Red LSTM se utilizan para poder estimar relaciones más complejas entre las variables predictoras y la variable objetivo, por último el Regresor Lineal Múltiple servirá de contraste con los demás modelos.

5.5 Detección de valores atípicos

La detección de valores atípicos es una parte importante del análisis descriptivo de datos. Los valores atípicos, también conocidos como *outliers*, son observaciones que se encuentran significativamente lejos de los demás puntos de datos en un conjunto de datos. Estos valores pueden ser el resultado de errores de medición, errores de entrada o simplemente representar eventos raros o inusuales en el fenómeno que estamos estudiando Agarwal (2013).

La presencia de valores atípicos puede afectar significativamente los resultados del análisis estadístico y del modelado. Por lo tanto, es importante detectarlos y manejarlos adecuadamente. Existen diversas técnicas y métodos para la detección de valores atípicos:

- **Estadísticos descriptivos:** Se pueden usar medidas como el rango intercuartílico (IQR) y los límites de corte para identificar valores atípicos basados en la dispersión de los datos Agarwal (2013).
- **Gráficos de caja y bigote:** Los diagramas de caja y bigote proporcionan una representación visual de la distribución de los datos y ayudan a identificar valores atípicos como puntos fuera de los límites de los bigotes Agarwal (2013). En la figura 5.16 se muestra como se vería un gráfico de cajas y bigotes
- **Pruebas estadísticas:** Se pueden realizar pruebas estadísticas, como la prueba de Grubbs, la prueba Z y la prueba de Dixon, para detectar valores atípicos en función de desviaciones estadísticas Agarwal (2013).
- **Algoritmos de detección de anomalías:** Existen algoritmos más avanzados, como el algoritmo Isolation Forest y el algoritmo de vecinos más cercanos (k-NN), que pueden identificar valores atípicos basados en patrones de comportamiento anormal Agarwal (2013) Liu et al. (2009).

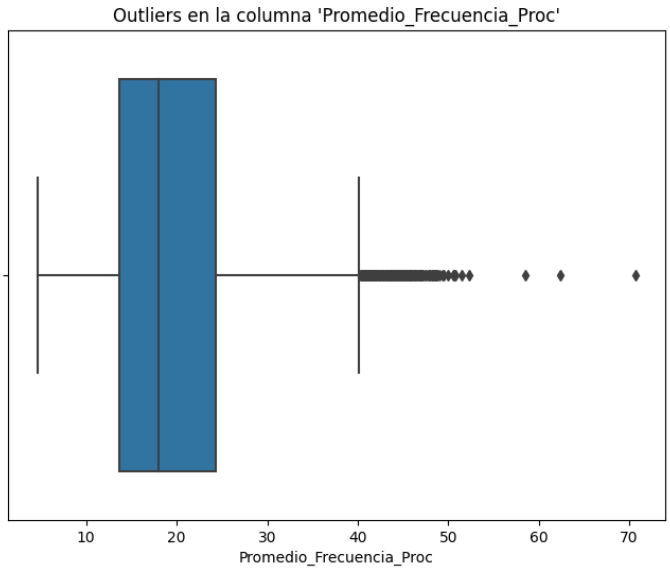

Figura 5.16: Ejemplo de un gráfico de caja y bigote

5.5.1. Interpretación de gráficos de caja y bigotes

Los diagramas de caja y bigotes, también conocidos como *boxplots*, son una herramienta gráfica útil para visualizar la distribución de un conjunto de datos y para identificar posibles valores atípicos. Un *boxplot* muestra la distribución de los datos a través de cinco estadísticas resumidas: el valor mínimo, el primer cuartil (Q1), la mediana (Q2), el tercer cuartil (Q3) y el valor máximo. El diagrama se compone de una caja que representa el rango intercuartílico (IQR) y dos "bigotes"que se extienden desde la caja hasta los valores mínimo y máximo, excluyendo los valores atípicos Perner (2002).

La interpretación de un boxplot es la siguiente:

- La caja representa el rango intercuartílico (IQR), que abarca desde el primer cuartil (Q1) hasta el tercer cuartil (Q3). La longitud de la caja muestra la variabilidad intercuartílica de los datos.
- La línea dentro de la caja representa la mediana (Q2), que es el valor que divide el conjunto de datos en dos partes iguales.
- Los bigotes se extienden desde la caja hasta los valores mínimo y máximo dentro del rango de 1.5 × IQR 1.5×IQR. Cualquier valor fuera de este rango se considera un valor atípico y se muestra como un punto fuera de los bigotes.
- Los valores atípicos se muestran como puntos individuales fuera de los bigotes. Estos puntos representan valores que están significativamente alejados del resto de los datos y pueden requerir una atención especial en el análisis.

5.5.2. Pruebas estadísticas para la detección de valores atípicos

Además de los gráficos de caja y bigotes, existen pruebas estadísticas que se pueden utilizar para detectar valores atípicos en un conjunto de datos. Algunas de las pruebas más comunes son Moore et al. (2012):

- **Prueba de Grubbs:** Esta prueba se basa en la hipótesis nula de que no hay valores atípicos en el conjunto de datos. Se compara el valor más extremo con la media y la desviación estándar de los demás datos para determinar si es estadísticamente significativamente diferente y, por lo tanto, puede considerarse un valor atípico .
- **Prueba Z:** Esta prueba se basa en la distribución normal de los datos y se utiliza para detectar valores que están lejos de la media en términos de desviaciones estándar. Los valores que se encuentran fuera de un umbral establecido se consideran valores atípicos.
- **Prueba de Dixon:** Esta prueba se utiliza para detectar un solo valor atípico en un conjunto de datos pequeño (generalmente menos de 30 observaciones). Compara el valor más extremo con los valores cercanos para determinar si es un valor atípico significativo.

5.5.3. Algoritmos de detección de anomalías

Los algoritmos de detección de anomalías son métodos avanzados que utilizan técnicas de aprendizaje automático para identificar observaciones inusuales o atípicas en un conjunto de datos. Estos algoritmos se basan en el principio de que las anomalías son puntos que difieren significativamente del patrón general del conjunto de datos y, por lo tanto, pueden ser tratados como instancias raras o poco probables.

En el contexto de la detección de valores atípicos, se considera que los algoritmos de detección de anomalías son especialmente útiles cuando los datos son complejos y no se pueden representar fácilmente por una distribución paramétrica. A diferencia de las pruebas estadísticas tradicionales, que dependen de supuestos sobre la distribución de los datos, los algoritmos de detección de anomalías son más flexibles y pueden adaptarse a diferentes estructuras de datos.

Uno de los algoritmos más utilizados para la detección de anomalías es el *Isolation Forest* Liu et al. (2009), que es un algoritmo basado en árboles de decisión. Este algoritmo construye un conjunto de árboles de decisión de forma aleatoria y busca aislar instancias inusuales en regiones separadas del espacio de características. La idea principal es que las observaciones normales se agrupan más densamente en el espacio de características, mientras que las observaciones anómalas son más aisladas y requieren menos particiones para ser separadas. La puntuación de anomalía asignada a cada observación se basa en el número de particiones necesarias para aislarla, lo que indica su grado de anormalidad.

Otro algoritmo comúnmente utilizado es el algoritmo de vecinos más cercanos (k-NN) Ramaswamy et al. (2000). En este enfoque, cada observación se clasifica en función de la distancia a sus vecinos más cercanos. Las observaciones que tienen una mayor distancia promedio a sus vecinos se consideran más anómalas. El parámetro k controla la cantidad de vecinos que se consideran en la clasificación, y se puede ajustar según las características específicas del conjunto de datos.

Es importante mencionar que el rendimiento de estos algoritmos de detección de anomalías depende en gran medida de la calidad del conjunto de datos y de la elección adecuada de parámetros. Por lo tanto, se recomienda realizar una exploración exhaustiva y experimentación con diferentes configuraciones para obtener los mejores resultados.

5.6 Análisis de correlación

El análisis de correlación es una técnica utilizada para explorar la relación entre dos o más variables en un conjunto de datos. La correlación cuantifica la fuerza y la dirección de la relación entre las variables, lo que permite identificar patrones y tendencias en los datos Agarwal (2013).

Existen diferentes coeficientes de correlación que se utilizan según el tipo de variables que se están analizando:

Correlación de Pearson:

La correlación de *Pearson* es una medida que evalúa la relación lineal entre dos variables continuas. Su valor está en el rango de -1 a 1, donde:

- 1 indica una correlación positiva perfecta, es decir, cuando una variable aumenta, la otra también lo hace proporcionalmente.

- -1 indica una correlación negativa perfecta, es decir, cuando una variable aumenta, la otra disminuye proporcionalmente.
- 0 indica que no hay correlación lineal, es decir, las variables no están relacionadas de manera lineal.

La fórmula para calcular la correlación de Pearson entre dos variables X e Y con n observaciones es:

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

donde r_{xy} es el coeficiente de correlación de Pearson, x_i y y_i son los valores de las variables X e Y respectivamente, $\bar{x}$ y $\bar{y}$ son las medias de X e Y respectivamente.

Correlación de Spearman:

La correlación de *Spearman* se utiliza cuando las variables no siguen una distribución normal o cuando la relación entre ellas es monótona en lugar de lineal. Esta correlación evalúa la relación entre los rangos de las observaciones en lugar de los valores originales.

La fórmula para calcular la correlación de *Spearman* entre dos variables X e Y con n observaciones es:

$$\rho = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)}$$

donde ρ es el coeficiente de correlación de *Spearman*, d_i es la diferencia entre los rangos de las observaciones X_i e Y_i y n es el número total de observaciones.

Correlación de Kendall:

La correlación de *Kendall* también se utiliza para variables no lineales y es especialmente útil para datos categóricos o de rango. Al igual que la correlación de Spearman, esta correlación evalúa la relación entre los rangos de las observaciones.

La fórmula para calcular la correlación de Kendall entre dos variables X e Y con n observaciones es:

$$\tau = \frac{\text{Número de pares concordantes} - \text{Número de pares discordantes}}{\text{Número total de pares posibles}}$$

donde τ es el coeficiente de correlación de Kendall.

Estos coeficientes de correlación permiten cuantificar y comprender mejor la relación entre las variables en un conjunto de datos. Al utilizar estas herramientas en el análisis exploratorio, se pueden identificar patrones, tendencias y asociaciones entre las variables, lo que ayuda a tomar decisiones informadas y construir modelos predictivos más precisos Perner (2002).

5.7 Análisis de distribución de la variable objetivo

En los problemas de regresión, una parte esencial del análisis descriptivo es examinar la distribución de la variable objetivo que se está tratando de predecir. La distribución

de la variable objetivo proporciona información sobre la dispersión y la variabilidad de los valores de la variable, lo que es fundamental para entender cómo se comporta y qué técnicas de modelado son más apropiadas.

El análisis de distribución puede incluir la visualización de histogramas, gráficos de densidad y diagramas de caja y bigote. Estas representaciones gráficas nos permiten identificar si la variable objetivo sigue una distribución normal o si tiene valores atípicos que podrían afectar el rendimiento de los modelos de regresión.

Además, el análisis de distribución también puede ayudar a identificar posibles transformaciones que podrían mejorar la distribución de la variable objetivo y hacerla más adecuada para el modelado.

En la figura 5.17 se puede visualizar un histograma que muestra la distribución de una variable hipotética. Los datos se dividen en intervalos, y la altura de cada barra representa la frecuencia de los datos en ese intervalo.

En este gráfico, los intervalos de la variable están en el eje X, y la frecuencia de los datos en cada intervalo está en el eje Y. Por ejemplo, se puede observar que hay 7 observaciones en el intervalo de 8 a 10, 29 observaciones en el intervalo de 10 a 12, y así sucesivamente.

La interpretación de este gráfico permite entender cómo se distribuyen los datos de la variable. Por ejemplo, se puedes ver que la mayoría de las observaciones se encuentran en los intervalos de 14 a 16 y de 16 a 18. Esto indica que estos son los valores más comunes para esta variable.

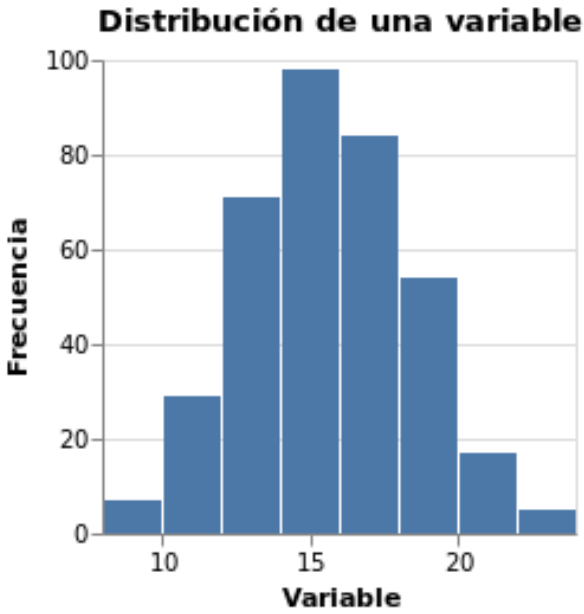

Figura 5.17: Ejemplo de la distribución de una variable con un histograma

5.8 Preprocesamiento de datos

5.8.1. Codificación

La codificación es el proceso de convertir datos categóricos en un formato que los modelos de aprendizaje automático puedan entender. Los métodos más comunes de codificación incluyen:

Codificación One-Hot: Este método, también conocido como codificación *dummy*, convierte cada categoría en una nueva columna y asigna un 1 (verdadero) si la categoría está presente y un 0 (falso) si no lo está. Es fácil de implementar y funciona bien con algoritmos de aprendizaje automático que no asumen ninguna relación de orden entre las categorías. Sin embargo, puede crear un gran número de columnas si la variable categórica tiene muchas categorías, lo que puede aumentar la dimensionalidad de los datos y hacer que el modelo sea más lento y más difícil de entrenar Hastie et al. (2001).

Codificación Ordinal: Este método convierte las categorías en números enteros. Es útil cuando las categorías tienen un orden natural (por ejemplo, "bajo", "medio", "alto"). Sin embargo, si no hay un orden natural, la codificación ordinal puede introducir una relación de orden que no existe, lo que puede afectar negativamente al rendimiento del modelo Hastie et al. (2001).

Codificación de Frecuencia o Conteo: Estas técnicas reemplazan las categorías por su frecuencia o conteo en el conjunto de datos. Pueden ser útiles para manejar variables categóricas con muchas categorías, y pueden capturar información sobre la distribución de las categorías en los datos. Sin embargo, pueden introducir sesgos si la frecuencia o el conteo están correlacionados con la variable objetivo Hastie et al. (2001).

Codificación Binaria: Este método convierte cada categoría en un código binario: cada dígito del código binario se convierte en una columna separada. Puede manejar eficientemente variables categóricas con muchas categorías, ya que crea menos columnas adicionales que la codificación *one-hot*. Sin embargo, puede ser más difícil de interpretar que otros métodos de codificación Hastie et al. (2001).

Target Encoding: Este método, también conocido como *Mean Encoding*, reemplaza cada categoría con la media de la variable objetivo para esa categoría. Puede ser muy útil cuando se trata de variables categóricas con alta cardinalidad, ya que no aumenta la dimensionalidad de los datos como lo hace la codificación *One-Hot*. Además, incorpora información sobre la variable objetivo, lo que puede resultar en un mejor rendimiento del modelo si la relación entre la categoría y la variable objetivo es fuerte. Sin embargo, puede llevar a un sobreajuste si algunas categorías tienen muy pocas observaciones, y si no se maneja correctamente, puede introducir fugas de datos. Al reemplazar las categorías con la media de la variable objetivo, se pierde toda la información sobre la frecuencia de las categorías Hastie et al. (2001).

5.8.2. Normalización

La normalización es el proceso de escalar los datos numéricos para que todos los atributos tengan un rango similar. Los métodos más comunes de normalización incluyen:

- Normalización Min-Max: Este método escala los datos para que estén en el rango de 0 a 1. La fórmula para la normalización Min-Max es:

$$X_{\text{norm}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}}$$

  donde X_{norm} es el valor normalizado, X es el valor original, y X_{min} y X_{max} son los valores mínimo y máximo de la característica, respectivamente.

- Estandarización (Z-score): Este método escala los datos para que tengan una media de 0 y una desviación estándar de 1. La fórmula para la estandarización es:

$$Z = \frac{X - \mu}{\sigma}$$

donde Z es el valor estandarizado, X es el valor original, μ es la media de la característica, y σ es la desviación estándar de la característica.

- Escalado por el valor máximo: Este método escala los datos dividiendo cada valor por el valor máximo. La fórmula para el escalado por el valor máximo es:

$$X_{\text{scaled}} = \frac{X}{X_{\max}}$$

donde X_{scaled} es el valor escalado, X es el valor original, y $X_{\max}$ es el valor máximo de la característica.

5.9 Entrenamiento del *Random Forest*

El proceso de entrenamiento del *Random Forest* de regresión se puede dividir en los siguientes pasos:

Generación del conjuntos de datos mediante bootstrap aggregating (*Bagging*):

El *Random Forest* utiliza la técnica de Bootstrap Aggregating (Bagging) para generar múltiples conjuntos de datos de entrenamiento. Cada conjunto se crea mediante muestreo con reemplazo a partir del conjunto de datos original.

Para cada conjunto de datos de entrenamiento k:

Seleccionar N instancias mediante muestreo con reemplazo: $x_{i1}, x_{i2}, ..., x_{iN}$.

Donde:

k: Índice que representa cada conjunto de datos de entrenamiento generado mediante Bagging.

N: Número de instancias en cada conjunto de datos de entrenamiento.

x_{ij}: Instancia i en el conjunto de datos de entrenamiento j. Cada instancia representa una observación individual dentro del conjunto de datos original.

En este contexto cada instancia representa una entidad única, que contiene información sobre varias características o variables, o sea, una fila del conjunto de datos.

El muestreo con reemplazo es una técnica utilizada en estadística e inferencia que consiste en seleccionar muestras de un conjunto de datos, permitiendo que una misma instancia se seleccione más de una vez en una misma muestra.

Cuando se realiza el muestreo con reemplazo, después de seleccionar una instancia para formar parte de una muestra, se vuelve a colocar en el conjunto original antes de seleccionar la siguiente instancia. Esto significa que cada instancia tiene la posibilidad de ser seleccionada múltiples veces o incluso no ser seleccionada en absoluto.

La principal característica del muestreo con reemplazo es que cada selección se realiza de manera independiente de las selecciones anteriores. Esto implica que la probabilidad de que una instancia se seleccione en una muestra particular es la misma en cada selección, lo que puede resultar en la repetición de instancias en la muestra.

Construcción de árboles de decisión:
Se construye un conjunto de árboles de decisión independientes, cada uno entrenado con un conjunto de datos de entrenamiento generado mediante el proceso de *Bagging*. Cada árbol se entrena de forma individual utilizando un subconjunto aleatorio de características.

Selección aleatoria de características:

Antes de construir cada árbol de decisión, se realiza una selección aleatoria de un subconjunto de características del conjunto de datos original.Esta selección aleatoria ayuda a diversificar los árboles y evitar la dependencia de un subconjunto particular de características en la construcción de los árboles. El número de características seleccionadas aleatoriamente se elige como un parámetro del modelo y se conoce como *mtry* o *maxfeatures*. Luego utilizando el conjunto de datos de entrenamiento generado mediante Bagging y el subconjunto aleatorio de características seleccionadas, se construye un árbol de decisión.

La construcción del árbol sigue el algoritmo estándar de construcción de árboles de decisión, aunque con alguna diferencia en la forma en que se manejan las variables continuas, para este caso el tiempo.
A continuación se describe el proceso:

- **Paso 1: Selección de la mejor característica para dividir los datos:**
  - Para cada característica, se evalúa su capacidad para reducir la varianza o el error en las predicciones.
  - Se pueden utilizar diferentes medidas de impureza, como la reducción de la varianza o el error cuadrático medio.
  - La característica que minimice la varianza o el error se selecciona como la mejor característica para dividir los datos.
- **Paso 2: División de los datos en subconjuntos:**
  - Los datos se dividen en subconjuntos basados en los valores de la mejor característica seleccionada.
  - En el caso de variables continuas, se establece un umbral o punto de corte para realizar la división.
  - Cuando se trata de una variable continua, el algoritmo evalúa diferentes puntos de corte posibles y selecciona aquel que minimiza la varianza o el error en la

predicción. La idea es encontrar el punto de corte que mejor separa los datos y reduce la variabilidad dentro de cada subconjunto resultante.

- Por ejemplo, en este caso se tiene una variable continua como el tiempo y al realizar una división en un nodo del árbol. El algoritmo evaluará diferentes puntos de corte posibles en la escala del tiempo y seleccionará aquel que minimice la varianza o el error en la predicción.
- Los datos se dividirán en dos subconjuntos: uno con valores de tiempo por debajo del umbral y otro con valores de tiempo por encima del umbral.

- **Paso 3: Repetición de los pasos 1 y 2 para cada subconjunto:**
  - Para cada subconjunto creado en el paso anterior, se repiten los pasos 1 y 2 para seleccionar la mejor característica y realizar una nueva división.
  - Este proceso se repite de forma recursiva hasta que se cumpla alguna condición de parada, como alcanzar una profundidad máxima o tener un número mínimo de instancias en un nodo.

- **Paso 4: Asignación de una etiqueta de valor promedio a cada hoja:**
  - Una vez que se han creado todos los nodos del árbol, se asigna una etiqueta de valor promedio a cada hoja.
  - En el caso de la regresión, se calcula el promedio de los valores de la variable objetivo que en este caso es el tiempo, correspondientes a las instancias en cada hoja y se asigna como la etiqueta de valor para esa hoja.

Paso 5: Podar el árbol (opcional):
Opcionalmente, se puede realizar una poda del árbol para evitar el sobreajuste. La poda implica eliminar nodos o fusionar ramas para mejorar la generalización del árbol.

Sin embargo, en la regresión del *Random Forest*, la poda no es necesaria. En la mayoría de los casos se utiliza comúnmente en árboles de decisión para evitar el sobreajuste y mejorar la generalización del modelo. Sin embargo, en el *Random Forest*, al utilizar múltiples árboles de decisión y promediar sus predicciones, el efecto del sobreajuste se reduce significativamente.

El *Random Forest* de regresión tiene la capacidad de calcular la importancia de las características durante el proceso de entrenamiento del modelo. Esta información puede ser utilizada para evaluar la relevancia y contribución de cada característica en la predicción del resultado.

La importancia de las características se calcula utilizando medidas como la ganancia de información o la reducción de la impureza en los árboles de decisión individuales que componen el *Random Forest*. Estas medidas se utilizan para determinar qué características son las más influyentes en la predicción y, por lo tanto, las más importantes. Lo cual resulta de gran importancia para este trabajo ya que permite conocer una medida de cómo influyeron ciertas variables en la predicción del tiempo.

Predicción en *Random Forest*:

Una vez que se han construido todos los árboles de decisión, se utiliza el proceso de votación para realizar las predicciones. En el caso del *Random Forest* de regresión, la predicción final se obtiene promediando las predicciones de todos los árboles.

5.10 Entrenamiento de la red neuronal secuencial

El proceso de entrenamiento de una red neuronal secuencial, también conocida como red neuronal feedforward, se puede dividir en varias etapas. A continuación, se explica detalladamente cada una de ellas:

1. **Inicialización de pesos**: Antes de comenzar el entrenamiento, se inicializan los pesos de la red neuronal de manera aleatoria o utilizando algún otro método. Sea $W^{[l]}$ la matriz de pesos para la capa l de la red neuronal, y $b^{[l]}$ el vector de sesgos para la capa l. Los pesos y los sesgos se inicializan típicamente con valores pequeños cercanos a cero Khuri (2013).

2. **Propagación hacia adelante (Forward Propagation)**: Durante la propagación hacia adelante, se calculan las salidas de cada neurona capa por capa. Para una capa oculta l y una capa de salida L:

   Para una capa oculta l:
   $$Z^{[l]} = W^{[l]} A^{[l-1]} + b^{[l]}$$
   $$A^{[l]} = g^{[l]}(Z^{[l]})$$

   Para la capa de salida L:
   $$Z^{[L]} = W^{[L]} A^{[L-1]} + b^{[L]}$$
   $$A^{[L]} = g^{[L]}(Z^{[L]})$$

   Donde $A^{[l]}$ es la matriz de activaciones para la capa l, $Z^{[l]}$ es la matriz de entradas ponderadas para la capa l, y $g^{[l]}$ es la función de activación utilizada en la capa l.

   Funciones de activación no lineales:

   En los modelos de regresión utilizando redes neuronales, se suelen utilizar funciones de activación no lineales en las capas ocultas para permitir al modelo aprender relaciones complejas y no lineales en los datos.

   Función de activación ReLU (Rectified Linear Unit): Esta función es ampliamente utilizada en redes neuronales debido a su simplicidad y eficacia Aggarwal (2018). La función ReLU devuelve el valor de entrada si es mayor que cero, y cero en caso contrario. Puede representarse como:
   $$f(x) = \text{máx}(0, x)$$

   Función de activación sigmoide: La función sigmoide es una función no lineal que genera una salida en el rango [0, 1]. Se utiliza comúnmente en problemas de regresión binaria. Puede representarse como:
   $$f(x) = \frac{1}{1 + e^{-x}}$$

Función de activación tanh (tangente hiperbólica): La función tanh es otra función no lineal que genera una salida en el rango [-1, 1]. Se utiliza en problemas de regresión cuando se desea tener una salida con valores negativos y positivos Aggarwal (2018). Puede representarse como:

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Estas funciones de activación no lineales introducen la capacidad de aprendizaje no lineal en la red neuronal, permitiéndole capturar patrones y relaciones complejas en los datos de regresión Aggarwal (2018).

3. **Cálculo de la función de pérdida**: Una vez que se obtienen las salidas de la red $A^{[L]}$, se calcula la función de pérdida que mide la discrepancia entre las salidas predichas y los valores reales. Para un problema de regresión, la función de pérdida comúnmente utilizada es el error cuadrático medio (MSE) :

$$L = \frac{1}{m} \sum_{i=1}^{m} (y^{(i)} - \hat{y}^{(i)})^2$$

Donde m es el número de ejemplos de entrenamiento, $y^{(i)}$ es el valor real del ejemplo i, y $\hat{y}^{(i)}$ es el valor predicho por la red neuronal para el ejemplo i.

4. **Retropropagación del error (Backpropagation)**: En la etapa de retropropagación, el error calculado en el paso anterior se propaga hacia atrás a través de la red neuronal para ajustar los pesos y los sesgos. La idea principal es calcular las derivadas parciales del error con respecto a los pesos y los sesgos utilizando la regla de la cadena Aggarwal (2018).

   - L: Representa la capa de salida de la red neuronal.
   - l: Representa una capa oculta genérica en la red neuronal.
   - $dZ^{[l]}$: Es el gradiente del costo con respecto a la entrada lineal $Z^{[l]}$ de la capa l.
   - $dW^{[l]}$: Es el gradiente del costo con respecto a los pesos $W^{[l]}$ de la capa l.
   - $db^{[l]}$: Es el gradiente del costo con respecto al sesgo $b^{[l]}$ de la capa l.
   - $A^{[l]}$: Es la salida de la capa l después de aplicar la función de activación.
   - $g'^{[l]}$: Es la derivada de la función de activación utilizada en la capa l.
   - m: Es el número de ejemplos en el conjunto de entrenamiento.

   Ecuaciones:

$$dZ^{[L]} = \frac{\partial L}{\partial A^{[L]}} \cdot g'^{[L]}(Z^{[L]})$$

Calcula el gradiente del costo con respecto a la entrada lineal de la capa de salida.

$$dW^{[L]} = \frac{1}{m} dZ^{[L]} A^{[L-1]T}$$

Calcula el gradiente del costo con respecto a los pesos de la capa de salida.

$$db^{[L]} = \frac{1}{m} \sum_{i=1}^{m} dZ^{[L](i)}$$

Calcula el gradiente del costo con respecto al sesgo de la capa de salida.

$$dZ^{[l]} = (W^{[l+1]T} dZ^{[l+1]}) \cdot g'^{[l]}(Z^{[l]})$$

Calcula el gradiente del costo con respecto a la entrada lineal de una capa oculta l.

$$dW^{[l]} = \frac{1}{m} dZ^{[l]} A^{[l-1]T}$$

$$db^{[l]} = \frac{1}{m} \sum_{i=1}^{m} dZ^{[l](i)}$$

Son similares a las ecuaciones para la capa de salida, pero aplicadas a una capa oculta l.

5. **Actualización de pesos**: Después de calcular las derivadas parciales, se actualizan los pesos y los sesgos utilizando un algoritmo de optimización. Los pesos y los sesgos se actualizan de la siguiente manera: Las ecuaciones:

$$W^{[l]} = W^{[l]} - \alpha dW^{[l]}$$
$$b^{[l]} = b^{[l]} - \alpha db^{[l]}$$

indican cómo se actualizan los pesos $W^{[l]}$ y sesgos $b^{[l]}$ de la capa l y α es la tasa de aprendizaje que controla la magnitud de los cambios realizados en cada actualización.

Algunos de los algoritmo más comunes para optimizar, son:

Descenso del gradiente estocástico (SGD): Es uno de los algoritmos más utilizados en el entrenamiento de redes neuronales. Se basa en actualizar los pesos de la red en cada iteración, utilizando la derivada del error con respecto a los pesos y una tasa de aprendizaje. El SGD se aplica en cada instancia de entrenamiento de forma aleatoria, lo que lo hace especialmente útil para conjuntos de datos grandes Aggarwal (2018).

Descenso del gradiente con momento (SGD con momento): Este algoritmo es una extensión del SGD que tiene en cuenta la dirección y magnitud de las actualizaciones anteriores de los pesos. Se utiliza un parámetro de momento que pondera las actualizaciones anteriores y les da una influencia en las actualizaciones actuales. El momento puede ayudar a acelerar el proceso de convergencia y evitar que el algoritmo quede atrapado en mínimos locales Aggarwal (2018).

RMSprop (Root Mean Square Propagation): Este algoritmo adapta la tasa de aprendizaje de forma individual para cada parámetro de la red neuronal. Utiliza una media móvil ponderada de los cuadrados de los gradientes anteriores para ajustar la tasa de aprendizaje. RMSprop es especialmente útil en problemas con gradientes esparsos Aggarwal (2018).

Adam (Adaptive Moment Estimation): Es un algoritmo de optimización que combina las ventajas del SGD con momento y RMSprop. Utiliza estimaciones adaptativas tanto del primer momento (media móvil ponderada de los gradientes) como del segundo momento (media móvil ponderada de los cuadrados de los gradientes) Aggarwal (2018).

5.11 Entrenamiento de la red LSTM

Las Redes LSTM son un tipo de red neuronal recurrente (RNN) que se utiliza para aprender secuencias de datos. Las RNN son útiles para aprender secuencias porque mantienen un "estado" que puede representar información de pasos de tiempo anteriores. Sin embargo, las RNN tradicionales tienen dificultades para aprender a largo plazo debido al problema de la desaparición del gradiente. Las LSTM resuelven este problema Van Houdt et al. (2020).

La arquitectura de una LSTM consta de una serie de "celdas" LSTM, cada una de las cuales tiene un estado interno (c) y una salida (h). Cada celda LSTM tiene tres "puertas" que controlan el flujo de información dentro y fuera de la celda: la puerta de entrada (*input gate*), la puerta de olvido (*forget gate*) y la puerta de salida (*output gate*) Van Houdt et al. (2020).

Aquí se presenta alguno de los conceptos útiles para luego entender su entrenamiento:

- **Puerta de olvido (Forget Gate)**: Decide qué información se va a descartar del estado de la celda. Se calcula como:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

  Donde σ es la función sigmoide, W_f son los pesos de la puerta de olvido, $[h_{t-1}, x_t]$ es la concatenación del estado oculto anterior y la entrada actual, y b_f es el sesgo de la puerta de olvido.

- **Puerta de entrada (Input Gate)**: Decide qué nueva información se va a almacenar en el estado de la celda. Se calcula como:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

  Donde W_i son los pesos de la puerta de entrada, y b_i es el sesgo de la puerta de entrada.

  Además, se calcula una actualización del estado de la celda:

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

Donde W_C son los pesos de la actualización del estado de la celda, y b_C es el sesgo de la actualización del estado de la celda.

- **Actualización del estado de la celda**: Combina la información de las puertas de olvido y entrada para actualizar el estado de la celda:

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t$$

- **Puerta de salida (Output Gate)**: Decide qué parte del estado de la celda se va a usar para generar la salida actual. Se calcula como:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

Donde W_o son los pesos de la puerta de salida, y b_o es el sesgo de la puerta de salida.

Finalmente, se calcula el estado oculto actual, que se usará para la salida de la celda LSTM y para la siguiente celda LSTM en la secuencia:

$$h_t = o_t \cdot \tanh(C_t)$$

El entrenamiento de una red LSTM es similar al entrenamiento de una red neuronal secuencial, en el sentido de que ambos utilizan la propagación hacia atrás, para calcular los gradientes y un algoritmo de optimización (como el descenso de gradiente estocástico) para actualizar los pesos de la red basándose en esos gradientes.

Sin embargo, hay algunas diferencias clave debido a la naturaleza recurrente de las LSTM:

- **Propagación hacia atrás a través del tiempo (BPTT)**: En una red neuronal secuencial, la propagación hacia atrás se realiza desde la última capa hasta la primera capa. En una LSTM, debido a que cada celda está conectada a través del tiempo, la propagación hacia atrás se realiza a través del tiempo. Esto significa que los gradientes se calculan y acumulan a lo largo de toda la secuencia antes de actualizar los pesos Van Houdt et al. (2020).

- **Estado oculto y estado de la celda**: Las LSTM mantienen un "estado oculto" y un "estado de la celda" que se pasan de una celda a la siguiente a lo largo de la secuencia. Estos estados permiten a la LSTM mantener la información a lo largo del tiempo y resolver el problema de la desaparición del gradiente que afecta a las RNN tradicionales Van Houdt et al. (2020).

- **Puertas de olvido, entrada y salida**: Las LSTM utilizan "puertas" que controlan el flujo de información dentro y fuera de cada celda. Estas puertas deciden qué información del estado de la celda se mantiene y cuál se descarta en cada paso de tiempo, lo que permite a la LSTM aprender dependencias a largo plazo Van Houdt et al. (2020).

A pesar de estas diferencias, el principio subyacente del entrenamiento - el uso de la propagación hacia atrás y un algoritmo de optimización para minimizar una función de pérdida - es el mismo en las redes LSTM y en las redes neuronales secuenciales Van Houdt

et al. (2020).

Sin embargo es importante señalar que las redes neuronales LSTM (Long Short-Term Memory) requieren que los datos de entrada estén en un formato específico. En particular, los datos de entrada a una LSTM deben ser un tensor[1] 3D cuyas dimensiones representan:

- **Muestras**: Esta es la dimensión que indexa las diferentes secuencias en el conjunto de datos. Cada muestra individual sería una secuencia.
- **Pasos de tiempo**: Esta es la dimensión que indexa los diferentes pasos de tiempo en cada secuencia.
- **Características**: Esta es la dimensión que indexa las diferentes características en cada paso de tiempo de cada secuencia.

Por lo tanto, si se está entrenando una LSTM en un conjunto de datos que consta de N secuencias individuales, cada una de las cuales tiene T mediciones de F variables diferentes, entonces el tensor de entrada tendría una forma de (N, T, F) Van Houdt et al. (2020).

Por ejemplo, en este caso que se está entrenando una LSTM para predecir el tiempo de ejecución de una tarea, basándose en las características mencionadas anteriormente la forma de los datos se pueden ver de la siguiente forma:

```
Muestra 1:
Observación: [tamaño de entrada, temperatura,
frecuencia, nivel de recursividad...]
Salida: tiempo de ejecución

Muestra 2:
Observación: [tamaño de entrada, temperatura,
frecuencia, nivel de recursividad...]
Salida: tiempo de ejecución

...
```

Cada "Observación" sería un paso de tiempo en la secuencia de entrada, y la "Salida" sería el valor que estás tratando de predecir.
En este caso solamente se tendría una observación por cada configuración distinta que se tenga de la tarea.

5.12 Entrenamiento de la Regresión lineal múltiple

A continuación se describe el proceso de entrenamiento de un regresor lineal múltiple:

1. Inicialización de los coeficientes: Se inicializan los coeficientes $w_0, w_1, w_2, ..., w_m$ con valores aleatorios o cero.

[1]Un tensor es una estructura de datos matemática generalizada que se extiende a conceptos como escalares, vectores y matrices. Un tensor puede ser pensado como un arreglo multidimensional de números, organizado en una cuadrícula que puede tener cualquier número de dimensiones.

2. Cálculo de las predicciones: Se calculan las predicciones $\hat{y}_i$ para cada muestra de entrenamiento i utilizando la fórmula de la regresión lineal múltiple:

$$\hat{y}_i = w_0 + w_1 x_{i1} + w_2 x_{i2} + ... + w_m x_{im}$$

donde x_{ij} es el valor de la variable independiente j para la muestra i.

3. Cálculo de la función de pérdida: Se calcula la función de pérdida, que mide la discrepancia entre las predicciones del modelo y los valores reales de la variable dependiente. Una función de pérdida comúnmente utilizada es MSE (error cuadrático medio):

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

donde n es el número total de muestras de entrenamiento y y_i es el valor real de la variable dependiente para la muestra i.

4. Actualización de los coeficientes: Se utilizan algoritmos de optimización, como el descenso del gradiente, para actualizar los coeficientes y minimizar la función de pérdida. En cada iteración, se actualiza cada coeficiente w_j (donde $j = 0, 1, 2, ..., m$) utilizando la regla de actualización del descenso del gradiente Khuri (2013):

$$w_j = w_j - \alpha \frac{\partial}{\partial w_j} MSE$$

donde α es la tasa de aprendizaje, que controla el tamaño de los pasos de actualización de los coeficientes.

5. Cálculo de las derivadas parciales: Se calculan las derivadas parciales de la función de pérdida con respecto a cada coeficiente w_j. La derivada parcial de la función de pérdida con respecto a w_j se calcula utilizando la regla de la cadena Khuri (2013):

$$\frac{\partial}{\partial w_j} MSE = \frac{2}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i) x_{ij}$$

donde x_{ij} es el valor de la variable independiente j para la muestra i.

6. Actualización de los coeficientes: Se actualizan los coeficientes utilizando las derivadas parciales calculadas:

$$w_j = w_j - \alpha \frac{2}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i) x_{ij}$$

Este proceso de actualización de los coeficientes se repite en cada iteración hasta que se alcance un criterio de convergencia, como el número máximo de iteraciones o la convergencia de la función de pérdida Khuri (2013).

Al finalizar el proceso de entrenamiento, se obtienen los valores óptimos de los coeficientes que proporcionan un mejor ajuste del modelo a los datos de entrenamiento. Estos coeficientes se pueden utilizar para realizar predicciones en nuevos datos utilizando la misma fórmula de regresión lineal múltiple.

Capítulo 6

Diseño del proyecto

En este trabajo se utilizaron 2 fases de manera general para poder estimar el rendimiento de las tareas. La primera fase radica en la recolección de datos, en donde se analizará el rendimiento de distintas tareas y el impacto que tuvo en varias unidades de procesamiento, midiendo algunas características del comportamiento del dispositivo en tiempo de ejecución.

La segunda, es la fase de entrenamiento del regresor determinado a partir de las características recopiladas en la primera. Dónde además, se pasarán a los datos obtenidos en la primera fase, por un proceso de análisis y preprocesamiento, antes del entrenamiento del regresor.

A continuación se procederá a explicar con más detalles cada una de las fases.

6.1 Primera fase

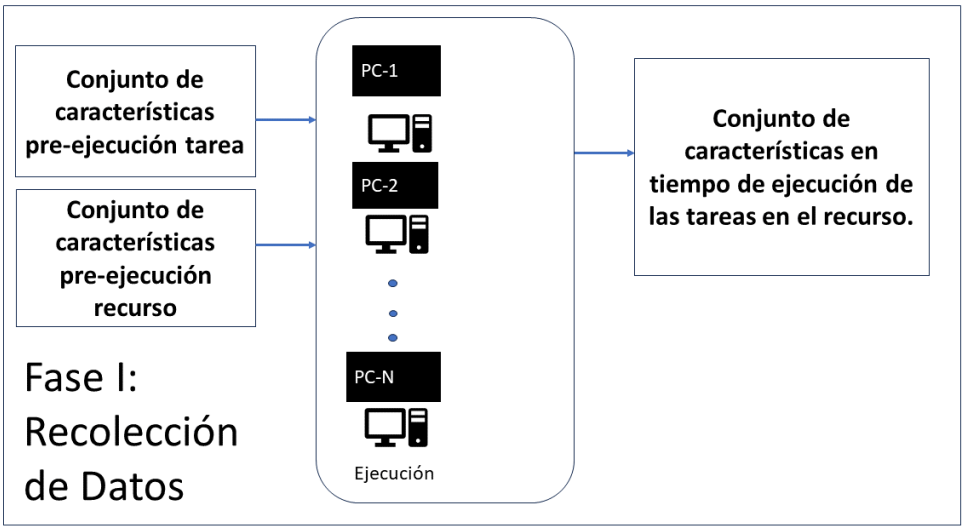

Figura 6.1: Fase 1: Recolección de datos

En la primera fase, como se explicó anteriormente, se analiza el rendimiento de un conjunto de tareas con distintas configuraciones o características pre-ejecución en varios

entornos o unidades de procesamiento. Monitoreando los parámetros de los distintos recursos computacionales en donde se ejecuten las tareas. De aquí se obtiene el tiempo de ejecución que tuvo la tarea con alguna configuración específica, y para cada configuración se obtiene, el gasto computacional que tuvo en el recurso determinado como se muestra en la figura 6.1.

Primeramente, se van a tener dos conjuntos de características. Las pre-ejecución o también llamadas configuraciones, para el caso de las tareas y las características pre-ejecución en la unidad de procesamiento específica. Luego en la ejecución se obtendrán mediciones de algunos parámetros que se explicarán luego.

6.1.1. Caracteríistización pre-ejecución de las tareas:

1. **N**: Tamaño del problema de la tarea. Por ejemplo en el algoritmo de Strassen sería el tamaño de la matriz. Se va a representar como un atributo numérico.

2. **Estrategia de paralelización**: Describe la estrategia de paralelización de las tareas. Se va a representar como un atributo nominal. Por ejemplo en el algoritmo de Strassen se puede utilizar un enfoque de paralelización usando OPENMP, CUDA, otro enfoque o una combinación de varios. Si no es paralelizable la tarea entonces esta característica obtuviera el valor de secuencial.

3. **Hilos**: En caso de usarse una estrategia de paralelización con OPenMP se indicará la cantidad de hilos OpenMP que se utilizaron. Se va a representar como un atributo numérico.

4. **Nivel de Recursividad**: En caso de que la tarea tuviera un planteamiento recursivo esta característica describirá el nivel de que posee la tarea. Por ejemplo el algoritmo de Strassen tiene un planteamiento recursivo y dependiendo del tamaño de la matriz se pudiera llegar a un nivel de recursividad específico. Se va a expresar como un atributo numérico.

5. **Uso de disco**: Describe si la tarea usa disco o no. Se va a representar como un atributo binario. En caso de que use disco toma el valor de 1 y en caso contrario 0.

Las distintas combinaciones de los valores de los parámetros descritos anteriormente representan configuraciones específicas a ejecutar para una tarea en las unidades de procesamiento. Por ejemplo una configuración específica a medir en la ejecución del algoritmo de Strassen podría ser:

N = 10000 elementos (en el caso del algoritmo de Strassen representa los tamaños de las matrices a multiplicar)

Nivel de recursividad = 2

Estrategia de paralelización= OPENMP

Hilos= 2.

6.1.2. Característización pre-ejecución de la unidad de procesamiento:

1. **Núcleos** : Describe la cantidad de núcleos físicos que va a tener el recurso. Se va a representar como un atributo numérico. tener cada núcleo. Se va a representar como un atributo numérico.

2. **Tipo de Procesador**: Describe el tipo de procesador que posee la unidad de procesamiento. Por ejemplo Intel Xeon X3360 Processor. Se va a representar como un atributo cualitativo.

3. **Sistema Operativo**: Describe el sistema operativo del recurso. Se va a representar como un atributo nominal.

4. **Tipo de Memoria Ram**: Describe el tipo de Memoria RAM que tiene la unidad de procesamiento. Por ejemplo DDR3. Se va a representar como un atributo cualitativo.

5. **Tipo de almacenamiento secundario**: Describe el almacenamiento secundario que posee la unidad de procesamiento en donde se va a ejecutar la tarea. Por ejemplo si es SATA, SSD, etc. Se va a representar como un atributo cualitativo.

   Esta caracterización va a representar el tipo de unidad de procesamiento en la que se va a ejecutar la tarea. Por ejemplo una posible comnbinación puede ser la siguiente:

   Núcleos=6
   Tipo de procesador= Intel Core i5 11400H
   Sistema Operativo= Ubuntu 22.04
   Tipo de Memoria Ram=DDR4
   Tipo de almacenamiento secundario= SSD
   Estas características corresponden a una laptop Asus TUF GAMING F15.

6.1.3. Características de monitoreo en ejecución:

1. **Tiempo de Ejecución**: Describe el tiempo de ejecución en segundos que tuvo la tarea en la unidad de procesamiento donde se ejecutó. Se va a representar como un valor numérico.

2. **Porciento de uso del procesador en general** : Describe el porcentaje de uso medio de todos los procesadores de la unidad de procesamiento que se obtuvo durante la ejecución de la tarea. Se va a tomar la media de las distintas mediciones que se obtuvieron. Se representa como un atributo numérico.

3. **Frecuencia del procesador** : Describe la frecuencia del procesador expresada en MHZ durante la ejecución de la tarea. Se va a tomar la media de las distintas mediciones que se obtuvieron. Se va a representar como un atributo numérico.

4. **Por ciento de uso de memoria** : Describe el por ciento que de memoria que usó la tarea con respecto a la memoria total del sistema. Se va a representar como un atributo numérico.

5. **Temperatura promedio**: Describe la temperatura en milésimas de grados celsius del procesador durante la ejecución de la tarea. Se va a tomar la media de las distintas mediciones que se obtuvieron. Se va a representar como un atributo numérico.

6. **Velocidad de lectura** : En caso de que la tarea utilice almacenamiento secundario, esta característica va a representar la velocidad de lectura en MB/s que se obtuvo durante la ejecución de la tarea. Se va a tomar la media de las distintas mediciones que se obtuvieron. Se va a representar como un atributo numérico.

7. **Velocidad de escritura**: En caso de que la tarea utilice almacenamiento secundario, esta característica va a representar la velocidad de lectura en MB/s que se obtuvo durante la ejecución de la tarea. Se va a tomar la media de las distintas mediciones que se obtuvieron. Se va a representar como un atributo numérico.

Al estarse monitoreando constantemente el porciento de uso del procesador, la frecuencia del procesador, la memoria usada, la temperatura, la velocidad de lectura y la velocidad de escritura durante el proceso de ejecución del conjunto de distintas configuraciones o características pre-ejecución de las tareas en la unidad de procesamiento, se van a obtener varias mediciones del comportamiento de esos parámetros. Por tanto hasta esta fase va a existir una separación entre los tiempos de ejecución obtenidos para cada una de las distintas configuraciones y las demás mediciones de los parámetros presentados en esta subsección.

Esto sucede ya que, para realizar las mediciones de los parámetros, se ejecutó un *script* que monitoreaba los valores que iban tomando, marcando la fecha exacta que se obtuvieron para cada medición.

En el caso del tiempo de ejecución de la tarea con la configuración específica también se marca una fecha exacta al momento de su inicio y al momento de su fin, lo que constituye un elemento importante para luego en otra fase relacionar las mediciones con cada configuración específica de la tarea.

En la figura 6.2 se muestra un ejemplo de ejecución de una tarea en una unidad de procesamiento con determinadas características. El proceso sería extensible para varias configuraciones de tareas, además se cambiará las unidades de procesamiento para recolectar los datos de ejecución de dichas tareas y poder medir entonces la influencia que tiene en el tiempo, las distintas propiedades de las unidades de procesamiento con las distintas configuraciones de la tarea.

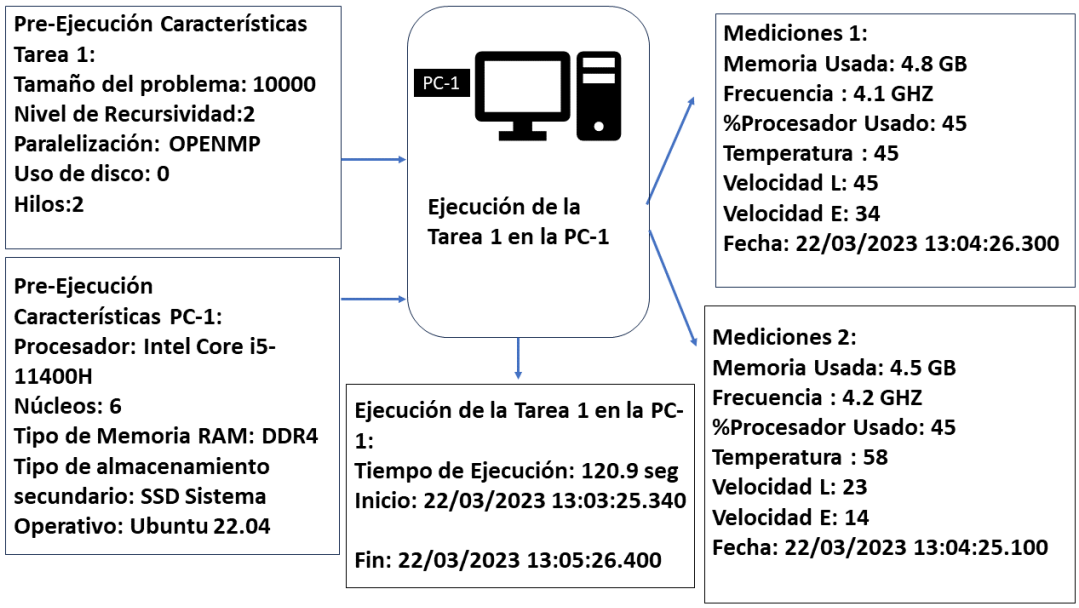

Figura 6.2: Análisis de Rendimiento

6.2 Segunda fase: Entrenamiento

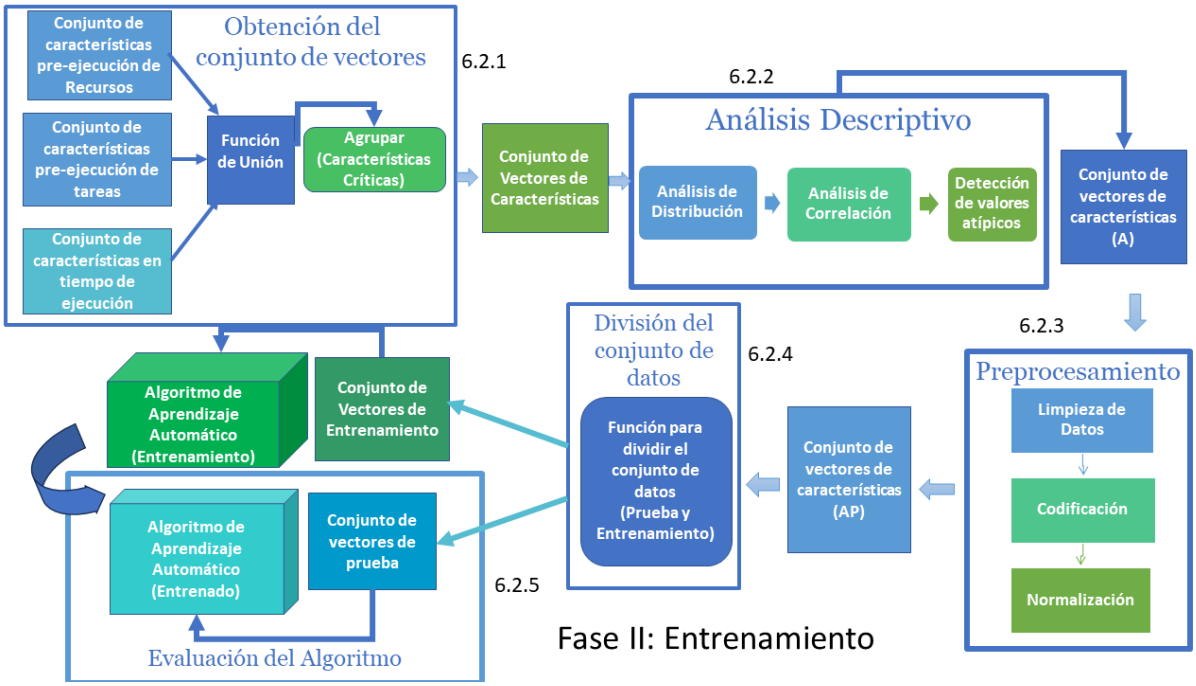

Figura 6.3: Fase 2: Entrenamiento

La segunda fase consiste en el entrenamiento de los modelos escogidos para realizar la regresión a partir de los datos recopilados en la primera. Sin embargo éstos datos obtenidos tienen que pasar por varios procesos antes de que puedan ser utilizados por el regresor en su entrenamiento.

6.2.1. Obtención del conjunto de vectores

El primer módulo dentro de la fase de entrenamiento sería la obtención del conjunto de vectores de características. Este vector representa la unión de cada configuración específica de la tarea junto a las características pre-ejecución de la unidad de procesamiento con las distintas mediciones que se obtuvieron en ese caso.

En la figura 6.2 se observa la ejecución de una tarea, con características específicas, en una unidad de procesamiento con características específicas, sin embargo este es un ejemplo reducido. La idea es tener varias configuraciones de tareas en varias unidades de procesamiento. Obteniéndose además varias mediciones registradas para cada unidad de procesamiento donde se ejecutó la tarea con alguna configuración específica.

El primer paso sería aplicar la función de unión, para poder obtener un vector único, que represente las características pre-ejecución de las tareas, las características pre-ejecución del recurso y las características durante la ejecución.

Para obtener el vector de unión se enlazan las mediciones obtenidas durante la ejecución total del experimento con la configuración específica de cada tarea. Cada medición hecha

durante la ejecución, viene acompañada de una estampa de tiempo que indica el momento exacto donde la medición fue hecha. Luego, cada configuración específica de la tarea tiene asociado el tiempo que se demoró, una estampa de tiempo para el inicio de la ejecución y una estampa de tiempo para el final de la ejecución, lo que indica la fecha en que empezó y la fecha en la que terminó.

Entonces para asociar una configuración específica de características pre-ejecución de una tarea y enlazarla con las mediciones específicas que se hicieron en el momento de su ejecución, las estampas de tiempo que indican el momento donde se realizaron cada una de las mediciones tienen que estar incluidas dentro del intervalo de tiempo en que comenzó a ejecutarse la configuración específica de la tarea y su terminación.

Una vez que se tengan enlazadas cada instancia específica de configuración con las distintas mediciones que se realizaron durante el intervalo que duró su ejecución, se obtiene el promedio de las distintas mediciones hechas. En el ejemplo de la figura 6.2 el promedio de temperatura sería 51.5, la cual representa la suma de la temperatura de la medición 1 con la temperatura de la medición 2 dividido entre la cantidad de mediciones que se obtuvieron para esa configuración de tarea.

Para explicar esto, asumamos que se tiene el algoritmo de Strassen como tarea y se obvia las velocidades de lectura y escritura ya que este tipo de tarea no hace uso de almacenamiento secundario. Además se tienen dos configuraciones específicas que se detallan a continuación: **Configuración 1**

- **N**: 8000
- **Nivel de recursividad**: 2
- **Hilos OpenMP**: 4
- **Estrategia de paralelización**: OpenMP
- **Uso de disco** : 0

Configuración 2

- **N**: 10000
- **Nivel de recursividad**: 3
- **Hilos OpenMP**: 6
- **Estrategia de paralelización**: CUDA
- **Uso de disco** : 0

Luego, se desea ejecutar dicho algoritmo con las dos configuraciones mencionadas anteriormente, en una computadora o unidad de procesamiento que tiene las siguientes características:

- **Núcleos**: 6
- **Tipo de procesador**: Intel Core i5 11400H

- **Sistema Operativo**: Ubuntu 22.04
- **Tipo de memoria RAM**: DDR4
- **Tipo de almacenamiento secundario**: SSD

La ejecución del algoritmo con la primera configuración tardó 80 segundos. Dicha ejecución empezó el 14/03/2023 a las 18:00:00.000 y terminó a el 14/03/2023 a las 18:01:20.000. La ejecución del algoritmo con la segunda configuración se demoró 120 segundos, y comenzó el 14/03/2023 a las 18:01:20.000 y terminó el 14/03/2023 a las 18:04:20.000 En el intervalo de tiempo que duraron las dos ejecuciones se registraron las siguientes mediciones:

Medición 1:

- **Estampa de tiempo**:14/03/2023 18:00:01:210
- **Por ciento de uso del procesador**: 8.3 %
- **Frecuencia del procesador**: 3.3 GHZ
- **Temperatura**: 31.3 grados Celcius
- **Por ciento de uso de memoria**: 4.4 %

Medición 2:

- **Estampa de tiempo**:14/03/2023 18:00:03:230
- **Por ciento de uso del procesador**: 10.3 %
- **Frecuencia del procesador**: 3.8 GHZ
- **Temperatura**: 24.3 grados Celcius
- **Por ciento de uso de memoria**: 9.3 %

Medición 3:

- **Estampa de tiempo**:14/03/2023 18:00:04:250
- **Por ciento de uso del procesador**: 41.3 %
- **Frecuencia del procesador**: 3.0 GHZ
- **Temperatura**: 23.4 grados Celcius
- **Por ciento de uso de memoria**: 6.2 %

Medición 4:

- **Estampa de tiempo**:14/03/2023 18:02:02:260
- **Por ciento de uso del procesador**: 43.3 %
- **Frecuencia del procesador**: 3.1 GHZ
- **Temperatura**: 28.2 grados Celcius

- **Por ciento de uso de memoria**: 4.3 %

Medición 5:

- **Estampa de tiempo**:14/03/2023 18:02:04:260
- **Por ciento de uso del procesador**: 49.3 %
- **Frecuencia del procesador**: 4.9 GHZ
- **Temperatura**: 56.2 grados Celcius
- **Por ciento de uso de memoria**: 2.3 %

En este caso se enlaza la configuración 1 con las mediciones 1,2 y 3 y la configuración 2 con la medición 4 y 5 ya que son las que coinciden con los intervalos de tiempo de cada configuración. Luego se obtiene el promedio de las mediciones que se obtuvieron para la configuración 1 y configuración 2, quedando de la siguiente forma.

Promedio de las mediciones de la configuración 1(mediciones 1,2,3)

- **Promedio Por ciento de uso del procesador**: 19.6 %
- **Promedio de frecuencia del procesador**: 3.36 GHZ
- **Promedio de temperatura**: 26.3 grados Celcius
- **Promedio de uso de memoria**: 6.63 %

Promedio de las mediciones de la configuración 2 (mediciones 4,5)

- **Promedio Por ciento de uso del procesador**: 46.35 %
- **Frecuencia del procesador**: 4.0 GHZ
- **Temperatura**: 42.2 grados Celcius
- **Promedio de uso de memoria**: 3.3

Por consiguiente, se generan 2 vectores que corresponden a la unión de la configuración 1, con las características de la unidad de procesamiento en donde se ejecutó y el promedio de las mediciones 1,2 y 3, de igual manera se hace para la configuración 2.

Vector 1:

- **N**: 8000
- **Nivel de recursividad**: 2
- **Hilos OpenMP**: 4
- **Estrategia de paralelización**: OpenMP
- **Uso de disco** : 0
- **Núcleos**: 6
- **Tipo de procesador**: Intel Core i5 11400H

- **Sistema Operativo**: Ubuntu 22.04
- **Tipo de memoria RAM**: DDR4
- **Tipo de almacenamiento secundario**: SSD
- **Promedio Por ciento de uso del procesador**: 19.6 %
- **Frecuencia del procesador**: 3.36 GHZ
- **Temperatura**: 26.3 grados Celcius
- **Promedio de uso de memoria**: 6.63

Vector 2:

- **N**: 10000
- **Nivel de recursividad**: 3
- **Hilos OpenMP**: 6
- **Estrategia de paralelización**: CUDA
- **Uso de disco** : 0
- **Núcleos**: 6
- **Tipo de procesador**: Intel Core i5 11400H
- **Sistema Operativo**: Ubuntu 22.04
- **Tipo de memoria RAM**: DDR4
- **Tipo de almacenamiento secundario**: SSD
- **Promedio Por ciento de uso del procesador**: 46.35 %
- **Frecuencia del procesador**: 4.0 GHZ
- **Temperatura**: 42.2 grados Celcius
- **Promedio de uso de memoria**: 3.3

Esto es un ejemplo sencillo, sin embargo el proceso es extensible, cuando se tienen varias unidades de procesamiento ejecutando varias configuraciones de tareas.

Agrupación de características críticas:

Luego de este paso, se realiza la agrupación de características críticas. Las características críticas identificadas representan parámetros que no todas las tareas incluyen, como la utilización de almacenamiento secundario, el nivel de recursividad y la estrategia de paralelización. En el caso del almacenamiento secundario no todas las tareas hacen uso de operaciones de lectura o escritura. Por lo tanto no resulta conveniente incluir dentro del vector de características generado para entrenar el algoritmo de regresión este tipo de parámetro. De manera similar pasa con el nivel de recursividad, ya que todas las

tareas no tienen un planteamiento recursivo. Por tanto esta característica no tiene una influencia directa en el tiempo de ejecución si la tarea en cuestión no tiene este tipo de planteamiento.

En el caso de la estrategia de paralelización funciona igual. Muchas tareas no son paralelizables o quizás no es de interés hacer uso de la paralelización. Además, resulta conveniente agrupar según la estrategia de paralelización utilizada para hacer que el regresor se especialice en predecir el tiempo según qué tipo de estrategia se usó.

Todo este proceso radica en hacer que el regresor sea un modelo lo más ajustado al tipo de características específicas que se introduzcan, para que sus predicciones sean lo más precisas posibles, sin que pierda generalidad el modelo. Este proceso haría que se obtengan varios conjuntos de regresores, en dependencia de la combinación de uso de las características críticas.

Por ejemplo si se identifican dentro del conjunto de datos, tareas que hacen uso de paralelización OPENMP, no usan disco, y tienen un planteamiento recursivo. Se agruparían las características y se enlazarían directamente con todos los regresores escogidos en la sección 5.4 para proceder al entrenamiento. En la figura 6.4 se evidencia este proceso. Es importante señalar que en las distintas resoluciones del árbol de la figura 6.4 se utilizarían todos los regresores escogidos, la única diferencia radicaría en la inclusión o no, de las características críticas, para luego realizar el entrenamiento de todos los regresores, .

Este proceso de agrupación haría que se obtuvieran varios conjuntos de datos, los cuales se diferencian, en la inclusión o no de las características críticas. Por lo que los regresores mencionados anteriormente, tomarían en cuenta para su entrenamiento las características obtenidas, luego de realizar la agrupación. Los conjuntos de datos obtenidos, luego de aplicar esta agrupación, pasarían por todas la estapas descritas en la figura 6.3. Por lo que dichas etapas son genéricas para cualquier conjunto de datos que se obtenga.

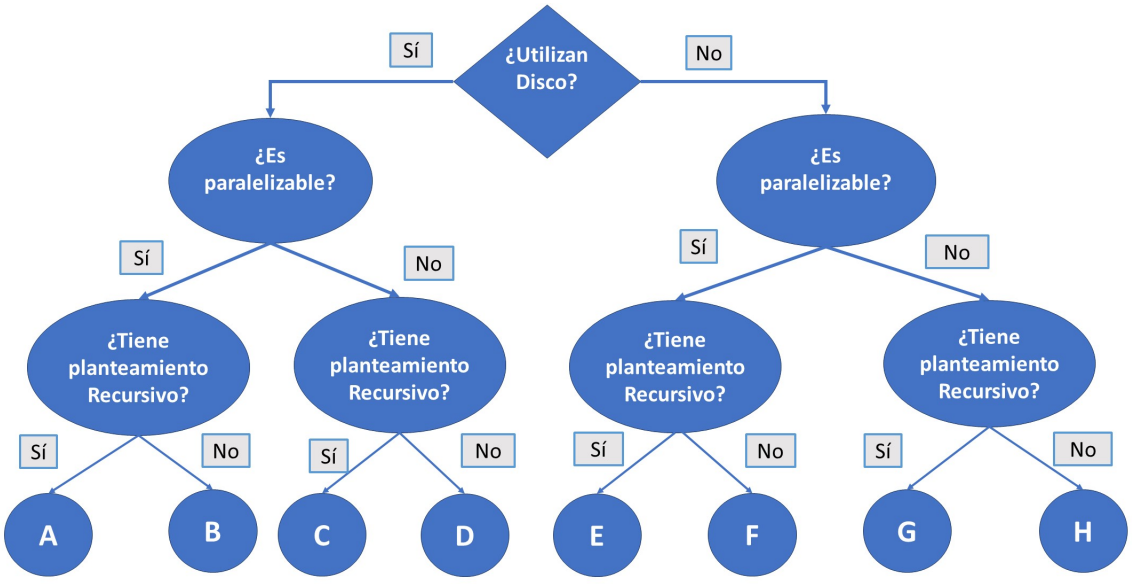

Figura 6.4: Agrupación de características críticas

6.2.2. Análisis descriptivo

El segundo módulo de la fase de entrenamiento es el análisis descriptivo que incluye, el análisis de distribución, el análisis de correlación y la detección de valores atípicos.

El análisis descriptivo es una etapa fundamental en cualquier análisis de datos y resulta importante por varias razones:

1. **Comprensión de los datos:** El análisis descriptivo permite comprender mejor la naturaleza de los datos que estamos trabajando. Nos da una visión general de cómo se distribuyen los datos, qué tendencias y patrones pueden existir, y cómo están relacionadas entre sí las diferentes variables Agarwal (2013).

2. **Detección de errores y anomalías:** Durante el análisis descriptivo, es posible identificar errores en los datos o valores atípicos que podrían ser errores de medición o simplemente datos inusuales. Identificar y corregir estos problemas es crucial para garantizar la calidad de los datos y la validez de los resultados del análisis Agarwal (2013).

3. **Selección de técnicas de análisis adecuadas:** El análisis descriptivo ayuda a determinar qué técnicas de análisis son más adecuadas para nuestros datos. Por ejemplo, si encontramos que hay una fuerte correlación entre ciertas variables, podríamos considerar utilizar técnicas de regresión para predecir una variable en función de otra. Si encontramos que los datos siguen una distribución normal, podríamos utilizar pruebas estadísticas paramétricas Perner (2002).

4. **Identificación de relaciones entre variables:** El análisis de correlación nos permite identificar qué variables están relacionadas entre sí y en qué grado. Esto es crucial para entender la estructura de los datos y para identificar posibles relaciones causales entre variables Perner (2002).

5. **Comunicación de resultados:** El análisis descriptivo proporciona una forma clara y concisa de comunicar los hallazgos y resultados del análisis de datos a otras personas. Las visualizaciones y resúmenes estadísticos que se generan durante el análisis descriptivo pueden ser utilizados para presentar de manera efectiva los resultados a audiencias no técnicas Perner (2002).

En resumen, el análisis descriptivo es una etapa crítica en cualquier proyecto de análisis de datos, ya que proporciona una visión general de los datos, ayuda a detectar problemas y errores, y nos permite seleccionar las mejores técnicas de análisis para obtener conclusiones significativas y útiles a partir de los datos.

Análisis de distribución:

El primer paso de interés, sería realizar un análisis de distribución de la variable objetivo. En el caso de la regresión, la suposición clave que a menudo se hace es sobre los residuos (errores) del modelo. Por ejemplo en una regresión lineal clásica, a menudo se asume que los residuos siguen una distribución normal Agarwal (2013). Esto es crucial, ya que la variable puede tener una distribución determinada, y entender esta distribución puede dar una idea del comportamiento de esa variable. Por ejemplo, si una variable tiene

una distribución normal (o gaussiana), eso dice que la mayoría de los valores están cerca de la media. Para el caso de este trabajo se verificará con una representación visual a través de un histograma distribución del tiempo.

Por ejemplo en la figura 6.5 se muestra la distribución de tiempo de usando una estrategia de parelización con CUDA y OpenMP, con planteamiento recursivo. La tarea utilizada para ejemplificar este diagrama fue el algoritmo de Strassen para la multiplicación de matrices y se ejecutó en una Jetson TX2 con un tipo de procesador ARM Cortex A52, almacenamiento secundario embedded MultiMediaCard 5.1 con velocidades de lectura secuencial de hasta 250MB/s y velocidades de escritura secuencial de hasta 125MB/s, una tarjeta de memoria de 8gb, y el sistema operativo que se utilizó fue Ubuntu 22.04. En la tabla 6.1 se muestra un ejemplificación de los datos. El eje y representa la frecuencia o densidad de los valores de la variable, y el eje x representa los valores que toma la variable del tiempo.

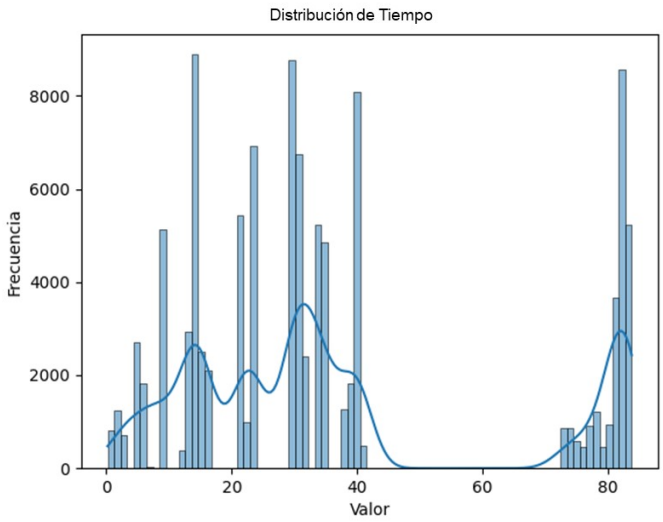

Figura 6.5: Distribución del tiempo en la variante CUDA con OpenMP recursivo

Detección de valores atípicos:

Por consiguiente, se pasaría a realizar el proceso de detección de valores atípicos. Este proceso es de suma importancia, ya que los valores atípicos pueden tener un gran impacto en las estadísticas descriptivas como la media y la desviación estándar. Por ejemplo, un solo valor atípico muy grande puede aumentar significativamente la media, dando una impresión errónea de que todos los valores en el conjunto de datos son altos Perner (2002).

Además, los valores atípicos pueden tener un efecto desproporcionado en los modelos de predicción. Por ejemplo, en la regresión lineal, un solo valor atípico puede cambiar significativamente la línea de mejor ajuste. Esto puede llevar a predicciones inexactas para otros datos Perner (2002). Su detección en los datos que se generen en este trabajo, por ejemplo, podría ayudar a identificar alguna medida fuera de lo normal en las distintas mediciones que se realizaron, ya que se podría asumir que una inconsistencia en el flujo

Tabla 6.1: Ejemplo de tabla con valores de tiempo

Valores del Tiempo
0.29291
0.303482
0.3097
0.403068
0.350694
0.381646
0.392883
0.914812
0.914812
0.929739
0.929739
0.872171
0.872171
0.757497
0.757497
0.958601
1.15349
1.15349
1.16709
1.16709
1.07264
1.07264
1.07533
1.07533
1.07533
1.18422
1.18422
1.95918
1.95918
1.95918
1.95918
1.96587
1.96587
1.96587
1.96587
1.94146
4.49189
4.49189
4.49189
4.49189
4.49189
4.49189

eléctrico de la unidad de procesamiento causó un aumento o disminución de la frecuencia del procesador o de la temperatura.

Sin embargo en ocasiones un valor atípico puede representar información valiosa para el modelo ya que pueden indicar la presencia de relaciones no lineales entre las variables. Estos valores atípicos pueden revelar patrones no detectados inicialmente en los datos y brindar información sobre la verdadera naturaleza de la relación entre las variables. Incluir los valores atípicos en el modelo puede ayudar a capturar estas relaciones no lineales y mejorar la capacidad predictiva del modelo Agarwal (2013).

Por ejemplo en la figura 6.6 se representa un ejemplo de un diagrama de cajas y bigote donde se puede visualizar los valores atípicos de una de las características descritas en la subsección 6.1.3, el porciento de uso de memoria. Pudiéndose visualizar que los valores atípicos se encuentran luego de la segunda línea a la izquierda de la caja azul, aproximadamente comprendidos en el intervalo de valores de 50 a 85. Para más información de la interpretación de este diagrama ver la subsección 5.5.1. La tarea utilizada para ejemplificar este diagrama fue el algoritmo de Strassen para la multiplicación de matrices usando una estrategia de paralelización con OpenMP. En la tabla 6.2 se muestra una representación del total de registros.

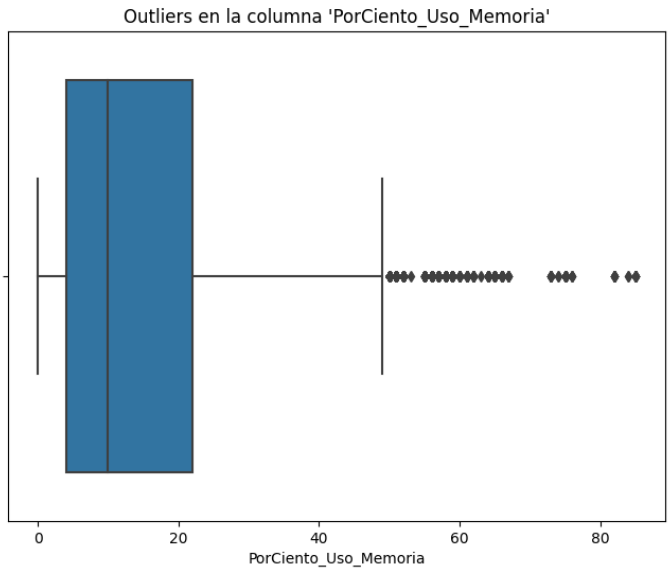

Figura 6.6: Diagrama de caja y bigotes para el por ciento de uso de memoria en la variante OpenMP recursivo

Análisis de correlación:

Luego sería importante realizar un análisis de correlación ya que permite identificar si existe una relación entre dos variables y cuán fuerte es esa relación Agarwal (2013). Por ejemplo, podría ser interesante saber si existe alguna correlación entre la temperatura y el tiempo de ejecución o entre el tipo de procesador y el tiempo.

Además, si dos variables independientes están altamente correlacionadas entre sí (multicolinealidad), esto puede ser un problema para algunos modelos de regresión. Como por

Tabla 6.2: Porcentaje de Uso de Memoria

Porcentaje de Uso de Memoria
22
51
50
50
51
50
50
51
50
51
51
49
50
22
22
22
22
22
22
22
22
22
22
22
57
57
57
57
57
57
57
57
57
58
57
58
57
59

ejemplo la regresión lineal ya que puede ser difícil para el modelo determinar el efecto de cada variable independiente en la variable dependiente. Esto puede resultar en coeficientes de regresión inestables y un incremento en el error estándar de los coeficientes Agarwal (2013).

Existen varios métodos de correlación que se utilizan para medir la relación entre dos variables (Pearson, Spearman y Kendall), los cuales fueron explicados en la sección 5.6.

Es importante tener en cuenta que el coeficiente de correlación de Pearson es sensible a la presencia de valores atípicos y puede verse afectado por ellos. Además, incluso si los supuestos de normalidad y linealidad no se cumplen exactamente, el coeficiente de Pearson aún puede proporcionar una medida útil de la relación lineal entre las variables, especialmente cuando las desviaciones son pequeñas Härdle and Simar (2012).

Por otro lado, los coeficientes de correlación de Spearman y Kendall no tienen supuestos de normalidad o linealidad. Estos coeficientes se basan en los rangos de las observaciones y evalúan la relación de clasificación entre las variables. Por lo tanto, son más robustos frente a las desviaciones de los supuestos y pueden capturar relaciones no lineales entre variables Härdle and Simar (2012).

Sin embargo, para este trabajo se decidió utilizar los tres coeficientes de correlación para medir las correlaciones tanto lineales como no lineales, entendiendo que pueden existir, quizás, correlaciones entre variables que no se expresen de manera lineal.

Por ejemplo en la figura 6.7 se puede visualizar la matriz de correlación de Kendall usada con la estrategia de paralelización de CUDA con OpenMP, con planteamiento recursivo. La tarea utilizada para ejemplificar este diagrama fue el algoritmo de Strassen para la multiplicación de matrices y se ejecutó en una Jetson TX2 con un tipo de procesador ARM Cortex A52, almacenamiento secundario embedded MultiMediaCard 5.1 con velocidades de lectura secuencial de hasta 250MB/s y velocidades de escritura secuencial de hasta 125MB/s, una tarjeta de memoria de 8gb, y el sistema operativo que se utilizó fue Ubuntu 22.04. En la tabla 6.3 se muestra una selección de los datos totales para generar la matriz de Kendall. Se puede observar en la matriz de correlación de la figura 6.7 una alta correlación entre el tamaño N de las matrices y el tiempo de ejecución de dicho algoritmo

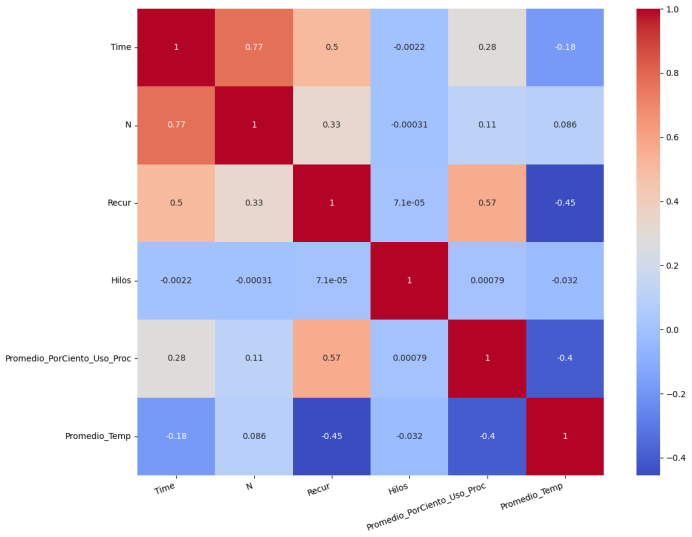

Figura 6.7: Matriz de correlación de Kendall para la variante de CUDA con OMP recursivo

Tabla 6.3: Datos de ejemplo para la generación de la matriz de Kendall

Time	N	Recur	Hilos	Promedio_Porciento_Uso_Proc	Promedio_Temp
0.29291	500	1	2	31.5	27
0.303482	500	1	3	28.83333333	27
0.3097	500	1	5	26.66666667	27
0.403068	500	2	1	31.5	27
0.350694	500	2	2	46	27
0.381646	500	2	4	40.16666667	27.5
0.392883	500	2	5	27.83333333	27
0.914812	1000	1	1	38	27.5
0.914812	1000	1	1	26.5	27
0.929739	1000	1	2	25.16666667	27.5
0.929739	1000	1	2	22	27
0.872171	1000	1	3	34	27.5
0.872171	1000	1	3	23.66666667	27.5
0.757497	1000	1	4	40	27.5
0.757497	1000	1	4	41.66666667	27.5
0.958601	1000	1	5	70.66666667	28
1.15349	1000	2	1	31.66666667	27.5

6.2.3. Preprocesamiento

Luego de que se tenga el conjunto de datos analizados, se procederá a realizar su preprocesamiento. El preprocesamiento de datos es un paso esencial que puede mejorar la calidad de los datos, aumentar la eficiencia computacional, facilitar el análisis y preparar los datos para el modelado. Sin un preprocesamiento de datos adecuado, los resultados del análisis de datos y del aprendizaje automático pueden ser inexactos o engañosos Agarwal (2013).

Limpieza de datos:
En esta etapa se empieza por la limpieza del conjunto de datos, en donde se identificarán valores faltantes. En este trabajo si se encuentran valores faltantes se procederá a eliminar el registro completo, ya que resulta incoveniente para esta investigación realizar cualquier tipo de método de imputación como la media o mediana, puesto que afectaría directamente al modelo de regresión y podría resultar en el aumento de errores en la predicción del tiempo.

De manera similar se procedería con los valores atípicos, sin embargo para este caso resulta importante saber primero qué causó ese valor atípico. Por ejemplo para decidir si se elimina se debe determinar si fue un error de medición a causa de algún mal funcionamiento eléctrico u otra problema, pero si no existió ningún error de este tipo entonces dicho valor podría significar información valiosa para el proceso de entrenamiento del modelo.

Por ejemplo en la tabla 6.4 se puede identificar en el tercer registro de la columna Time un valor faltante, en este trabajo el registro entero se eliminaría.

Codificación:

Tabla 6.4: Datos de ejemplo con valores faltantes

Time	N	Recur	Hilos	Max_PorCiento_Uso_Proc	Max_Temp
0.29291	500	1	2	31.5	27
0.303482	500	1	3	28.83333333	27
	500	1	5	26.66666667	27
0.403068	500	2	1	31.5	27
0.350694	500	2	2	46	27
0.381646	500	2	4	40.16666667	27.5
0.392883	500	2	5	27.83333333	27

Una vez que se haya realizado la limpieza de los datos se procede a realizar la codificación de las variables categóricas o nominales. Este proceso es importante ya que permite convertir datos categóricos o textuales en un formato numérico, ya que hay muchos algoritmos de aprendizaje automático que requieren un vector de entrada que sea de valores numéricos.

En general, los algoritmos de regresión, incluyendo los utilizados en este trabajo, la regresión lineal múltiple, la regresión con LSTM y la regresión con Redes Neuronales Secuenciales, requieren que las variables con datos categóricos se codifiquen de alguna manera antes de ser utilizadas en el modelo. Nelli (2018) Kuhn and Johnson (2019).

Las variables categóricas, como el tipo de procesador, no son numéricas y no pueden ser procesadas directamente por estos algoritmos. Por lo tanto, dichas variables deben ser convertidas a un formato numérico antes de ser utilizadas en un algoritmo de regresión . Por ejemplo en la tabla 6.5 se muestra un ejemplo de dos procesadores, y en la tabla 6.6 se muestra la codificación resultante usando el método *One-Hot-Encoding*.

Atributos a codificar:

- **Tipo de procesador**
- **Sistema Operativo**
- **Tipo de Memoria RAM**
- **Tipo de Disco Duro**

En este trabajo se decidió usar el método de codificación *One-Hot* ya que las características descritas anteriormente no tienen una relación de orden natural. Por ejemplo el atributo del tipo de procesador, o el sistema operativo no representa lo mismo que cuando se habla de un atributo como la altura (alto, bajo, medio) expresada de forma nominal.

Además, esta codificación preserva la información de todas las categorías, ya que cada categoría se convierte en una característica binaria independiente. Por lo tanto el modelo de regresión puede entrenarse de una forma más precisa.

No obstante, resulta conveniente mencionar que la codificación *one-hot*, como se explicó anteriormente, aumenta la dimensionalidad de los datos debido a la forma en que maneja las variables categóricas. En lugar de representar una variable categórica con una

Tabla 6.5: Tabla con Tipo de procesador

Tipo_Procesador
Intel(R) Core(TM) i7-3930K
Intel(R) Core(TM) i7-3930K
Intel(R) Core(TM) i7-3930K
Intel(R) Core(TM) i7-3930K
11th Gen Intel(R) Core(TM) i5-11400H
11th Gen Intel(R) Core(TM) i5-11400H
11th Gen Intel(R) Core(TM) i5-11400H
11th Gen Intel(R) Core(TM) i5-11400H
11th Gen Intel(R) Core(TM) i5-11400H

Tabla 6.6: One-Hot Encoding de tipo de procesador

Intel(R) Core(TM) i7-3930K	11th Gen Intel(R) Core(TM) i5-11400H
1	0
1	0
1	0
1	0
0	1
0	1
0	1
0	1
0	1

Tabla 6.7: Datos originales del tiempo y el consumo de memoria

Time	Max_Consumo_Memoria
3.351757	25332
1.736818	24996
1.143891	25440
0.855465	25452
0.683705	25460

sola columna, la codificación *one-hot* crea una nueva columna (o dimensión) para cada categoría única dentro de esa variable.

Por ejemplo, en este trabajo se tiene una variable categórica como el tipo de procesador, y debido a la amplia gama de tipos de procesadores que se manejan (core i5,core i7, core i9, celeron, xeon, amd ryzen 7...) la codificación *one-hot* los transformaría en nuevas variables de columna, lo que haría que tuviera un aumento gigantesco en la dimensionalidad. Este mismo caso del tipo de procesador, es extensible también para las demás características a codificar.

Sin embargo, todo depende de cuántas distintas unidades de procesamiento con distintos procesadores se estén utilizando para evaluar una tarea y recolectar datos. Ya que si los experimentos no son en tantas unidades, como es el caso de éste trabajo entonces este tipo de codificación resulta conveniente.

Normalización:
Luego de que se codifiquen los datos nominales, se pasaría al proceso de normalización. Al escogerse como algoritmo de regresión las redes neuronales y la regresión lineal, se deben normalizar los datos.Esto se debe a que los algoritmos de optimización utilizados para entrenar redes neuronales pueden ser sensibles a la escala de las características. La normalización ayuda a evitar problemas numéricos y a mantener una mejor estabilidad durante el proceso de optimización. Además ayuda a evitar que algunas características dominen el proceso de aprendizaje y garantiza que todas las características contribuyan de manera equilibrada a las predicciones de la red Aggarwal (2018).

Por otra parte cuando las características están en diferentes escalas, los coeficientes estimados por la regresión lineal pueden ser difíciles de interpretar. La normalización de los datos coloca las características en la misma escala, lo que facilita la interpretación de los coeficientes y permite comparar su influencia relativa Agarwal (2013).

Por ejemplo en la tabla 6.7 se muestra los datos del tiempo y el consumo de memoria sin normalizar y en la tabla 6.8 se muestran los datos normalizados usando la normalización *z-score*. Los datos son una representación de un experimento realizado usando el algoritmo de Strassen en una computadora con un procesador Intel(R) Core(TM) i7-8700 y una estrategia de paralelización con OpenMP.

En resumen, tanto en redes neuronales como en regresión lineal, la normalización de los datos ayuda a garantizar la estabilidad numérica, equilibrar las características y facilitar la interpretación de los coeficientes. Estos beneficios contribuyen a un mejor rendimiento

Tabla 6.8: Datos normalizados usando z-score

Time (z-score)	Max_Consumo_Memoria (z-score)
1,727 819	−0,498 831
0,646 699	−1,573 131
−0,262 267	0,344 992
−0,828 338	0,481 722
−1,283 914	0,644 248

y facilitan el análisis de los modelos de regresión Nelli (2018).

Aunque es cierto que los algoritmos basados en árboles, como *Random Forest*, no requieren que los datos estén normalizados para funcionar correctamente Liu et al. (2012), hay varias razones por las que aún podría ser útil normalizar los datos:

1. **Interpretabilidad**: Normalizar los datos puede hacer que los resultados sean más interpretables. Por ejemplo, si se está utilizando una medida de importancia de características como es el caso de este trabajo, entonces las características con rangos más grandes podrían parecer artificialmente más importantes. Normalizar los datos asegura que todas las características están en la misma escala, lo que puede hacer que las medidas de importancia de las características sean más comparables.

2. **Preprocesamiento consistente**: Si se está comparando varios algoritmos de aprendizaje automático, como es el caso de esta investigación, y algunos de esos algoritmos requieren normalización y otros no, puede ser más fácil y menos propenso a errores simplemente normalizar los datos para todos los algoritmos.

3. **Velocidad de convergencia**: Aunque los algoritmos basados en árboles no requieren normalización para funcionar correctamente, en algunos casos, la normalización puede mejorar la velocidad de convergencia del algoritmo.

4. **Reducción de la influencia de los valores atípicos**: La normalización puede ayudar a reducir la influencia de los valores atípicos en el modelo. Aunque los algoritmos basados en árboles son generalmente robustos a los valores atípicos, en algunos casos, los valores atípicos pueden afectar la calidad del modelo.

En resumen, aunque la normalización no es necesaria para los algoritmos basados en árboles como *Random Forest*, aún puede seguir siendo útil.

División del conjunto de datos:

Luego de normalizar se divide el conjunto de datos, en datos de entrenamiento y datos de prueba ya que al tener datos separados se puede evaluar de manera objetiva y realista el rendimiento del modelo en datos no vistos previamente. Esto proporciona una medida más confiable de la capacidad de generalización del modelo. Además, la división del conjunto de datos previene el sobreajuste, ya que si el modelo se hace muy bueno prediciendo los datos de entrenamiento no puede generalizar correctamente a nuevos datos. Además, permite comparar diferentes modelos y seleccionar el que tenga el mejor rendimiento en

Tabla 6.9: Tabla original

Time	N	Recur	Hilos	Max_PorCiento_Uso_Proc	Max_Temp
0.29291	500	1	2	31.5	27
0.303482	500	1	3	28.83333333	27
	500	1	5	26.66666667	27
0.403068	500	2	1	31.5	27
0.350694	500	2	2	46	27
0.381646	500	2	4	40.16666667	27.5
0.392883	500	2	5	27.83333333	27
0.914812	1000	1	1	38	27.5
0.914812	1000	1	1	26.5	27
0.929739	1000	1	2	25.16666667	27.5
0.929739	1000	1	2	22	27
0.872171	1000	1	3	34	27.5
0.872171	1000	1	3	23.66666667	27.5
0.757497	1000	1	4	40	27.5
0.757497	1000	1	4	41.66666667	27.5
0.958601	1000	1	5	70.66666667	28
1.15349	1000	2	1	31.66666667	27.5
1.15349	1000	2	1	20.5	27.5
1.16709	1000	2	2	32.16666667	27
1.16709	1000	2	2	22.66666667	27.5
1.07264	1000	2	3	33.33333333	27.5

el conjunto de prueba, ayudando así a elegir el modelo más adecuado para la predicción del tiempo, ya que en este trabajo se cuentan con cuatro modelos.

Por ejemplo en la tabla 6.9 se muestra un conjunto de datos que se obtuvo a partir de un experimento realizado usando el algoritmo de Strassen en una tarjeta Jetson TX2 y una estrategia de paralelización con OpenMP. Luego se aplicó una división de 80/20 en donde se utilizó el 80 % de los datos para entrenar como se muestra en la tabla 6.10 y el 20 % para probar la eficiencia del algoritmo de regresión y que pueda generalizar como se muestra en la tabla 6.11.

6.2.4. Entrenamiento

Una vez dividido los datos, en datos de entrenamiento y prueba, se pasa a la etapa de entrenamiento de cada uno de los modelos, de la forma explicada en las secciones 5.9, 5.10, 5.11 y 5.12 usando para este proceso los datos de entrenamiento obtenidos a través de la función de división explicada anteriormente. Por ejemplo en las tablas 6.12, 6.13, 6.14 y 6.15, se muestra una breve representación de los pesos y sesgos obtenidos en una Red neuronal secuencial de 3 capas, luego de entrenarla con los datos representados en la tabla 6.10, tomando como variable a predecir el tiempo. El proceso de entrenamiento se realizó usando la biblioteca tensorflow en python.

6.2.5. Evaluación del algoritmo de regresión

Para poder evaluar los regresores se utilizan cuatro métricas:

Tabla 6.10: Datos de entrenamiento

Time	N	Recur	Hilos	Max_PorCiento_Uso_Proc	Max_Temp
0.29291	500	1	2	31.5	27
0.303482	500	1	3	28.83333333	27
0.403068	500	2	1	31.5	27
0.350694	500	2	2	46	27
0.381646	500	2	4	40.16666667	27.5
0.392883	500	2	5	27.83333333	27
0.914812	1000	1	1	38	27.5
0.914812	1000	1	1	26.5	27
0.929739	1000	1	2	25.16666667	27.5
0.872171	1000	1	3	34	27.5
0.872171	1000	1	3	23.66666667	27.5
0.757497	1000	1	4	40	27.5
0.757497	1000	1	4	41.66666667	27.5
0.958601	1000	1	5	70.66666667	28
1.15349	1000	2	1	31.66666667	27.5
1.15349	1000	2	1	20.5	27.5

Tabla 6.11: Datos de prueba

Time	N	Recur	Hilos	Max_PorCiento_Uso_Proc	Max_Temp
1.16709	1000	2	2	32.16666667	27
1.16709	1000	2	2	22.66666667	27.5
1.07264	1000	2	3	33.33333333	27.5

Capa_1_Peso_1	Capa_1_Peso_2	Capa_1_Peso_3
-0.2554113	-0.14038442	-0.119239695

Tabla 6.12: Pesos de la Capa 1

Capa_2_Peso_1066	Capa_2_Peso_1067	Capa_2_Peso_1068	Capa_2_Peso_1069
0.15008338	-0.056453474	0.23620409	-0.20589916

Tabla 6.13: Pesos de la Capa 2

- **R cuadrado (R^2)**: Esta es una métrica que indica cuánta variación en la variable dependiente (o variable de respuesta) es explicada por la variable independiente (o variables) en un modelo de regresión. Un R^2 de 1 indica que el modelo explica toda la variación en la variable de respuesta, mientras que un R^2 de 0 indica que el modelo no explica ninguna de la variación. R^2 siempre está entre 0 y 1 Agarwal (2013).
- **Error cuadrático medio (MSE)**: Esta es una métrica que mide la diferencia promedio al cuadrado entre los valores predichos por un modelo y los valores reales. Es una medida de la precisión de un modelo de regresión. Un MSE más bajo indica un modelo más preciso Agarwal (2013).
- **Error absoluto medio (MAE)**: Esta es una métrica que mide la diferencia promedio absoluta entre los valores predichos por un modelo y los valores reales. Al igual que el MSE, es una medida de la precisión de un modelo de regresión. A diferencia del MSE, el MAE no penaliza tanto los errores grandes, ya que no eleva al cuadrado las diferencias Agarwal (2013).
- **Error porcentual absoluto medio (MAPE)**: Esta es una métrica que mide la diferencia promedio absoluta entre los valores predichos por un modelo y los valores reales, como un porcentaje de los valores reales. Es útil cuando se quiere expresar el error de un modelo en términos relativos en lugar de absolutos. Un MAPE más bajo indica un modelo más preciso Agarwal (2013).

Estas métricas se utilizan comúnmente para evaluar la precisión de los modelos de regresión, y cada una tiene sus propias ventajas y desventajas. Por ejemplo, el MSE y el MAE son fáciles de interpretar y calcular, pero pueden ser sensibles a los valores atípicos. El R^2 es una medida útil de la calidad de ajuste de un modelo, pero no dice nada sobre su precisión absoluta. El MAPE puede ser útil cuando se quiere expresar el error en términos porcentuales, pero puede ser indefinido o infinito si los valores reales son cero.

Aunque el coeficiente de R^2 no aporta información sobre la precisión absoluta del modelo puede resultar útil para entender qué tan bien el modelo de regresión puede explicar la variabilidad en los tiempos de ejecución del algoritmo.

R^2 mide la proporción de la variabilidad total en los tiempos de ejecución que es explicada por el modelo. Un R^2 cercano a 1 indica que el modelo puede explicar una gran proporción de la variabilidad en los tiempos de ejecución, mientras que un R^2 cercano a 0 indica que el modelo no puede explicar mucho de la variabilidad en los tiempos de ejecución.

Capa_3_Peso_28	**Capa_3_Peso_29**	**Capa_3_Peso_30**	**Capa_3_Peso_31**
0.12740378	-0.29977205	0.08586057	-0.40096363

Tabla 6.14: Pesos de la Capa 3

Capa_1_Sesgo_1	**Capa_1_Sesgo_2**	**Capa_1_Sesgo_3**	**Capa_1_Sesgo_4**
-0.024006277	0.012752869	0.00498414	-0.016842034

Tabla 6.15: Sesgos de la Capa 1

Por lo tanto, si se obtiene un alto valor de R^2, esto podría indicar que el modelo es capaz de capturar las relaciones subyacentes entre las variables predictoras y los tiempos de ejecución del algoritmo.

Capítulo 7

Resultados

Con el fin de poder validar la propuesta se decidió realizar una serie de experimentos en donde se tomó una tarea y se midió su comportamiento bajo distintas configuraciones en varias unidades de procesamiento. Luego se procedió a analizar y preprocesar los datos con el fin de introducirlos en los algoritmos de regresión presentados en la propuesta.

En este caso se decidió escoger como tarea el algoritmo de Strassen. El algoritmo de Strassen es altamente paralelizable porque las operaciones de multiplicación y adición en las submatrices pueden realizarse de manera independiente unas de otras como se muestra en la figura 7.1. Cada una de las multiplicaciones de las submatrices se puede asignar a un procesador o un hilo separado, lo que significa que muchas operaciones pueden realizarse simultáneamente, aprovechando al máximo las capacidades de los sistemas de computación en paralelo. Además el algoritmo de Strassen realiza menos multiplicaciones que el algoritmo estándar de multiplicación de matrices, lo que también contribuye a su eficiencia y a su capacidad para ser paralelizado. Sin embargo, aunque el algoritmo de Strassen es teóricamente más rápido para matrices muy grandes, en la práctica, debido a la sobrecarga de dividir las matrices y combinar los resultados, a menudo es más lento que el algoritmo estándar para matrices de tamaño pequeño a mediano. Además, requiere más memoria que el algoritmo estándar, lo que puede ser un problema para matrices muy grandes Cormen (2013).

7.1 Estrategia de paralelización del algoritmo de Strassen

El algoritmo de Strassen para la multiplicación de matrices se puede paralelizar de varias formas. Una de las formas más comunes es paralelizar la multiplicación en el último nivel de recursividad. Esto se hace generalmente cuando las submatrices alcanzan un cierto tamaño umbral en el que la multiplicación de matrices se puede realizar de manera eficiente en paralelo Cormen (2013). En lugar de realizar la multiplicación estándar de matrices, que requiere $O(n^3)$ operaciones, el algoritmo de Strassen realiza la multiplicación en $O(n^{log_2 7})$ operaciones Cormen (2013). El algoritmo funciona dividiendo cada una de las matrices en cuatro submatrices, y luego calculando siete productos de estas submatrices. Las submatrices y los productos se definen de la siguiente manera para dos matrices A y B de tamaño $2n \times 2n$ Cormen (2013):

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \quad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

donde cada A_{ij} y B_{ij} es una matriz de tamaño $n \times n$.

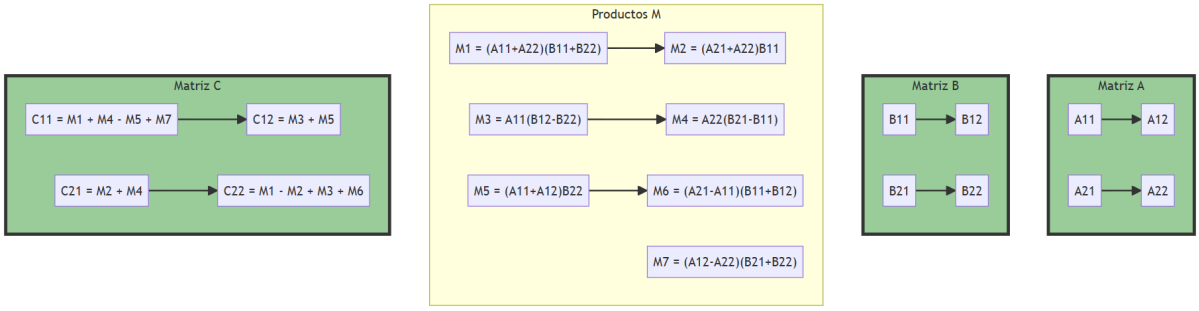

Figura 7.1: Proceso de obtención de la matriz resultante C en el algoritmo de Strassen

Los siete productos, P_i, se calculan de la siguiente manera:

$$
\begin{aligned}
P_1 &= A_{11}(B_{12} - B_{22}) \\
P_2 &= (A_{11} + A_{12})B_{22} \\
P_3 &= (A_{21} + A_{22})B_{11} \\
P_4 &= A_{22}(B_{21} - B_{11}) \\
P_5 &= (A_{11} + A_{22})(B_{11} + B_{22}) \\
P_6 &= (A_{12} - A_{22})(B_{21} + B_{22}) \\
P_7 &= (A_{11} - A_{21})(B_{11} + B_{12})
\end{aligned}
$$

Finalmente, los elementos de la matriz resultante C se calculan a partir de los productos P_i:

$$
C = \begin{bmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_5 + P_1 - P_3 - P_7 \end{bmatrix}
$$

En el último nivel de recursividad, cuando las submatrices son lo suficientemente pequeñas, la multiplicación de matrices se realiza de manera trivial (es decir, se realiza la multiplicación estándar de matrices).

Esta multiplicación se puede paralelizar. Cada elemento de la matriz resultante se puede calcular de forma independiente, por lo que estos cálculos se pueden realizar en paralelo.

Por ejemplo, si en el último nivel de recursividad tenemos dos matrices de 4×4:

$$
A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}, \quad B = \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix}
$$

La matriz resultante C se calcula como:

$$C = \begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & c_{34} \\ c_{41} & c_{42} & c_{43} & c_{44} \end{bmatrix}, \quad \text{donde} \quad c_{ij} = \sum_{k=1}^{4} a_{ik} b_{kj}$$

Cada uno de los elementos c_{ij} se puede calcular en paralelo utilizando diferentes hilos de procesamiento.

Además, las operaciones de suma y resta de matrices en el algoritmo se pueden realizar también se pueden paralelizar a nivel de cada elemento de la matriz. Cada operación de suma o resta puede ser asignada a un hilo de procesamiento diferente.

Por ejemplo, si tenemos dos matrices 4×4 :

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}, \quad B = \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix}$$

Podemos calcular la suma de estas dos matrices en paralelo donde cada elemento de la matriz resultante $C = A + B$ se calcula como $c_{ij} = a_{ij} + b_{ij}$. Cada una de estas operaciones de suma puede ser realizada por un hilo de procesamiento diferente. Por ejemplo, un hilo puede calcular $c_{11} = a_{11} + b_{11}$, otro hilo puede calcular $c_{12} = a_{12} + b_{12}$, y así sucesivamente. De esta manera, los 16 elementos de la matriz C pueden ser calculados en paralelo. De manera similar, si necesitamos calcular la diferencia de las matrices A y B, cada elemento de la matriz resultante $D = A - B$ se calcula como $d_{ij} = a_{ij} - b_{ij}$. Cada una de estas operaciones de resta puede ser realizada por un hilo de procesamiento diferente. Para lograr esta paralelización tanto para la multiplicación como para las operaciones de suma y resta, se utilizó las tecnologías de CUDA y OpenMP (ver más en sobre paralelismo en la sección 2.10).

7.1.1. Estrategia de multiplicación y suma usando CUDA

CUDA es una plataforma de computación paralela y una API de modelo de programación creada por NVIDIA. Esta permite un uso significativo de la potencia de procesamiento de las GPU para la computación general Kirk (2007).

En CUDA, un hilo es la unidad básica de ejecución y está contenido dentro de un bloque. Un bloque es un grupo de hilos que pueden cooperar entre sí mediante el uso de mecanismos de sincronización y memoria compartida. Los bloques, a su vez, están organizados en una cuadrícula que define el espacio de índices de bloques. Cada hilo y bloque tiene un identificador único, que se puede usar para decidir qué parte del cálculo debe realizar Kirk (2007).

La GPU NVIDIA ejecuta bloques de hilos en grupos llamados warps. Un warp consta de 32 hilos que se ejecutan en paralelo. La GPU programa los warps para su ejecución, y si todos los hilos de un warp siguen la misma ruta de ejecución, la GPU puede ejecutar la instrucción una vez para todo el warp, lo que mejora la eficiencia Kirk (2007).

La paralelización de la multiplicación en el código CUDA se vería así:

```
__global__ void matrixMul(float* C, float* A, float* B, int N) {
    int row = blockIdx.y * blockDim.y + threadIdx.y;
    int col = blockIdx.x * blockDim.x + threadIdx.x;

    float val = 0;
    if (row < N && col < N) {
        for (int i = 0; i < N; ++i) {
            val += A[row * N + i] * B[i * N + col];
        }
        C[row * N + col] = val;
    }
}
```

En este código, *blockIdx*, *blockDim* y *threadIdx* son variables predefinidas en CUDA que identifican el bloque y el hilo actual, y N es el tamaño de las matrices. Cada hilo calcula un elemento de la matriz resultante, sumando los productos de los elementos correspondientes de las filas de la matriz A y las columnas de la matriz B.

La paralelización de la suma (extensible para la resta al cambiar el signo) en el código CUDA se vería así:

```
__global__ void matrixAdd(float* C, float* A, float* B, int N) {
    int row = blockIdx.y * blockDim.y + threadIdx.y;
    int col = blockIdx.x * blockDim.x + threadIdx.x;

    if (row < N && col < N) {
        C[row * N + col] = A[row * N + col] + B[row * N + col];
    }
}
```

En este código, *blockIdx, blockDim* y *threadIdx* son variables predefinidas en CUDA que identifican el bloque y el hilo actual, y 'N' es el tamaño de las matrices. Cada hilo calcula un elemento de la matriz resultante, sumando los elementos correspondientes de las matrices A y B.

7.1.2. Estrategia de multiplicación usando OpenMP

OpenMP es una API de programación que soporta la programación multiproceso de memoria compartida en C, C++, y Fortran en múltiples plataformas. OpenMP utiliza un modelo de ejecución de hilos de fork-join. En este modelo, un hilo maestro crea un punto de fork al comienzo de una región paralela, creando hilos adicionales que ejecutan el código dentro de la región paralela Mattson (2001).

En el caso de la multiplicación de matrices, cada hilo en OpenMP se encarga de calcular un elemento c_{ij} de la matriz resultante. Esto se puede lograr dividiendo la matriz en bloques y asignando cada bloque a un hilo diferente. De esta manera, cada hilo realiza una parte del cálculo total, lo que permite que la multiplicación de matrices se complete más rápidamente que si se realizara de manera secuencial.

En OpenMP, un hilo es la unidad básica de ejecución y los hilos pueden ser creados y destruidos dinámicamente durante la ejecución del programa. OpenMP proporciona directivas de compilador, funciones de biblioteca y variables de entorno que influyen en el comportamiento del tiempo de ejecución Mattson (2001).

En el código OpenMP, esto se vería así:

```
#pragma omp parallel for
for (int i = 0; i < N; ++i) {
    for (int j = 0; j < N; ++j) {
        float sum = 0;
        for (int k = 0; k < N; ++k) {
            sum += A[i * N + k] * B[k * N + j];
        }
        C[i * N + j] = sum;
    }
}
```

En este código, *#pragma omp parallel for* es una directiva de OpenMP que indica que el siguiente bucle 'for' debe ser ejecutado en paralelo por múltiples hilos. 'N' es el tamaño de las matrices. Cada hilo calcula un elemento de la matriz resultante, sumando los productos de los elementos correspondientes de las filas de la matriz A y las columnas de la matriz B Mattson (2001).

En el caso de la suma de matrices, cada hilo puede calcular la suma de un conjunto de elementos de la matriz de manera independiente. Esto se puede hacer dividiendo la matriz en bloques y asignando cada bloque a un hilo diferente.

```
#include <omp.h>

void matrixAdd(float** C, float** A, float** B, int N) {
    #pragma omp parallel for
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < N; ++j) {
            C[i][j] = A[i][j] + B[i][j];
        }
    }
}
```

En este código, la directiva *#pragma omp parallel for*le dice a OpenMP que cree un equipo de hilos y divida la iteración del bucle entre ellos. Cada hilo calcula la suma de un conjunto de elementos de la matriz de manera independiente.

7.2 Primera fase

En la primera fase se decidió probar el algoritmo de Strassen en distintas plataformas o unidades de procesamiento, variándole distintas configuraciones y midiendo los parámetros durante su ejecución de la forma mencionada en el capítulo de Propuesta
.

Se decidió configurar las ejecuciones usando las dos estrategias de paralelización con CUDA y OPENMP mencionadas anteriormente, una variante secuencial en donde se hace la multiplicación de matrices de manera estándar y una combinación de la estrategia de paralelización CUDA con OpenMP en donde los hilos OpenMP se encargan de realizar la paralelización de la suma de matrices y los hilos CUDA se encargan de realizar las distintas multiplicaciones. Para todos los casos se usó una optimización *-03* del compilador GNU. Luego para las estrategias de paralelización de la multiplicación con OPENMP se aplicó otra variante en donde se decidió quitarle el planteamiento recursivo al algoritmo de Strassen y ejecutar cada uno de los productos de submatrices P_n que se propone en Strassen de manera estándar.

Por tanto quedaron entonces las siguientes variantes:

- **Variante CUDA**: Paraleliza las multiplicaciones del algoritmo de Strassen en el último nivel de recursividad solamente usando CUDA.
- **Variante OpenMP**: Paraleliza las multiplicaciones del algoritmo de Strassen en el último nivel de recursividad solamente usando OpenMP.
- **Variante CUDA con OpenMP**: Paraleliza las multiplicaciones del algoritmo de Strassen en el último nivel de recursividad solamente usando CUDA y paraleliza las operaciones de suma y resta que se hagan en el algoritmo usando OpenMP.
- **Variante OpenMP no recursivo**: Paraleliza las operaciones de multiplicación de las submatrices P_n. Por lo que se elimina el planteamiento recursivo.
- **Variante secuencial**: No se realiza ninguna paralelización

Las variaciones en la configuración de cada variante se describen a continuación:

- **Variante CUDA**: Esta variante solamente se pudo probar en un solo equipo debido a las limitaciones para encontrar dispositivos que poseyeran tarjetas de video Nvidia.
  - Características pre-ejecución tarea:
    - **Tamaño N**: El tamaño N de las matrices a multiplicar se definió como punto de partida en 1000 y como punto final en 7500 y se fue variando con un paso de 500 para cada instancia.
    - **Hilos OpenMP**: En el caso de la variante con CUDA solamente, no se considera el uso de hilos OpenMP por lo que se introduce el valor 0
    - **Recursividad**: La recursividad se ajustó de 1 a la permitida para cada tamaño de las matrices a multiplicar. El tamaño máximo de recursividad al que se llega se calcula a priori para cada par de matrices a multiplicar. Por lo tanto, el número máximo de niveles de recursión que se pueden alcanzar para una matriz de tamaño $n \times n$ es $\log_2 n$, redondeado hacia abajo al entero más cercano. Por ejemplo, si se tiene una matriz de tamaño 1024×1024, se puede dividir en 2 hasta 10 veces antes de llegar a una matriz de tamaño 1×1. Entonces, el número máximo de niveles de recursión para una matriz de tamaño 1024×1024 es 10.

- **Hilos CUDA**: Los hilos CUDA se establecieron al máximo. Se utilizó una cuadrícula 2D de bloques, y cada bloque contiene una cuadrícula 2D de hilos. El número de bloques en cada dimensión es el tamaño de la matriz N dividido por el número de hilos por bloque NT, redondeado hacia arriba. Esto asegura que haya suficientes bloques para cubrir toda la matriz, incluso si N no es un múltiplo de NT. Esta configuración se mantiene constante para todas las pruebas.

  ```
  int NT = 32;
  dim3 bloques((int)ceil((double)n/(double)NT)
  , (int)ceil((double)n/(double)NT));
  dim3 hilos(NT,NT);
  ```

- Características pre-ejecución unidades de procesamiento: Al solamente utilizarse un equipo para entrenar los distintos modelos de regresión usando esta variante no se tomaron en cuenta las características pre-ejecución de la unidad de procesamiento puesto que una variable predictora (o característica) que tiene el mismo valor en todas las filas no aporta ninguna información útil para un modelo de regresión. La varianza en los datos es esencial para que un modelo pueda aprender de ellos. Si una característica no tiene varianza, es decir, si todos los valores son iguales, entonces esa característica no puede tener ningún efecto en la variable de salida y, por lo tanto, no tiene sentido incluirla en el modelo Hastie et al. (2001).

  Los algoritmos de regresión se basan en la relación entre las características (variables independientes) y la variable objetivo (variable dependiente). Si una característica no tiene varianza (es decir, todos los valores son iguales), entonces no hay ninguna relación que pueda ser explotada por el algoritmo de regresión Hastie et al. (2001).

  El equipo donde se hizo la prueba fue una Jetson TX2 con un tipo de procesador ARM Cortex A52, almacenamiento secundario embedded MultiMediaCard 5.1 con velocidades de lectura secuencial de hasta 250MB/s y velocidades de escritura secuencial de hasta 125MB/s, una tarjeta de memoria de 8gb, y el sistema operativo que se utilizó fue Ubuntu 22.04.

- Características de monitoreo : Del conjunto total de características a monitorear que se habían propuesto, para esta variante se consideró el tiempo de ejecución, el porciento de uso del procesador, la frecuencia del procesador, la memoria usada y la temperatura.

  Las características de monitoreo del almacenamiento secundario, no se tomaron en cuenta ya que esta tarea no hace uso del disco para realizar ninguna operación de lectura o escritura, aunque las mediciones se guardaron en un archivo csv, el algoritmo de Strassen directamente no hace ninguna operación de uso de disco, puesto que los tamaños de las matrices cabían perfectamente en la memoria RAM y no se necesitó de espacio adicional. Por tanto las mediciones de las características del disco no influyen en el tiempo de ejecución del algoritmo.

Por otro lado los valores que se obtuvieron de las mediciones de la frecuencia del procesador fueron constantes en esta plataforma, y pasa lo mismo que con las características pre-ejecución, al no haber varianza no aporta ninguna información útil al algoritmo de regresión. Por tanto esta característica no se consideró.

- **Variante CUDA OPENMP**:
  - Características pre-ejecución tarea:
    - **Tamaño N**: Se definió exactamente igual que la variante de CUDA .
    - **Hilos OpenMP**: Los hilos OpenMP se configuraron de 1 a 12, para cada tamaño del problema y se fue variando con un paso de 1 en cada instancia.
    - **Recursividad**: Se definió exactamente igual que la variante de CUDA
    - **Hilos CUDA**: Se establecieron exactamente igual que en CUDA, al límite de lo permitido.
  - Características pre-ejecución unidades de procesamiento: En esta variante por la misma razón que en la variante de CUDA se probó en el mismo dispositivo(Jetson TX2), generándose el mismo problema de la varianza para las características. Por tanto tampoco se consideraron como parámetros para el entrenamiento del algoritmo de regresión.
  - Características de monitoreo : Situación similar al caso de la variante de CUDA. Solamente se consideraron el tiempo de ejecución, el porciento de uso del procesador, la frecuencia del procesador, la memoria usada y la temperatura y para el caso de la frecuencia del procesador se obtuvieron valores iguales en cada instancia de configuración. Por tanto esta característica se desechó para el entrenamiento del algoritmo de regresión

- Variante OpenMP:
  - Características pre-ejecución tareas:
    - **Tamaño N**: El tamaño N de las matrices a multiplicar se definió como punto de partida en 1000 y como punto final en 7500 y se fue variando con un paso de 500 para cada instancia.
    - **Recursividad**: Se estableció igual que en las variantes de CUDA.
    - **Hilos OpenMP**: Se estableció igual que la variante de CUDA con OpenMP.
    - **Hilos CUDA**: Con esta variante de paralelización no se considera el uso de hilos CUDA.
  - Características pre-ejecución de la unidad de procesamiento: Del conjunto total de características se consideraron todas exceptuando las características de el dispositivo de almacenamiento, ya que como se explicó anteriormente en esta variante de Strassen con esta estrategia de paralelización no se hace uso de disco. Las pruebas para esta variante se realizaron en 28 unidades de procesamiento independientes con características similares en algunas. Las características de cada uno se pueden evidenciar en la tabla C.1 y la tabla C.2 expuesta en los apéndices.

- Características de monitoreo: Las características que se monitorearon en las 28 unidades de procesamiento fueron el tiempo de ejecución, el porciento de uso del procesador, la frecuencia del procesador, la memoria usada y la temperatura. Las características de monitoreo del disco no se toman en cuenta tampoco por la misma razón del tipo de algoritmo que se está analizando.

- **Variante OpenMP no recursivo**: Este caso es parecido a OpenMP recursivo variando en que no se toma en cuenta dentro de las características pre-ejecución de la tarea o la configuración, el nivel de recursividad, y además esta variante solamente se probó en 21 unidades de procesamiento independientes con características similares entre sí.

- **Variante secuencial**: Este caso también es parecido a OpenMP recursivo, varía en que no se toma en cuenta los hilos OpenMP y solamente se probó en 4 unidades de procesamiento independientes con características similares entre sí.

7.3 Segunda fase

Una vez obtenidos los datos de los experimentos se procede a realizar la función de unión en donde se concatenaron las mediciones obtenidas del monitoreo para cada configuración del algoritmo de Strassen. Para luego agrupar el conjunto de datos a partir de las características críticas como se muestra en 6.4.

Quedaron entonces cinco conjunto de datos distintos:

- **Conjunto de datos para OpenMP recursivo**
- **Conjunto de datos para OpenMP no recursivo**
- **Conjunto de datos para CUDA recursivo**
- **Conjunto de datos para CUDA con OpenMP**
- **Conjunto de datos para secuencial**

Con cada conjunto de datos obtenido entonces se procedió a realizar el análisis descriptivo para cada uno. En las gráficas 7.2, 6.5, 7.3, 7.4, 7.5, 7.6 se pueden observar las distintas de distribuciones del tiempo para cada variante y una curva de ajuste. El eje y representa la frecuencia o densidad de los valores de la variable, y el eje x representa los valores que toma la variable del tiempo. La distribución que siguen estas gráficas es desconocida sin embargo se instuía que la distribución que se siguía para las gráficas de OpenMP recursivo, OpenMP no recursivo y secuencial podían seguir algún tipo de distribución exponencial, por tanto se le realizó la prueba de Kolmogorov-Smirnov para determinar con un nivel de significancia del 5 % si seguía o no esa distribución, obteniéndose entonces un p-valor de 0 por lo que se rechazó la hipótesis nula.

La prueba de Kolmogorov-Smirnov es una prueba de bondad de ajuste que compara la función de distribución acumulativa (CDF) de los datos con la CDF de una distribución teórica. Un p-valor de 0 significa que se rechaza la hipótesis nula de que los datos siguen la distribución teórica con un alto grado de confianza Lopes (2011).

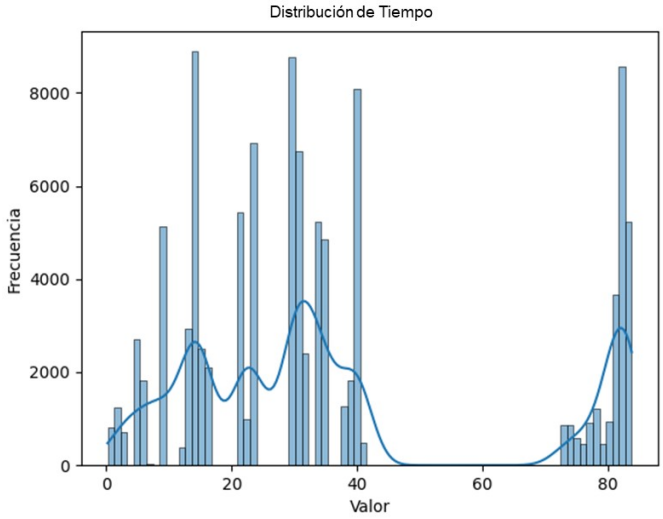

Figura 7.2: Distribución del tiempo en la variante CUDA con OpenMP recursivo

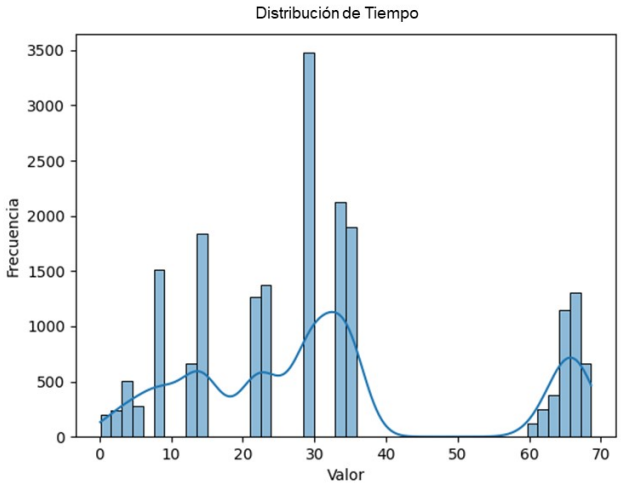

Figura 7.3: Distribución del tiempo en la variante CUDA recursivo

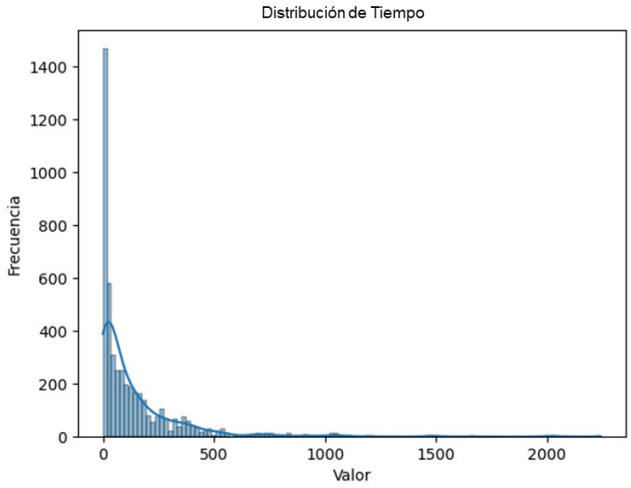

Figura 7.4: Distribución del tiempo en la variante OpenMP no recursivo

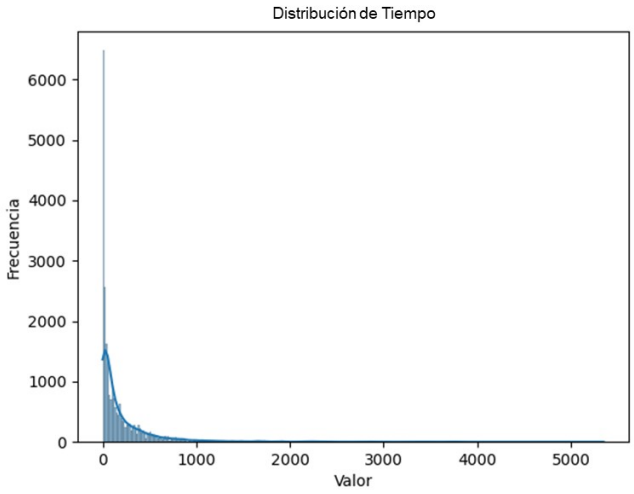

Figura 7.5: Distribución del tiempo en la variante OpenMP recursivo

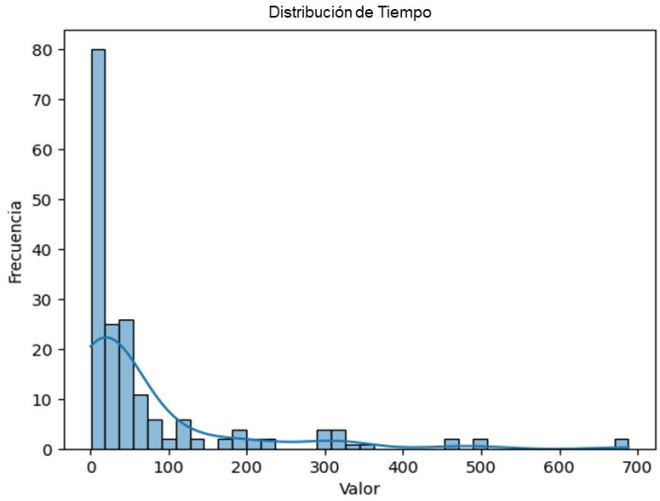

Figura 7.6: Distribución del tiempo en la variante secuencial

Por otra parte se realizó el análisis de correlación usando los tres métodos descritos en el capítulo anterior, Spearman, Pearson y Kendall. Obteniéndose los coeficientes de correlación entre dos variables expresados en cada celda de las matrices para cada variante. En las figuras 7.7, 7.8, 7.9, 7.10, 7.11, 7.12, 7.13, 7.14, 7.15, 7.16, 7.17, 7.18, 7.19, 7.20, y 7.21, se pueden apreciar las matrices de correlación con los tres métodos para cada variante. En las figuras 7.7, 7.9, 7.10, 7.13, 7.15, 7.16, 7.17, 7.19, y 7.20, los hilos expuestos como atributo se refiere a hilos OpenMP. Se puede observar de manera general que existe una correlación fuerte positiva entre el tamaño N de las matrices y el tiempo de ejecución lo que puede resultar lógico, debido a que el aumento de la cantidad de elementos a procesar puede influir linealmente en el aumento del tiempo. Además se encontró que el nivel de recursividad para las variantes donde se utiliza CUDA influye en el aumento del tiempo de ejecución. Este resultado se presupone que sea provocado por la cantidad de transferencias desde el *host* al *device* que aumentan proporcionalmente al aumentar la recursividad. Dichas transferencias se realizan para poder pasar los datos de las matrices a las tarjetas de video, y una vez que esta se encargue de procesarlas, se devuelve el resultado hacia atrás. Esto provoca un costo al tiempo, aún cuando se están paralelizando las multiplicaciones con los hilos de la tarjeta de video.

Por otra parte se puede verificar por ejemplo en la figura 6.7 que los hilos OpenMP tienen una baja correlación con el tiempo. Esto se presupone que sea debido a que la carga de procesamiento alta se condensa en los hilos CUDA, que son los que se encargan de las multiplicaciones de las matrices en el último nivel de recursividad, los hilos OpenMP solamente se utilizan para paralelizar las sumas y restas lo cual en proporción, no representan una alta carga computacional.

Sin embargo la correlación no implica causalidad. Las medidas de correlación, como las correlaciones de Pearson, Spearman y Kendall, miden la fuerza y la dirección de la relación lineal o monótona entre dos variables, pero no pueden determinar si un cambio en una variable causa un cambio en la otra Härdle and Simar (2012).

Incluso si existe una correlación alta entre dos variables, esto no significa que se pueda usar una variable para predecir con precisión la otra. La correlación solo mide la relación lineal o monótona entre las variables, pero la relación real podría ser más com-

pleja y podría estar influenciada por otros factores Härdle and Simar (2012).

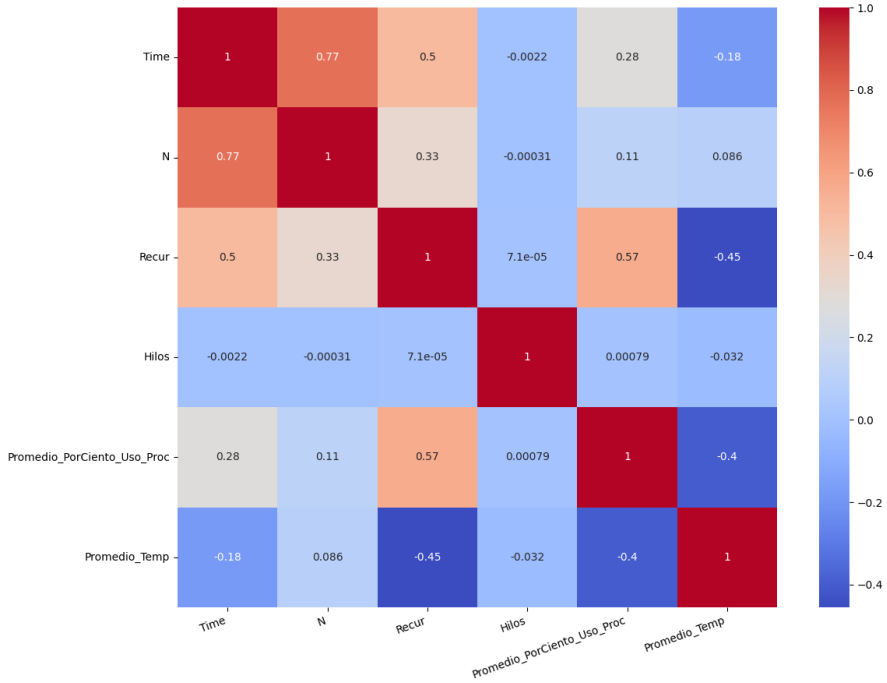

Figura 7.7: Matriz de correlación de Kendall para la variante de CUDA con OMP recursivo

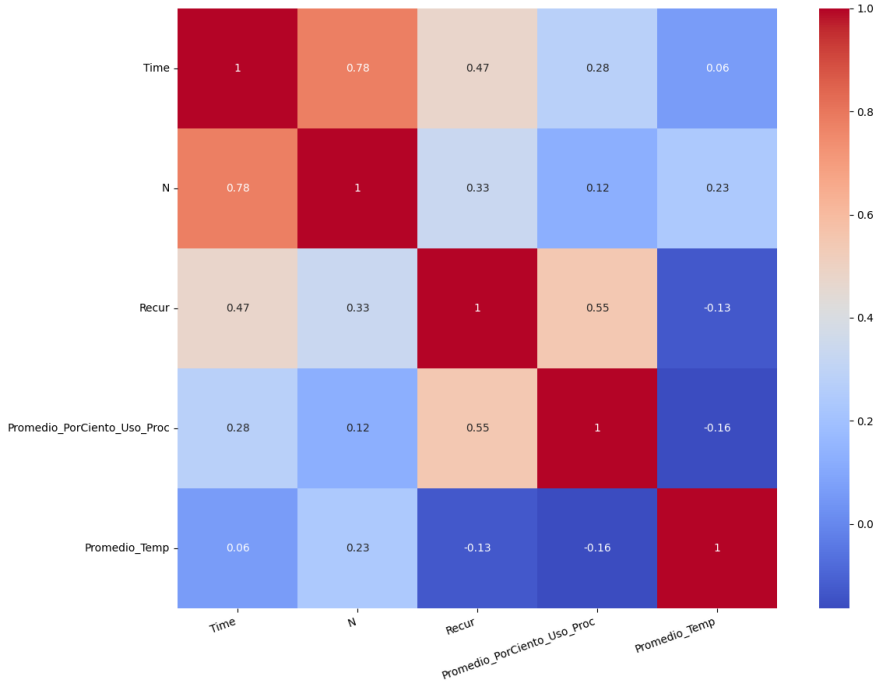

Figura 7.8: Matriz de correlación de Kendall en la variante CUDA recursivo

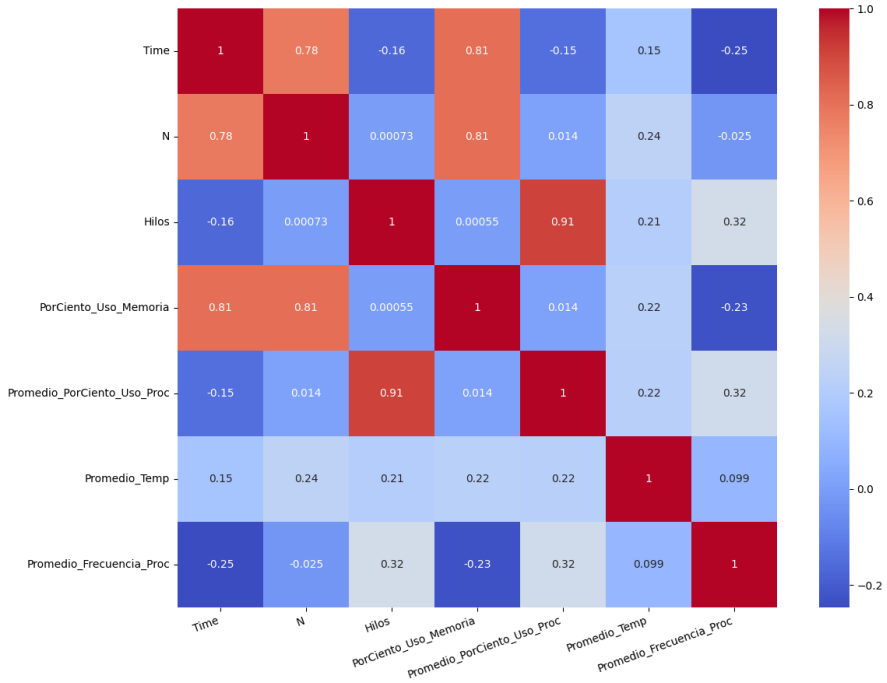

Figura 7.9: Matriz de correlación de Kendall en la variante OMP no recursivo

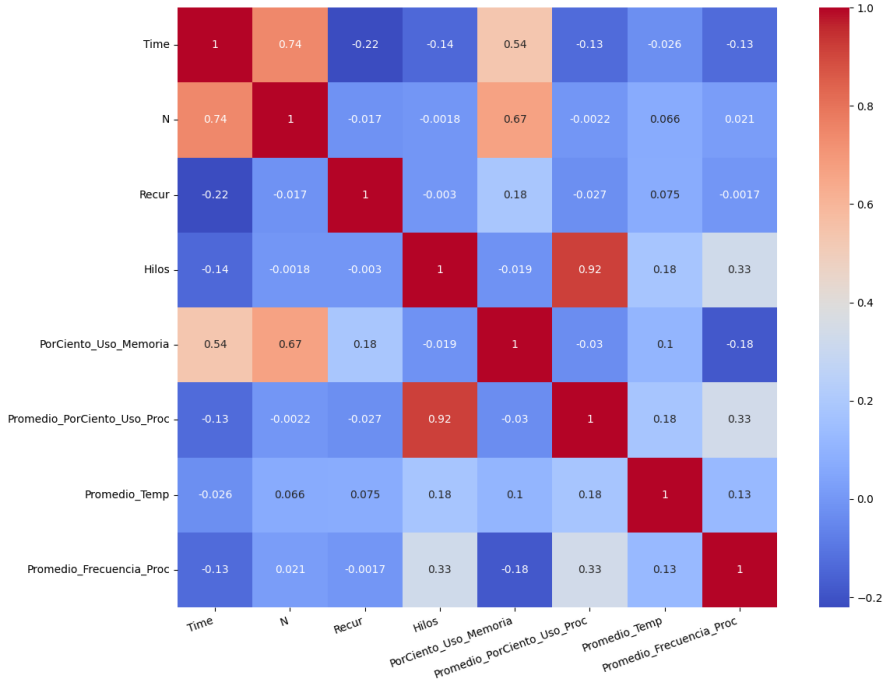

Figura 7.10: Matriz de correlación de Kendall en la variante OMP recursivo

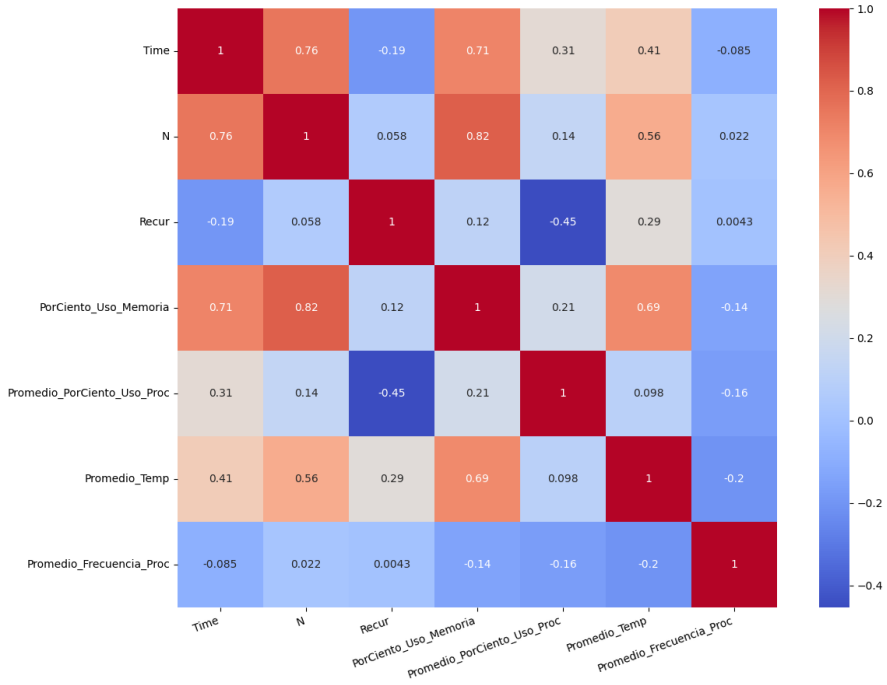

Figura 7.11: Matriz de correlación de Kendall en la variante secuencial

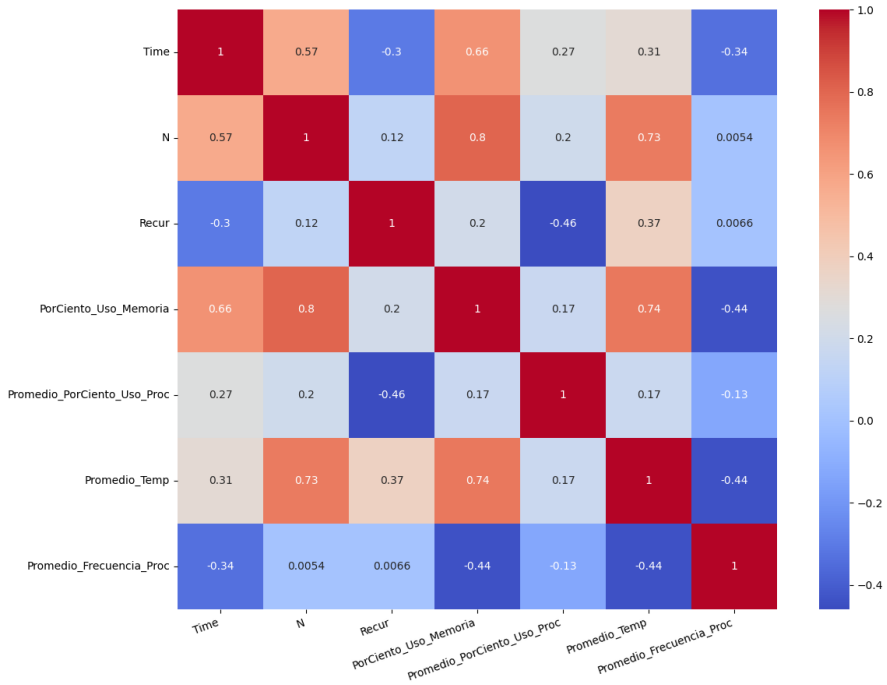

Figura 7.12: Matriz de correlación de Pearson en la variante secuencial

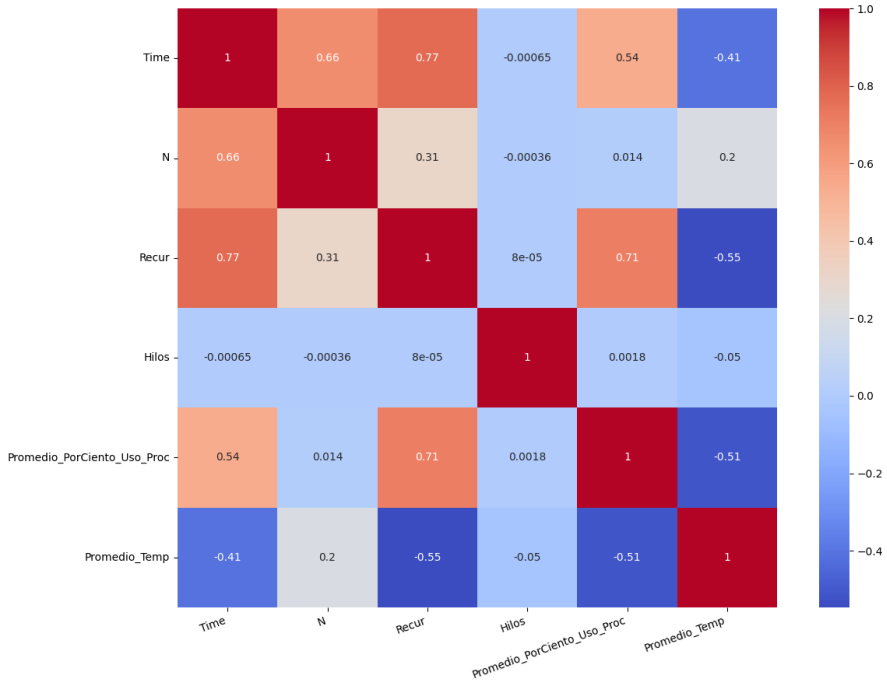

Figura 7.13: Matriz de correlación de Pearson en la variante CUDA con OMP recursivo

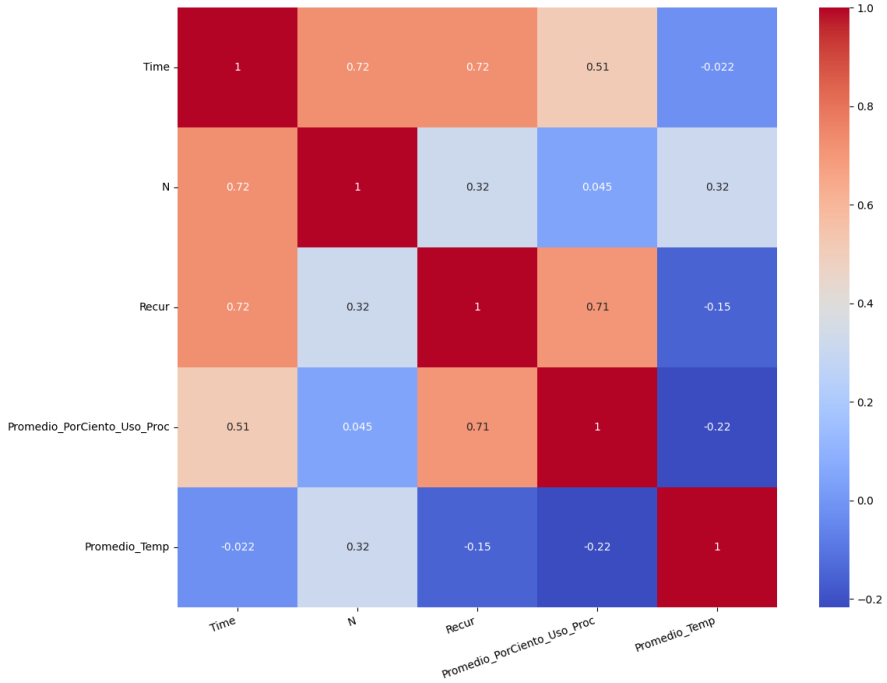

Figura 7.14: Matriz de correlación de Pearson en la variante CUDA recursivo

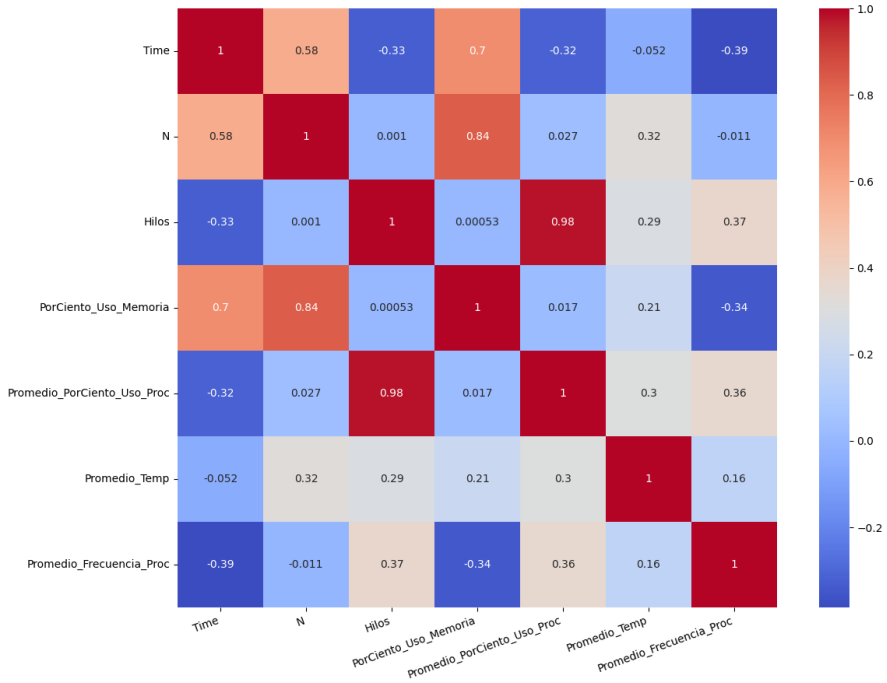

Figura 7.15: Matriz de correlación de Pearson en la variante OMP no recursivo

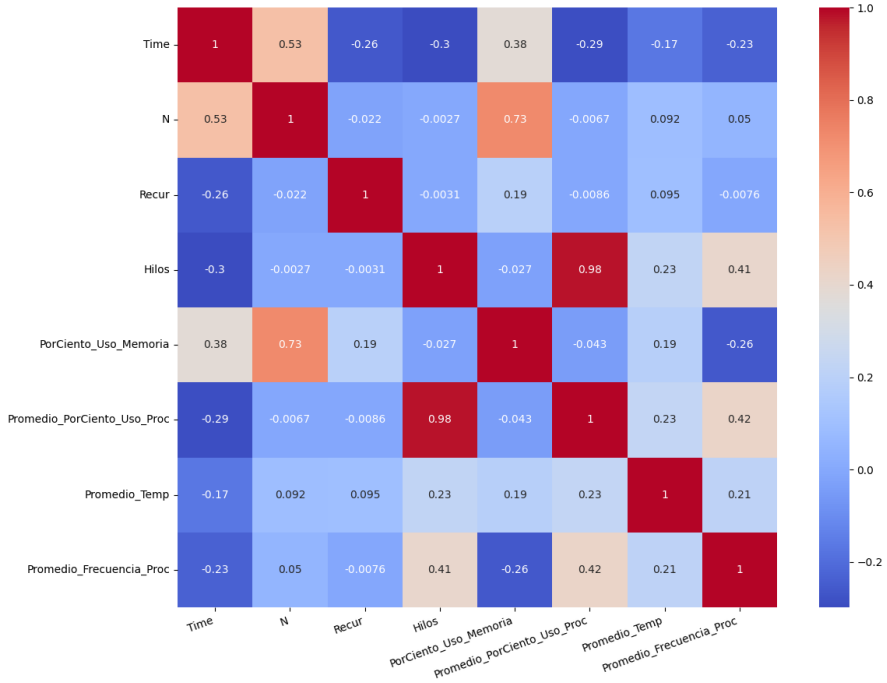

Figura 7.16: Distribución del tiempo en la variante OpenMP recursivo

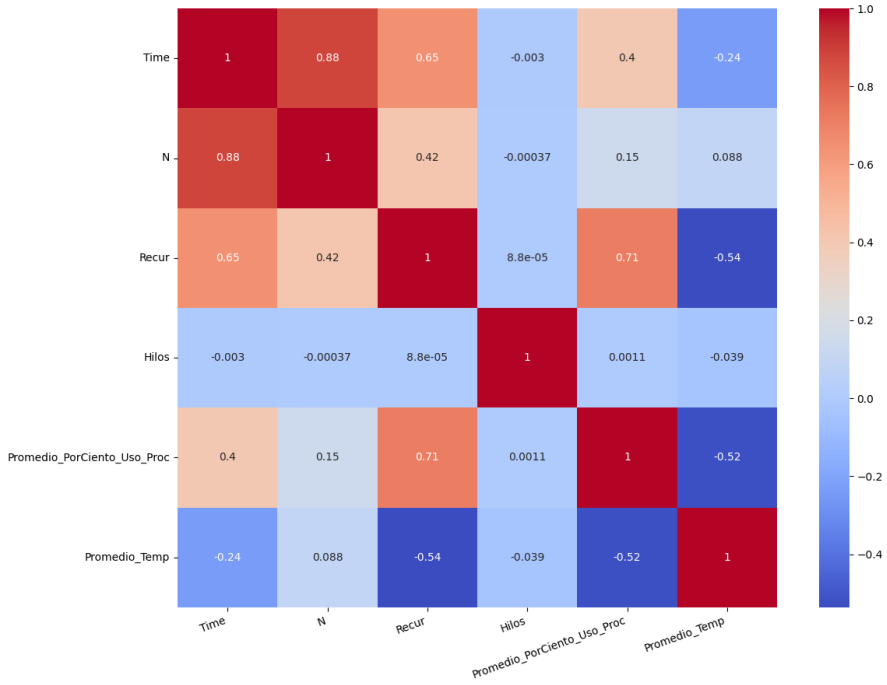

Figura 7.17: Matriz de correlación de Spearman en la variante CUDA con OMP recursivo

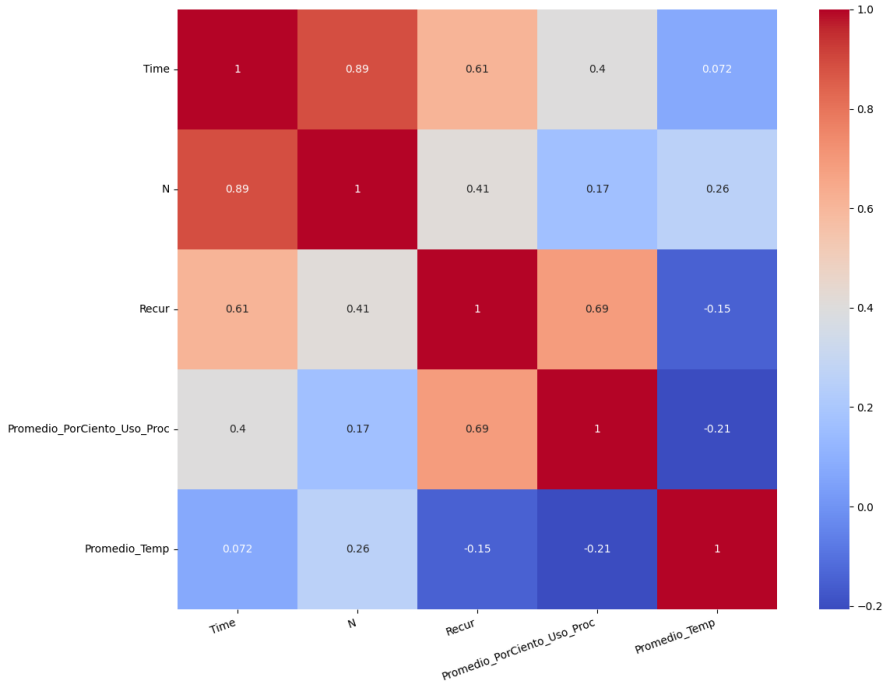

Figura 7.18: Matriz de correlación de Spearman en la variante CUDA recursivo

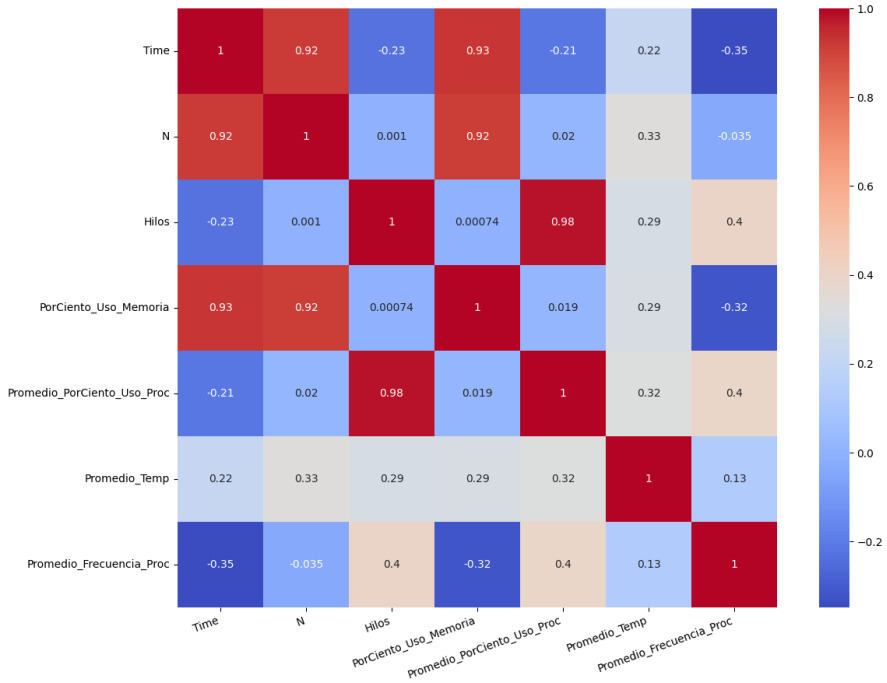

Figura 7.19: Matriz de correlación de Spearman en la variante OMP no recursivo

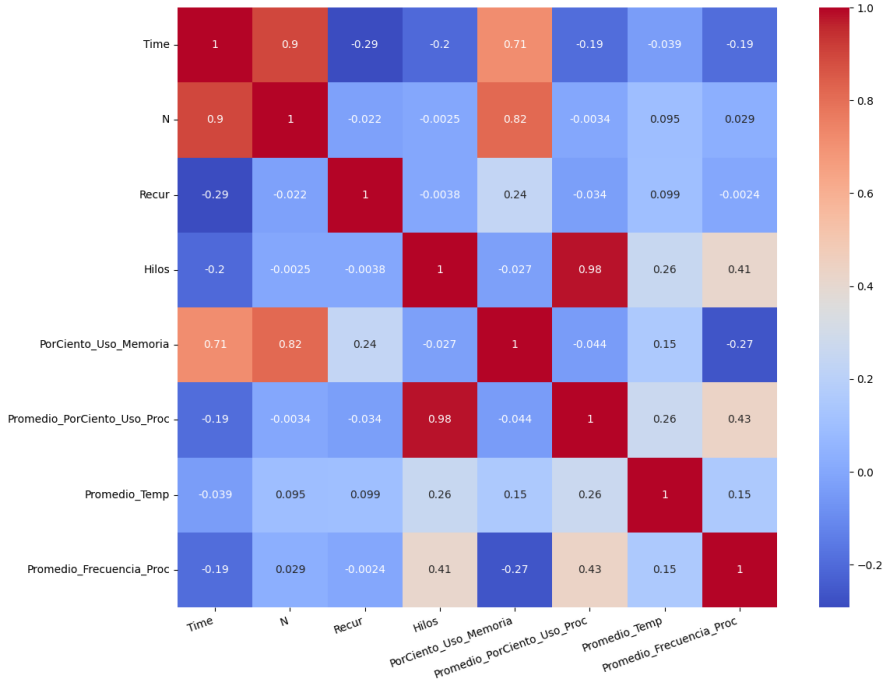

Figura 7.20: Matriz de correlación de Spearman en la variante OMP recursivo

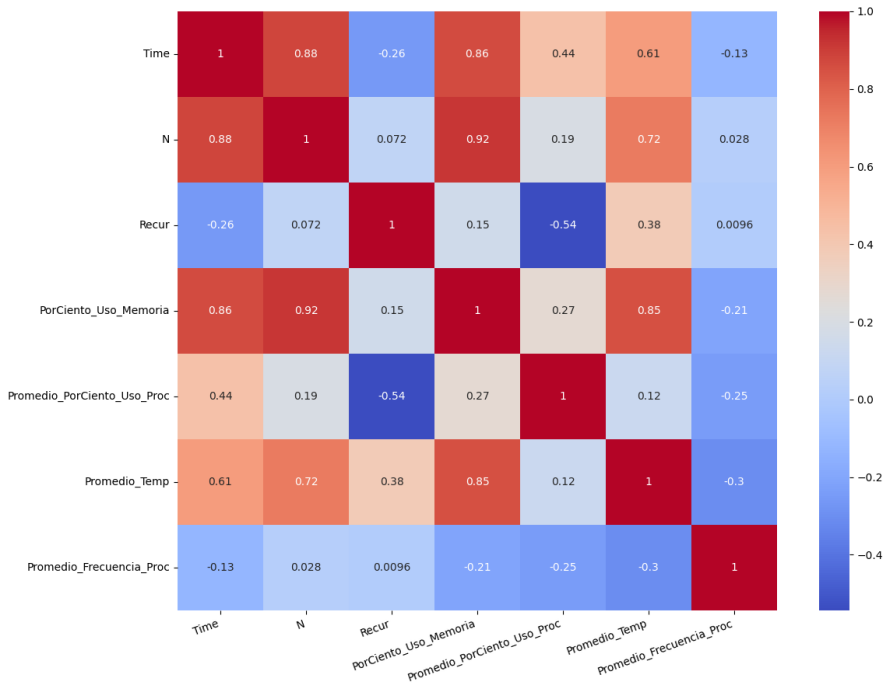

Figura 7.21: Matriz de correlación de Spearman en la variante secuencial

Por consiguiente se procedió a realizar la detección de valores atípicos auxiliándose de la herramienta de los diagramas de cajas y bigotes. Las características que fueron analizadas representan los valores que se obtuvieron durante el monitoreo de la tarea en sus distintas variantes y configuraciones, ya que se entiende que puedan ser las más sensibles a poder obtener errores en mediciones o valores que no reflejen una consonancia con los resultados esperados.

En las gráficas 7.22, 7.23, 7.24, 7.25, 7.26, 7.27 se muestran las columnas donde se obtuvieron valores atípicos para cada una de las variantes. Sin embargo el análisis arrojó que los valores obtenidos no representan incongruencias en los datos, y se deben a casos excepcionales por la forma en que se diseñó el experimento. Por tanto no se decidió realizar ningún tipo de procesamiento adicional con estos datos ya que pueden representar información valiosa para los modelos de regresión.

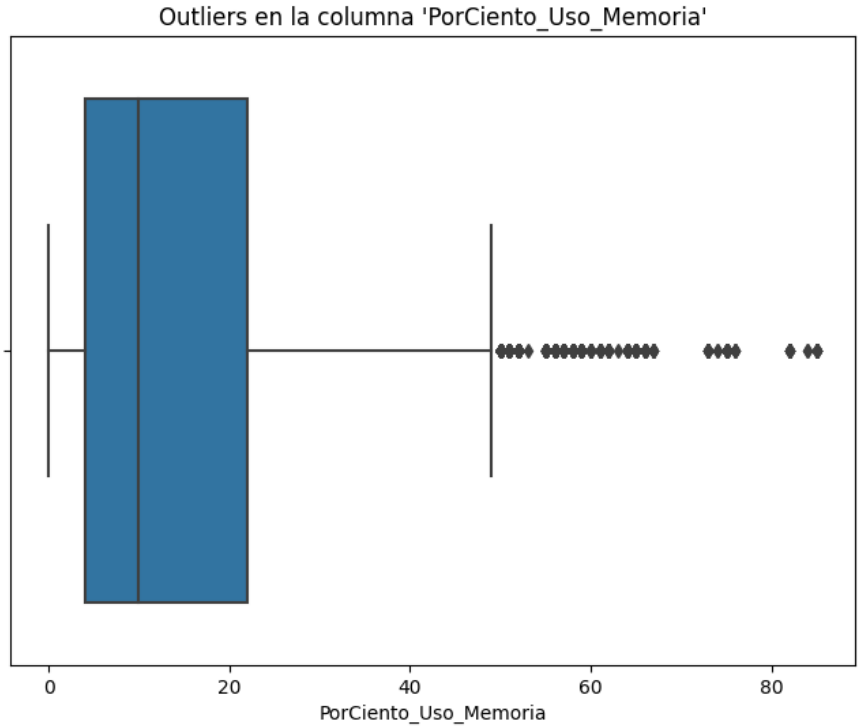

Figura 7.22: Diagrama de caja y bigotes para el por ciento de uso de memoria en la variante OpenMP recursivo

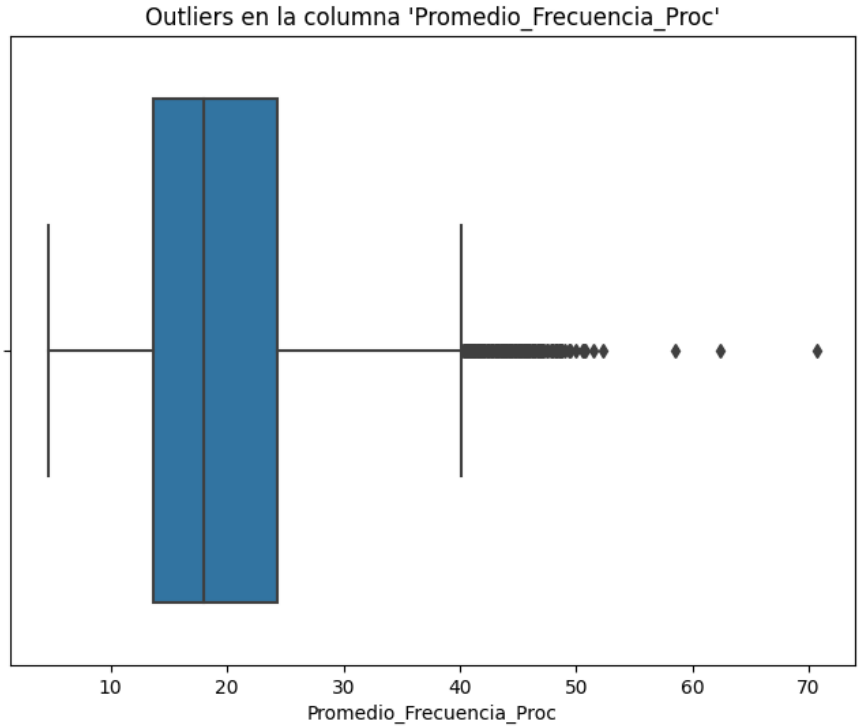

Figura 7.23: Diagrama de caja y bigotes para el promedio de frecuencia del procesador en la variante CUDA con OpenMP recursivo

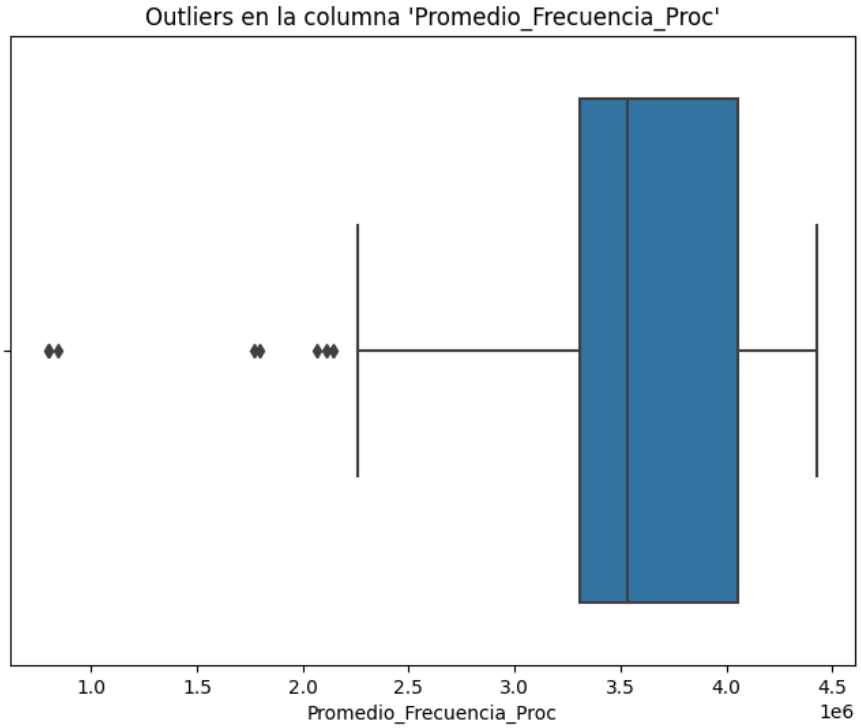

Figura 7.24: Diagrama de caja y bigotes para el promedio de frecuencia del procesador en la variante OpenMP recursivo

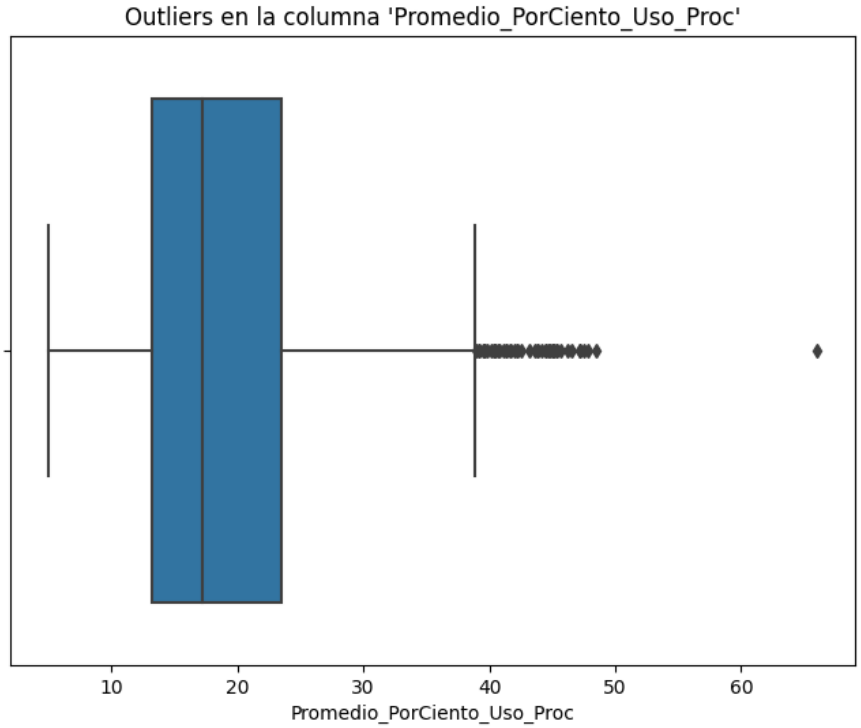

Figura 7.25: Diagrama de caja y bigotes para el por ciento de uso del procesador en la variante CUDA recursivo

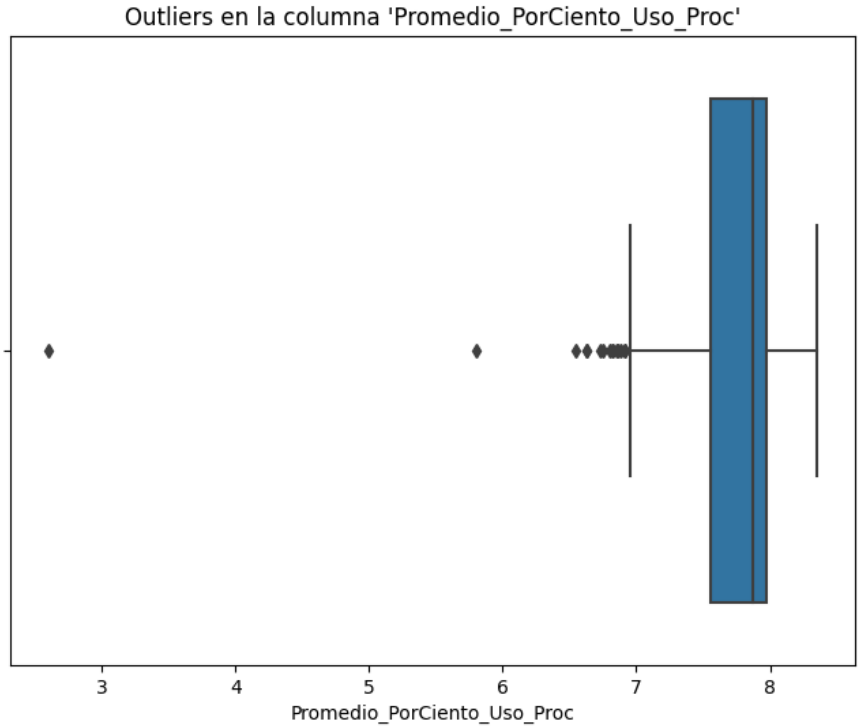

Figura 7.26: Diagrama de caja y bigotes para el por ciento de uso del procesador en la variante secuencial

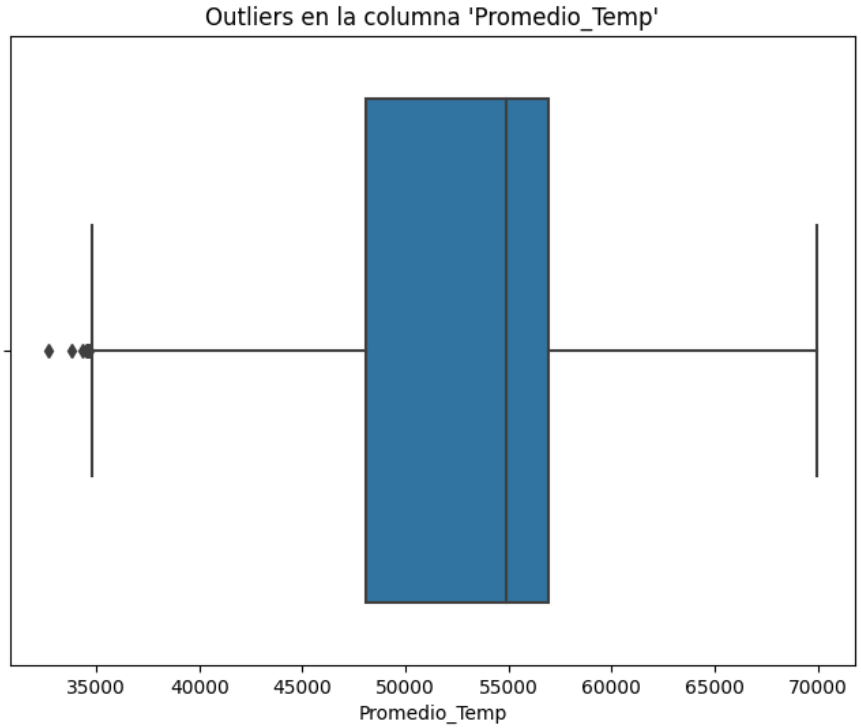

Figura 7.27: Diagrama de caja y bigotes para el promedio de temperatura en la variante OpenMP no recursivo

Una vez que se han identificado los valores atípicos se procedió a encontrar los valores faltantes. En este proceso se identificaron un conjunto de filas con valores faltantes, las cuales se procedieron a eliminar, después de inspeccionar que dichos valores que faltaban se debían a problemas en el monitoreo de las características en ejecución.

Por consiguiente se procedió a realizar la codificación y normalización de los conjuntos de datos. En este proceso para el caso de la codificación en el conjunto de datos de CUDA recursivo y CUDA con OpenMP recursivo, no se realizó ninguna codificación, ya que al solamente realizar el experimento en una plataforma las características pre-ejecución propuestas a codificar no tenían ningún tipo de varianza.

Una vez codificadas y normalizadas las variables para cada conjunto de datos, se procedió a realizar la división de cada uno de los conjuntos de datos, en entrenamiento y prueba. En este caso y a partir de investigaciones previas Nelli (2018) Agarwal (2013) Perner (2002), se decidió dividir un 70 % de cada conjunto de datos para entrenar los regresores y un 30 % de cada conjunto para la prueba y evaluación de los distintos regresores.

El entrenamiento de cada uno de los modelos para las distintas variante se realizó para el caso de la Regresión Lineal Múltiple y el *Random Forest* la biblioteca *Scikit-learn* de Python y en el caso de la Red Neuronal Secuencial y la Red LSTM se utilizó la biblioteca Keras también de Python.

La Red Neuronal utilizada constó de tres capas. La primera es la capa de entrada compuesta por n neuronas siendo n el conjunto de variables predictoras para cada variante. Luego la segunda es una capa oculta densa o sea donde cada neurona en esa capa está conectada a todas las neuronas en la capa anterior, está compuesta por $2n$ neuronas

siendo n la cantidad de variables predictoras. La última es una capa de salida compuesta por una sola neurona, esta capa también es densa.

Para el caso de la red LSTM se utilizaron dos capas, la primera es una capa LSTM compuesta por 80 unidades. La segunda es la capa de salida compuesta por una sola neurona, siendo esta última capa densa.

Para evitar el sobreajuste en ambos modelos se aplicó una técnica de regularización llamada *Dropout* que consiste en "desactivar" o "apagar" algunas neuronas durante la fase de entrenamiento.Esto significa que durante cada paso de entrenamiento, cada neurona tiene la posibilidad de ser ignorada durante la propagación hacia adelante y la propagación hacia atrás Aggarwal (2018).

La idea detrás de esto es que al apagar aleatoriamente ciertas neuronas, se reduce la dependencia de la red en cualquier neurona individual y se fomenta la distribución de la çarga"de aprendizaje a través de todas las neuronas de la capa. Esto puede ayudar a prevenir el sobreajuste, que es una situación en la que la red aprende patrones específicos de los datos de entrenamiento que no se generalizan bien a los datos no vistos Aggarwal (2018).

Para evitar el sobreajuste en el modelo de Regresión Lineal Múltiple se utilizó la técnica de regularización L1 o Lasso en la cual se añade un término de penalización a la función de coste del modelo, quen en este caso fue de 0.1 Khuri (2013).

Por otra parte el *Random Forest* internamente utiliza métodos para evitar el sobreajuste sin embargo, se decidió modificar algunas hiperparámetros que permiten ajustar mejor la dependencia al sobreajuste como por ejemplo :

- **n_estimators**: Este parámetro controla el número de árboles en el bosque. En general, a medida que aumenta el número de árboles, el modelo se vuelve más robusto y menos propenso al sobreajuste. Sin embargo, hay un punto de rendimientos decrecientes después del cual agregar más árboles no mejora significativamente el rendimiento del modelo Liu et al. (2012).
- **max_depth**: Este parámetro controla la profundidad máxima de los árboles. Un árbol más profundo puede capturar más detalles y patrones complejos, pero también es más propenso al sobreajuste. Limitar la profundidad de los árboles puede ayudar a prevenir el sobreajuste Liu et al. (2012).
- **min_samples_leaf** y **min_samples_split**: Estos parámetros controlan el tamaño mínimo de las hojas y los nodos internos, respectivamente. Al aumentar estos valores, se restringe la capacidad del modelo para ajustarse a los datos de entrenamiento, lo que puede ayudar a prevenir el sobreajuste Rajaraman (2012).
- **max_features**: Este parámetro controla el número de características que se consideran al buscar la mejor división. Al reducir el número de características consideradas, se puede aumentar la diversidad de los árboles y hacer que el modelo sea más robusto Liu et al. (2012).

Para poder evaluar los regresores se utilizaron las cuatro métricas descritas en la sección 6.2.5. En las gráficas 7.28, 7.29, 7.30, 7.31 y 7.32 se muestran las evaluaciones de los resultados de cada modelo de regresión, para las métrica mape. La métrica mape como se explicó en la 6.2.5 mide el promedio de los errores absolutos porcentuales. Un valor más bajo de mape indica un mejor rendimiento del modelo, ya que significa que el modelo tiene, en promedio, un menor porcentaje de error en sus predicciones. Se puede observar que el *Random Forest* resultó ser el modelo que mejores resultados arrojó para la estrategia de CUDA con OpenMP recursivo, CUDA recursivo, OpenMP no recursivo y OpenMP recursivo, como se muestra en las gráfica 7.28, 7.29, 7.30, 7.31. Por otra parte la red neuronal LSTM también muestra muy buenos resultados llegando a tener en las predicciones para la estrategia secuencial, resultados de la métrica mape más bajas que el modelo de *Random Forest*.

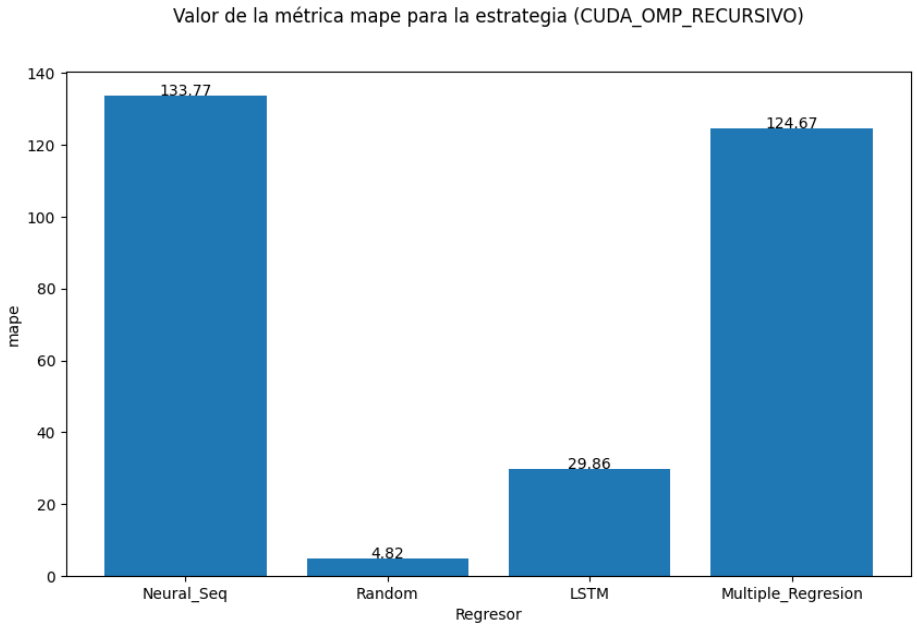

Figura 7.28: Métrica de evaluación mape para el conjunto de regresores utilizados, en la estrategia CUDA con OpenMP recursivo

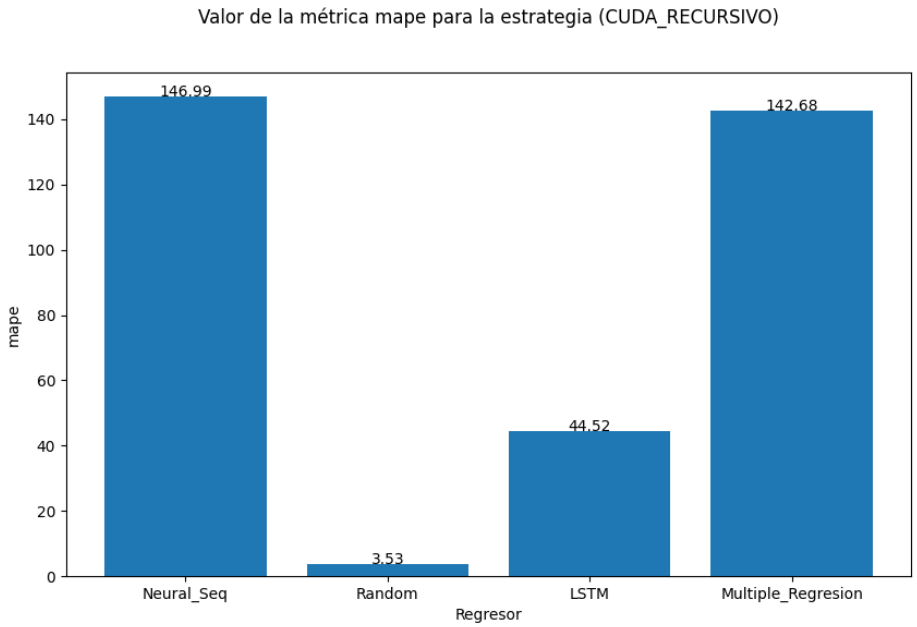

Figura 7.29: Métrica de evaluación mape para el conjunto de regresores utilizados, en la estrategia CUDA recursivo

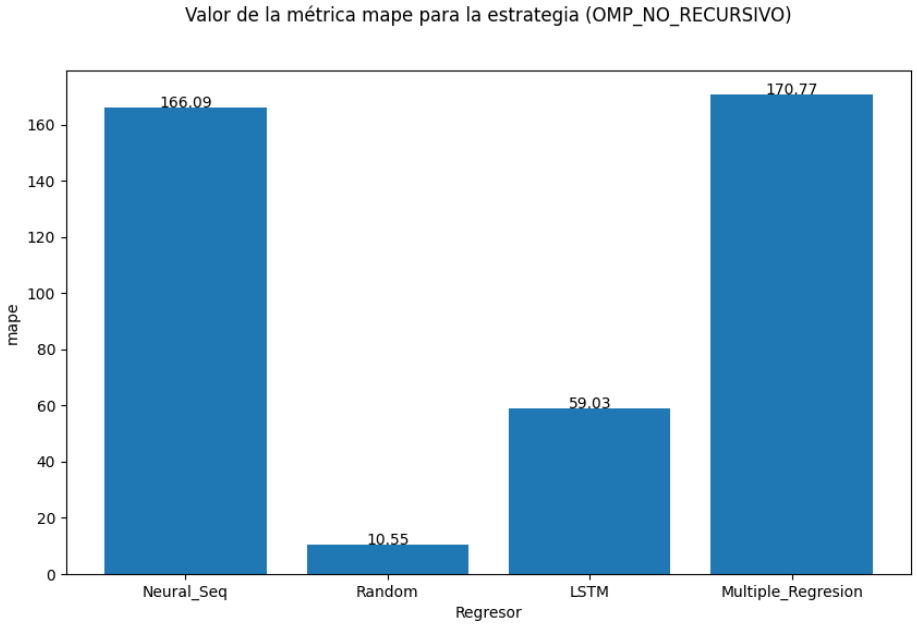

Figura 7.30: Métrica de evaluación mape para el conjunto de regresores utilizados, en la estrategia OpenMP no recursivo

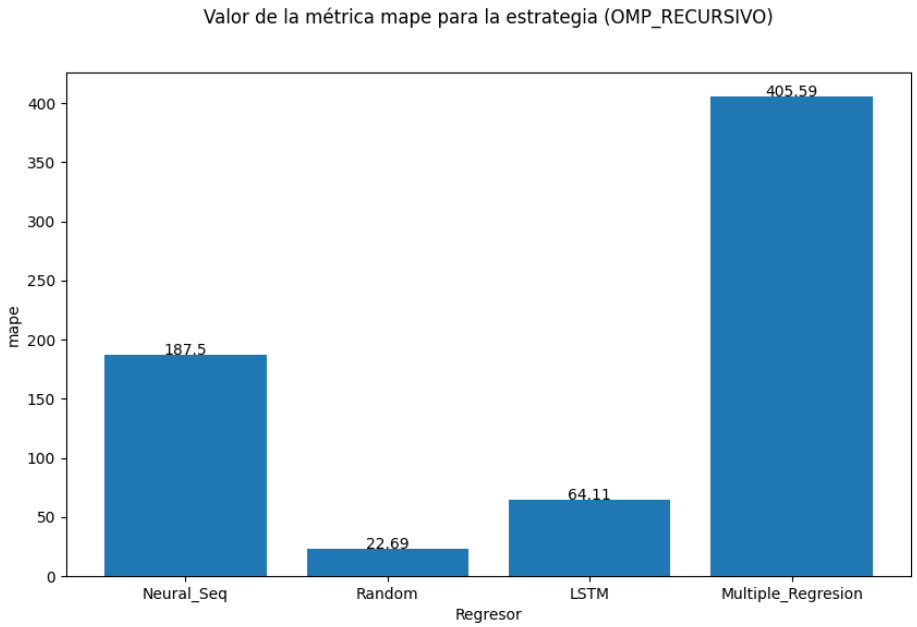

Figura 7.31: Métrica de evaluación mape para el conjunto de regresores utilizados, en la estrategia OpenMP recursivo

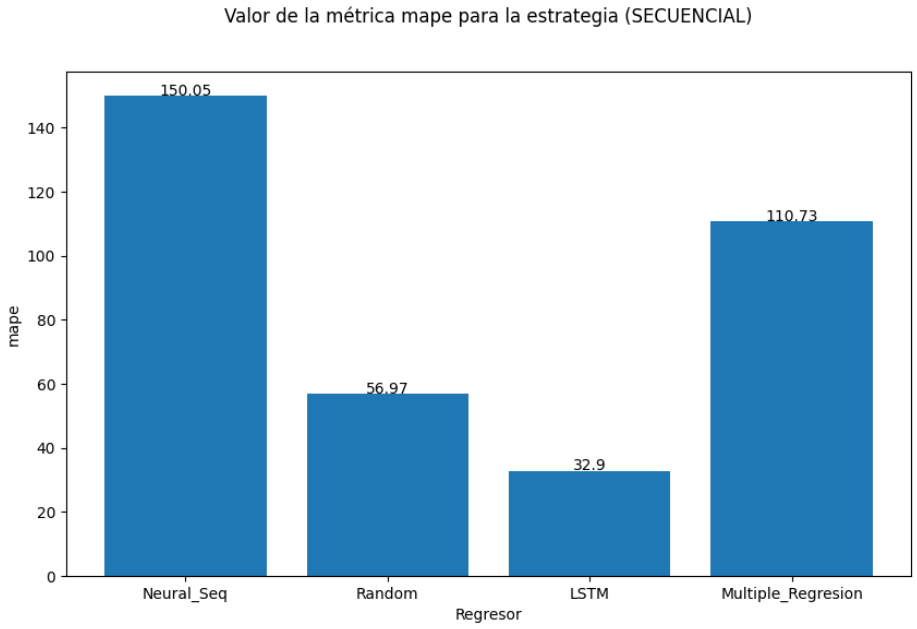

Figura 7.32: Métrica de evaluación mape para el conjunto de regresores utilizados, en la estrategia Secuencial

Por otra parte en las gráficas 7.33, 7.34, 7.35, 7.36 y 7.37 se puede observar la evaluación de la importancia de cada una de las características en cada estrategia determinada por el *Random Forest*.

La importancia de una característica en el modelo de Random Forest se interpreta como la contribución promedio de esa característica a la reducción del error en las predicciones del modelo. Una característica con una alta importancia ha contribuido en gran medida a

la reducción del error en las predicciones del modelo, mientras que una característica con una baja importancia ha contribuido poco. En la gráficas 7.33 y 7.34, se puede observar como la característica que mayor contribución en la predicción del tiempo para estrategia de CUDA con OpenMP recursivo y CUDA recursivo fue el nivel de recursividad. En la variante OpenMP no recursivo la característica que más contribuyó para el modelo fue el porciento de uso de memoria, como se muestra en la gráfica 7.35, y para el caso de las variantes OpenMP recursivo y Secuencial la característica que más contribuyó fue el tamaño de las matrices expresado como N, como se muestra en las gráficas 7.36 y 7.37

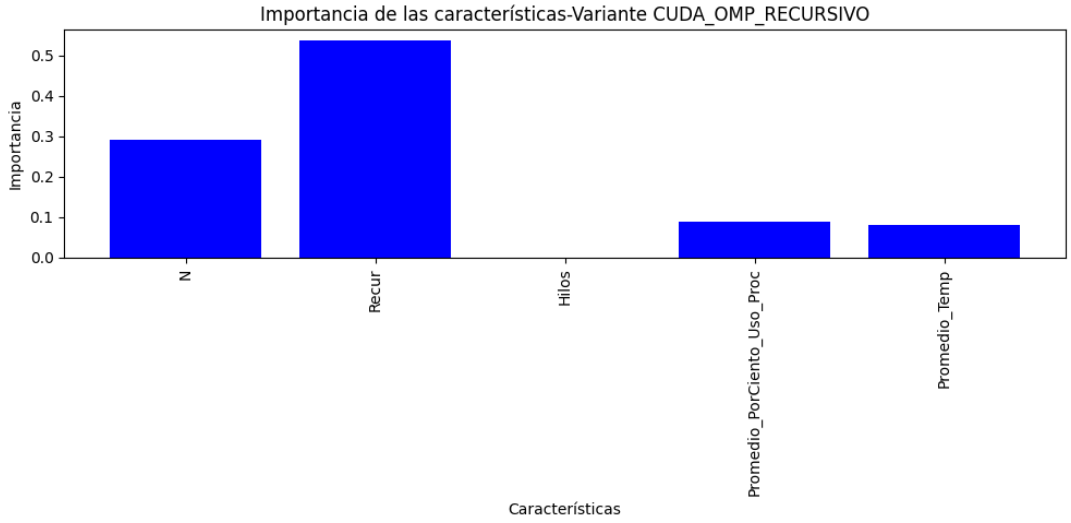

Figura 7.33: Evaluación de características por *Random Forest* en la estrategia CUDA con OpenMP recursivo

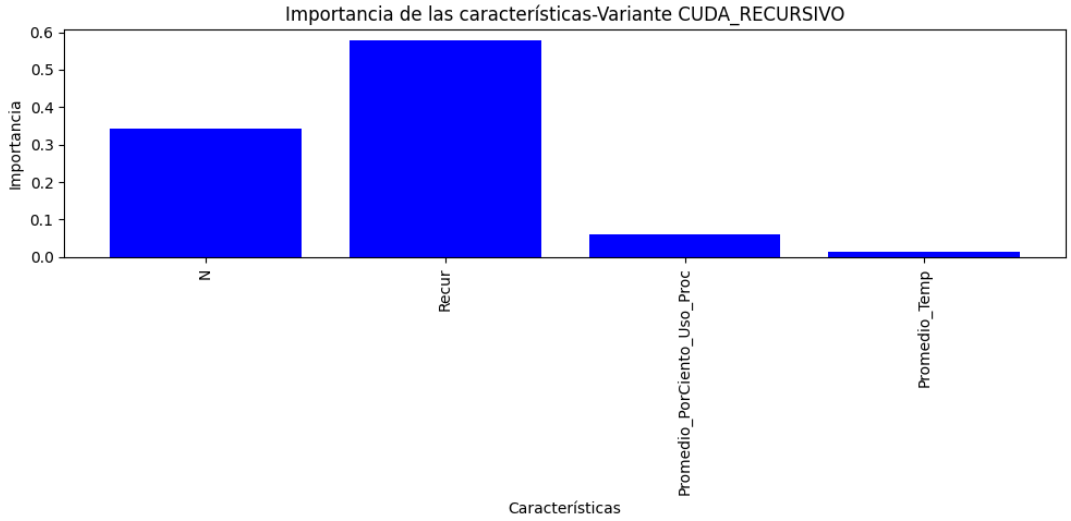

Figura 7.34: Evaluación de características por *Random Forest* en la estrategia CUDA recursivo

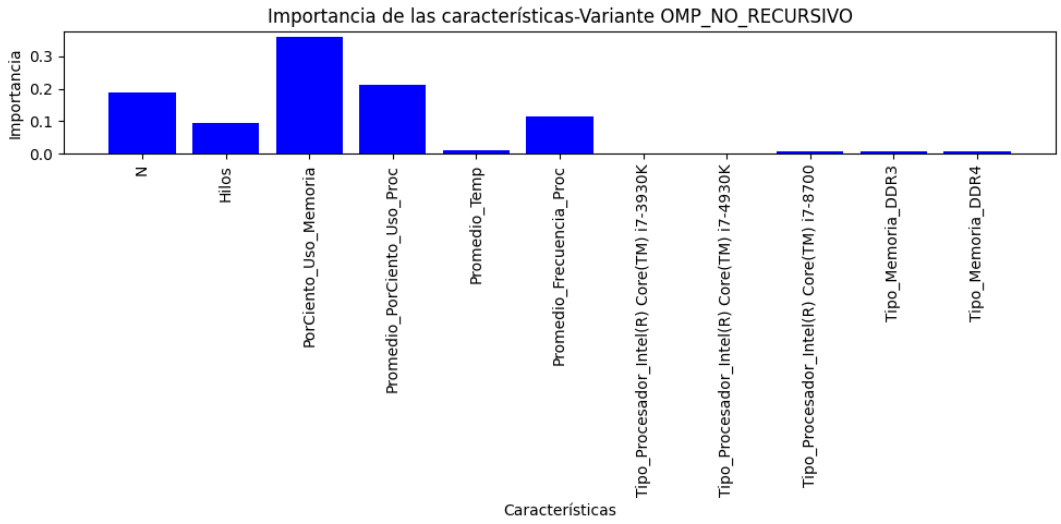

Figura 7.35: Evaluación de características por *Random Forest* en la estrategia OpenMP no recursivo

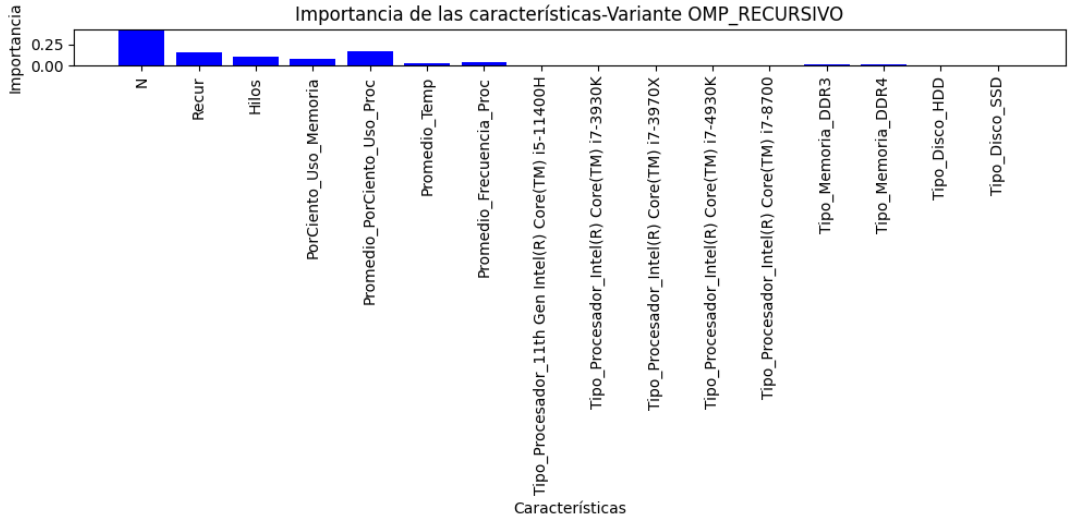

Figura 7.36: Evaluación de características por *Random Forest* en la estrategia OpenMP recursivo

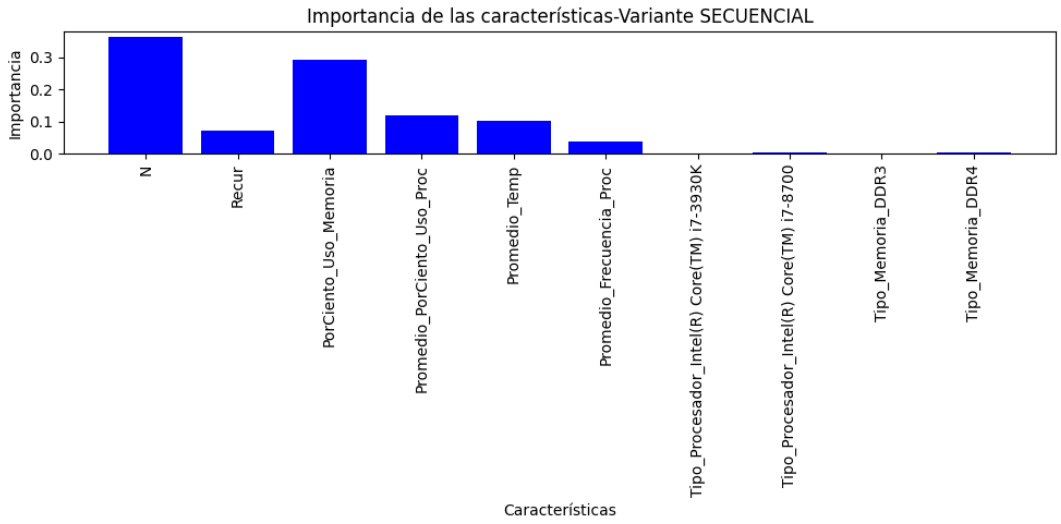

Figura 7.37: Evaluación de características por *Random Forest* en la estrategia Secuencial

Por consiguiente en las gráficas 7.38, 7.39, 7.40, 7.41, 7.42, 7.43, 7.44, 7.45, 7.46 y 7.47 se muestra la relación entre los valores predichos y los valores reales, para el modelo *Random Forest* y la red LSTM en las distintas variantes usadas. Es conveniente mencionar que los datos están normalizados, por tanto, van a existir valores tanto reales como predichos por el modelo en cada uno de los ejes que son negativos.

Este tipo de gráfico, a menudo llamado gráfico de dispersión de predicciones vs reales, es una herramienta útil para visualizar qué tan bien los modelos están prediciendo los valores reales.

En estos gráficos, cada punto representa una observación del conjunto de datos para el tiempo. La coordenada x del punto es el valor real de la observación del tiempo, y la coordenada y es el valor que el modelo predijo para esa observación.

La línea de identidad es una línea con pendiente 1 y ordenada al origen 0. Si las predicciones son perfectas, entonces para cada observación, el valor real y el valor predicho serían exactamente iguales, por lo que cada punto caería exactamente en esta línea. Esto representa el caso ideal.

Si los puntos están cerca de la línea de identidad, eso indica que las predicciones son bastante precisas. Si los puntos están dispersos lejos de la línea de identidad, eso indica que las predicciones están lejos de los valores reales.

Se puede observar que el modelo de *Random Forest* de manera general, tuvo predicciones más ajustadas a la linea de identidad, corroborando el resultado de las métricas mape, en donde se visualizaba la tasa de error baja que este modelo proporcionó para este trabajo.

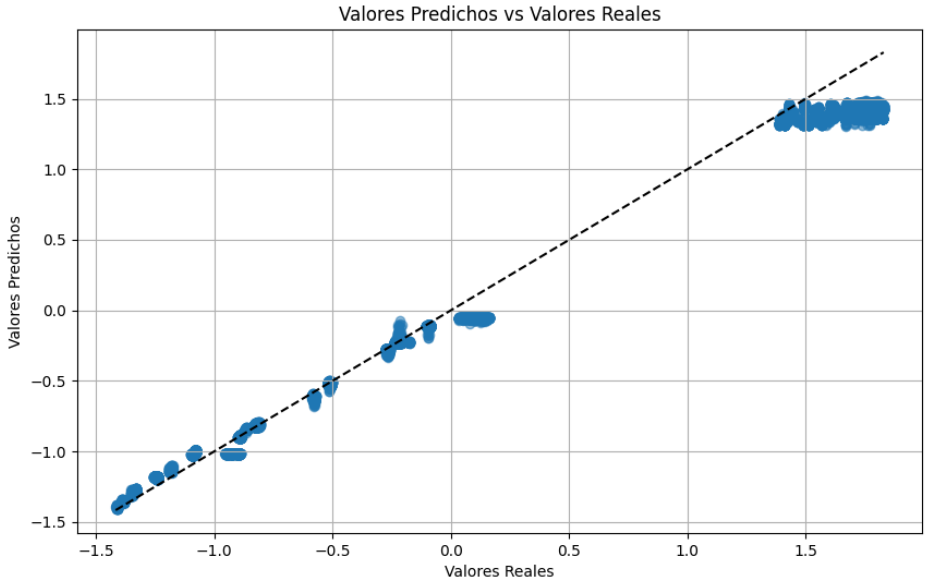

Figura 7.38: Relación entre los valores predichos y reales para la red LSTM en la variante CUDA con OpenMP recursivo

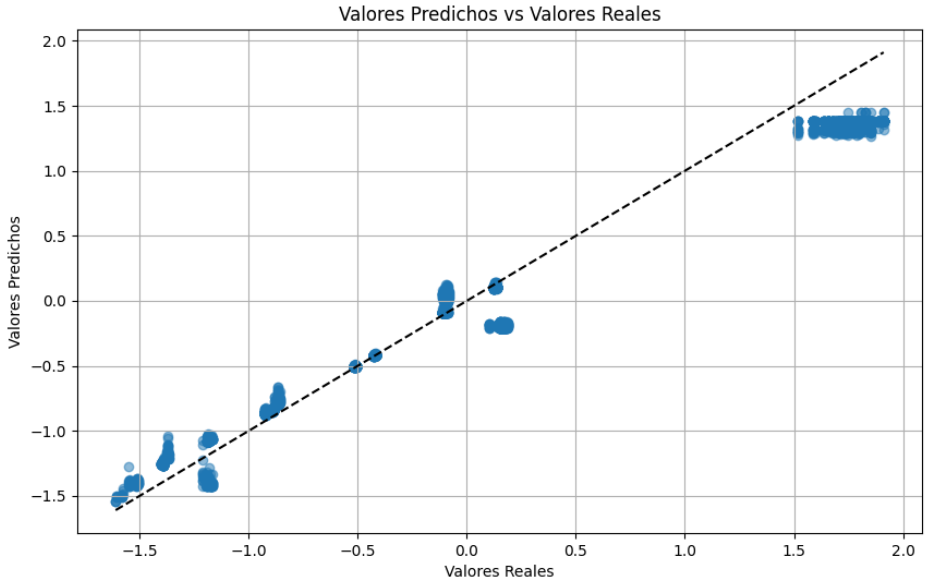

Figura 7.39: Relación entre los valores predichos y reales para la red LSTM en la variante CUDA recursivo

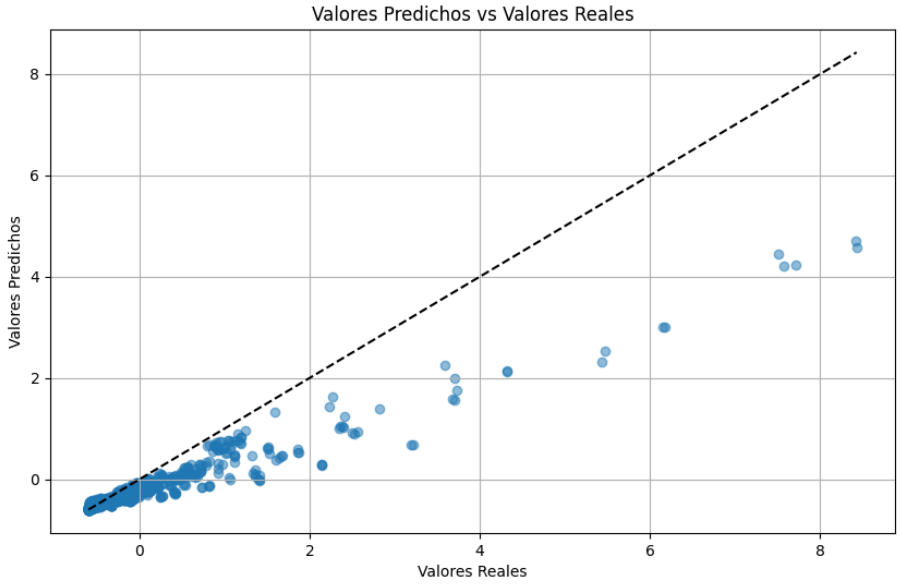

Figura 7.40: Relación entre los valores predichos y reales para la red LSTM en la variante OpenMP no recursivo

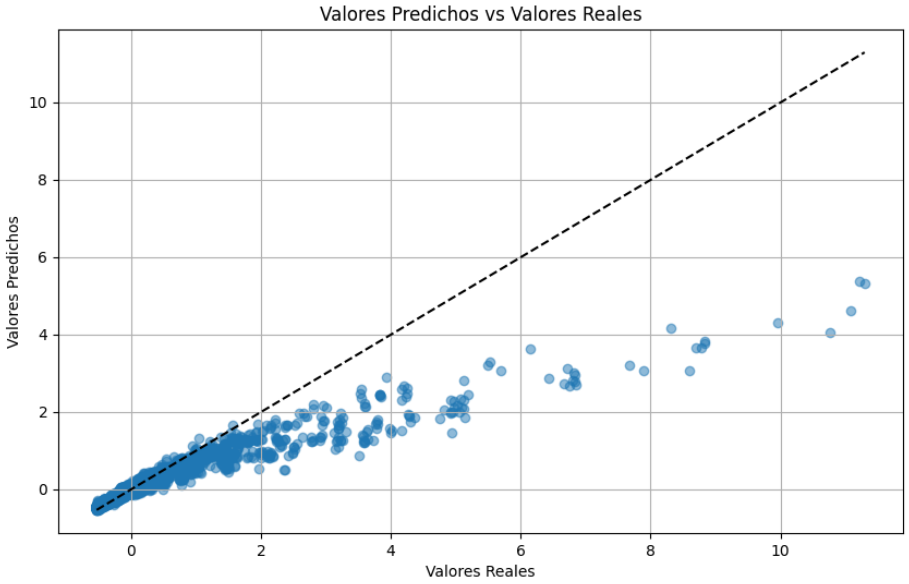

Figura 7.41: Relación entre los valores predichos y reales para la red LSTM en la variante OpenMP recursivo

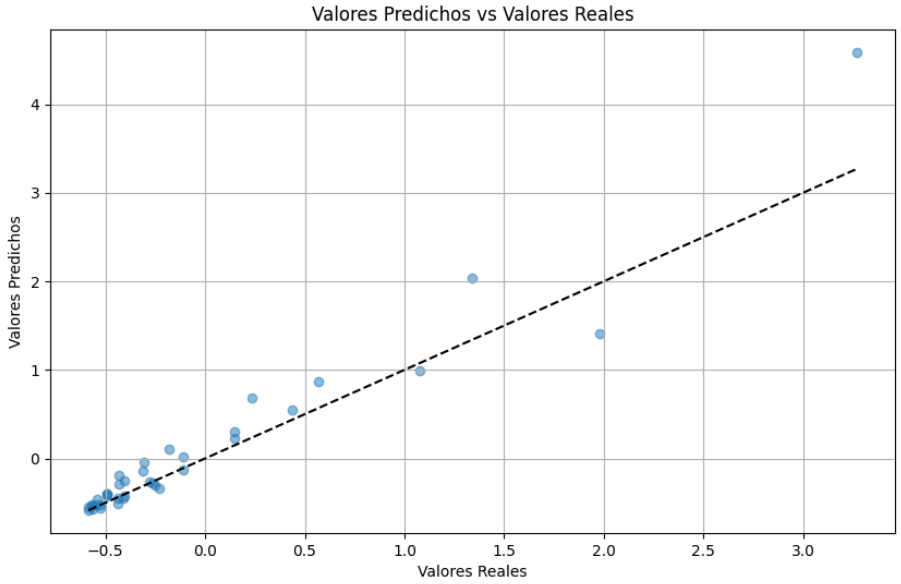

Figura 7.42: Relación entre los valores predichos y reales para la red LSTM en la variante secuencial

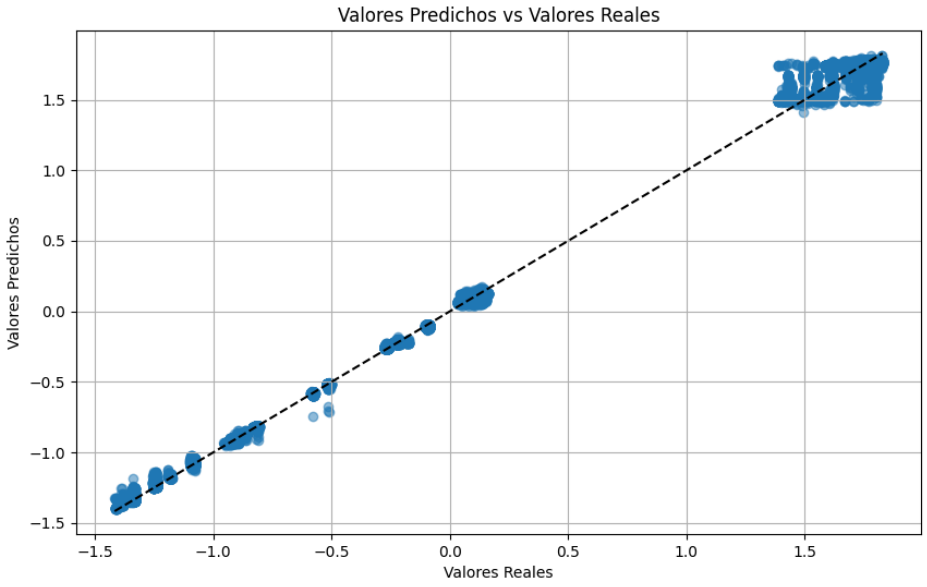

Figura 7.43: Relación entre los valores predichos y reales para el *Random Forest* en la variante CUDA con OpenMP recursivo

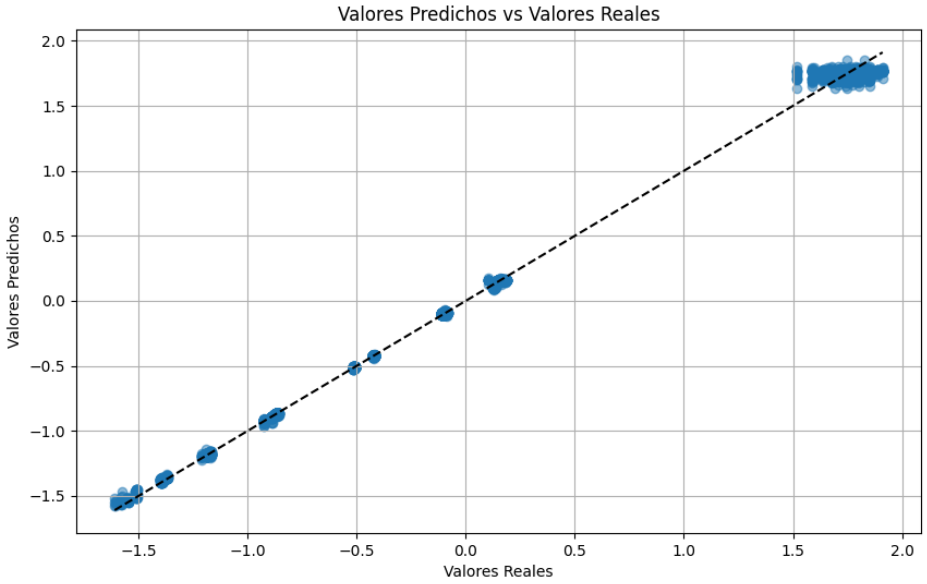

Figura 7.44: Relación entre los valores predichos y reales para el *Random Forest* en la variante CUDA recursivo

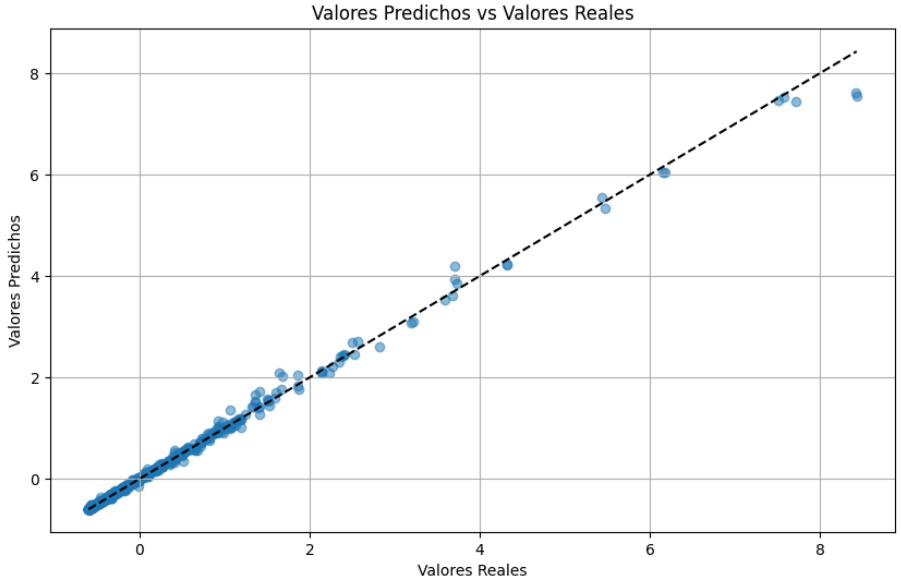

Figura 7.45: Relación entre los valores predichos y reales para el *Random Forest* en la variante OpenMP no recursivo

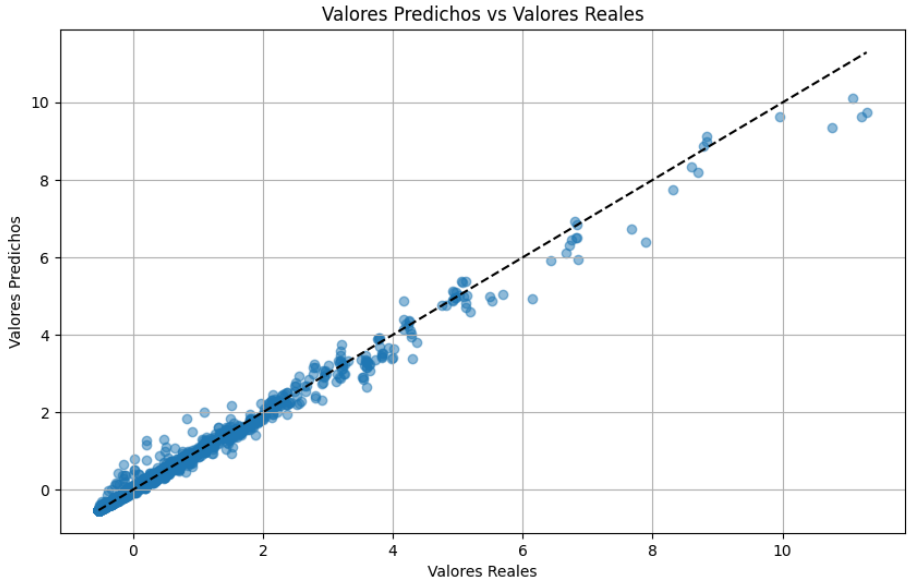

Figura 7.46: Relación entre los valores predichos y reales para el *Random Forest* en la variante OpenMP recursivo

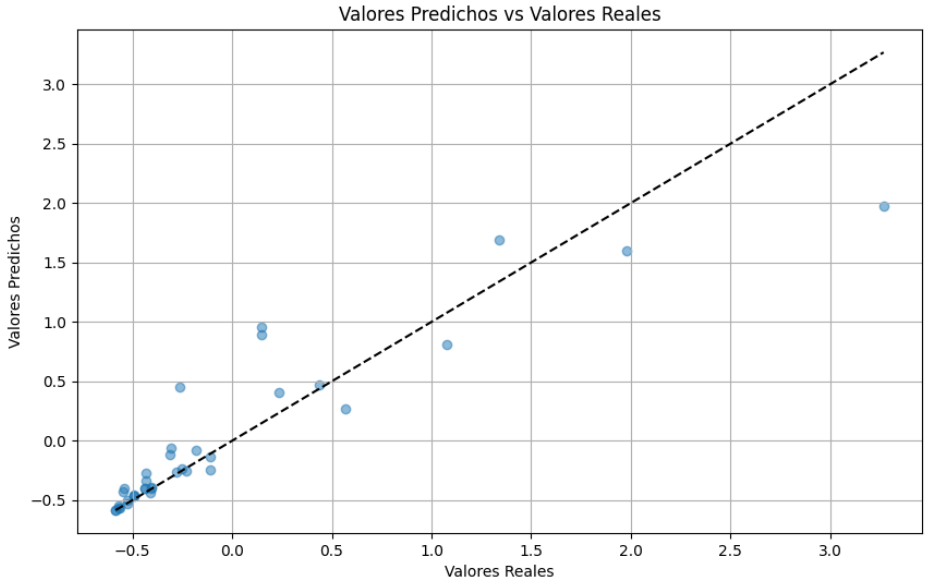

Figura 7.47: Relación entre los valores predichos y reales para el *Random Forest* en la variante secuencial

Capítulo 8

Conclusiones y trabajos futuros

El propósito principal de este trabajo fue desarrollar una caracterización detallada de tareas y recursos que pudiera emplearse como entrada para diversos algoritmos de Inteligencia Artificial. El objetivo de estos algoritmos es predecir el valor de un indicador que pueda integrarse en una función objetivo para optimizar la calendarización estática de tareas en entornos heterogéneos. El proyecto se desglosó en objetivos específicos y se estableció una metodología de trabajo que guió de manera efectiva el desarrollo de la investigación.

En primer lugar, se estableció un marco de referencia para sentar las bases de cómo se iban a interrelacionar, desde la perspectiva de este estudio, conceptos clave en la calendarización estática heterogénea. Estos incluyen el significado de tareas, procesos, hilos, trabajos y grafos DAG, y cómo cada uno de ellos se interrelaciona para abordar el problema de la calendarización. Resultó crucial definir el concepto de sistemas distribuidos, debido a su estrecha relación con los calendarizadores. Los sistemas distribuidos, que consisten en múltiples computadoras interconectadas que trabajan juntas para realizar tareas, son un componente esencial en muchos entornos de calendarización. Los calendarizadores, a su vez, son responsables de asignar tareas a los diferentes nodos en un sistema distribuido de manera eficiente.

Una comprensión sólida de cómo funcionan los sistemas distribuidos es fundamental para abordar el problema de la calendarización. Esta comprensión permite una mejor evaluación de las estrategias de calendarización y facilita la optimización de la asignación de tareas en entornos distribuidos. En consecuencia, se abordaron temas estrechamente vinculados con los dispositivos o unidades de procesamiento, como la virtualización. La virtualización es una tecnología que permite crear múltiples entornos simulados o dedicados a partir de un único recurso físico, lo que puede tener un impacto significativo en la forma en que se calendarizan las tareas. Se definieron algunos descriptores clave de las unidades de procesamiento, como los discos, las memorias y los procesadores. Estos descriptores son esenciales para entender las capacidades y limitaciones de las unidades de procesamiento, y por lo tanto son fundamentales para la calendarización eficiente de las tareas. Por ejemplo, los discos pueden variar en términos de capacidad de almacenamiento y velocidad de lectura/escritura, las memorias pueden variar en términos de tamaño y velocidad, y los procesadores pueden variar en términos de número de núcleos y velocidad de reloj. Estas variaciones pueden tener un impacto significativo en el rendimiento de las tareas y, por lo tanto, en la eficacia de la calendarización.

Se establecieron comparaciones entre las tecnologías más utilizadas dentro de los descriptores asociados al hardware de las plataformas. Esto proporcionó una visión valiosa

de las ventajas y desventajas relativas de diferentes tecnologías, lo que puede informar las decisiones sobre qué tecnologías utilizar en un sistema de calendarización.

Debido a la relación entre los sistemas heterogéneos, la concurrencia y el paralelismo, es crucial entender los elementos clave de algunas de las estrategias de paralelización más utilizadas. La concurrencia implica la gestión y el manejo de múltiples tareas que se ejecutan aparentemente al mismo tiempo, aunque pueden no estar ejecutándose simultáneamente en unidades de procesamiento separadas. La paralelización es el proceso de dividir una tarea en subprocesos más pequeños que pueden ser ejecutados simultáneamente, lo que puede mejorar significativamente la eficiencia y el rendimiento. Las estrategias de paralelización, como la paralelización de datos, la paralelización de tareas y la paralelización de instrucciones, tienen diferentes ventajas y desventajas, y la elección de la estrategia correcta puede tener un impacto significativo en la eficacia de la calendarización en un sistema heterogéneo.

Además, es importante comprender algunos conceptos sobre la programación híbrida heterogénea. La programación híbrida heterogénea es un enfoque que combina diferentes tipos de unidades de procesamiento o dispositivos en un solo sistema, como CPUs y GPUs, para maximizar el rendimiento y la eficiencia. Este enfoque puede ser especialmente útil en sistemas heterogéneos, donde diferentes tareas pueden ser más adecuadas para diferentes tipos de unidades de procesamiento. Comprender estos conceptos puede ayudar a optimizar la calendarización en sistemas heterogéneos, permitiendo que cada tarea sea asignada a la unidad de procesamiento más adecuada para ella.

En términos generales, las conclusiones obtenidas resaltan la importancia fundamental de la selección de características específicas, como los tipos de memoria, procesadores y discos, en el proceso de calendarización de tareas. Estas elecciones tienen un efecto directo en cómo una tarea se desempeña en una unidad de procesamiento determinada: pueden influir en su rendimiento, ya sea aumentándolo o reduciéndolo. En otras palabras, las decisiones acerca de los componentes tecnológicos que se utilizan en la ejecución de una tarea tienen un impacto significativo en cuán eficientemente se realiza.

Al analizar el concepto de "tarea" se ha delineado un nivel de detalle para poder discernir las diferencias entre tareas, procesos y trabajos que se utilizan en el contexto de sistemas operativos y distribuidos. Esta clarificación es crucial ya que cada uno de estos términos conlleva un significado específico en el contexto de la calendarización de tareas. Definir estas diferencias de manera adecuada proporciona una base sólida para abordar los desafíos de la calendarización, permitiendo una gestión más efectiva de las distintas unidades de trabajo.

En conjunto, estas conclusiones indican que las decisiones sobre componentes clave y la comprensión precisa de conceptos fundamentales son esenciales para optimizar la calendarización de tareas en entornos heterogéneos. La calidad de estas elecciones y comprensiones afecta directamente la eficacia general de la calendarización y, por lo tanto, la eficiencia operativa y el rendimiento en sistemas complejos.

En los Capítulo 3 y 4 se realizó una revisión exhaustiva del estado del arte en lo que respecta a los calendarizadores y más específico los calendarizadores en entornos heterogéneos, las políticas de calendarización y los algoritmos utilizados para la calendarización. Esto implicó una exploración detallada de la literatura existente para entender las técnicas y estrategias actuales, así como las tendencias emergentes en el campo de la calendarización.

Se exploró la complejidad del problema de la calendarización y los tipos de algoritmos que se utilizan para abordarlo. La calendarización es un problema notoriamente difícil, con

muchas variantes que son NP-completas. Sin embargo, existen varios algoritmos heurísticos y metaheurísticos que se han desarrollado para abordar este problema. Comprender estos algoritmos y cómo se aplican al problema de la calendarización es un componente esencial de este estudio.

En términos generales, se ha llegado a la conclusión de que muchas de las heurísticas y algoritmos diseñados para la calendarización de tareas presentan una limitación importante: la falta de mecanismos efectivos para generar matrices de costos de ejecución. Estos métodos asumían la existencia previa de tales matrices, lo que puede ser una restricción en la práctica. Además, se observó que la combinación de enfoques heurísticos con algoritmos de inteligencia artificial ha demostrado ser particularmente eficaz en la obtención de resultados satisfactorios en el ámbito de la calendarización estática, sobre todo en entornos caracterizados por su heterogeneidad.

La carencia de mecanismos eficientes para la generación de matrices de costos de ejecución implica que muchos enfoques de calendarización asumen información que puede no estar fácilmente disponible o ser costosa de obtener. Esta limitación resalta la necesidad de desarrollar metodologías más flexibles y adaptables, capaces de trabajar incluso cuando no se disponga de todos los datos requeridos de antemano.

Por otro lado, la integración de heurísticas con algoritmos de inteligencia artificial emergió como una estrategia altamente prometedora. Este enfoque combina métodos que imitan la intuición humana con técnicas avanzadas de aprendizaje automático, lo que ha demostrado ser especialmente eficaz en la resolución de problemas de calendarización estática. Este éxito es especialmente notorio en entornos heterogéneos, donde las complejidades y las variaciones entre las unidades de procesamiento demandan soluciones más sofisticadas.

En el Capítulo 5, se comenzó estableciendo la conexión entre la predicción del tiempo de ejecución y la calendarización. Se identificaron tres contextos clave en los que la predicción del tiempo de ejecución juega un papel crucial en la calendarización. Estos contextos, o escenarios, son:

1. Escenario de múltiples recursos y una sola tarea: En este escenario, se tiene una única tarea que puede ser ejecutada en varios recursos disponibles. La predicción precisa del tiempo de ejecución puede ayudar a seleccionar el recurso más adecuado para ejecutar la tarea.

2. Escenario de una sola unidad de procesamiento y múltiples tareas: En este caso, se tiene un conjunto de tareas que deben ser ejecutadas en un único recurso. La predicción del tiempo de ejecución puede ser útil para determinar el orden en que se deben ejecutar las tareas para maximizar la eficiencia.

3. Escenario de múltiples recursos y múltiples tareas: Este es el escenario más complejo, en el que se tienen varias tareas que pueden ser ejecutadas en varios recursos. En este caso, la predicción del tiempo de ejecución puede ser crucial para determinar una matriz de costos que sirva como entrada al algoritmo de calendarización que pueda realizar un asignación óptima de tareas a recursos.

Estos escenarios representan los distintos contextos de calendarización que pueden surgir en un sistema, y subrayan la importancia de tener predicciones precisas del tiempo de ejecución.

A partir de esta comprensión, se llegó a la conclusión de que la mejor estrategia sería realizar una caracterización conjunta de tareas y recursos. Esta caracterización se basaría

en la función objetivo que se desea optimizar. En otras palabras, en lugar de tratar las tareas y los recursos como entidades separadas, se considerarían en conjunto para obtener una visión más completa y precisa de cómo optimizar la calendarización.

Además, se entendió que la salida de este proceso de caracterización sería el producto cartesiano de tareas y recursos. Cada elemento de este producto representaría una combinación específica de tarea y recurso, y el valor de este elemento sería el indicador a optimizar, en este caso, el tiempo de ejecución. Esta perspectiva permite una optimización más efectiva, ya que se tiene en cuenta tanto la naturaleza de la tarea como las características del recurso en el que se ejecuta.

En este contexto, cada elemento del producto cartesiano $T \times R$ representa una combinación específica de tarea y recurso, y el valor de este elemento sería el indicador a optimizar, en este caso, el tiempo de ejecución. Es decir, para cada par (t, r) en $T \times R$, se tiene un tiempo de ejecución asociado.

Para resolver este producto cartesiano y predecir el tiempo como indicador de interés, se propone el uso de métodos de regresión. Estos métodos pueden modelar la relación entre las características de las tareas, los recursos y el tiempo de ejecución, permitiendo así predecir el tiempo de ejecución para nuevas combinaciones de tareas y recursos.

Por tanto, se revisaron algunos tipos de regresión existentes entendiendo conceptos como, la regresión paramétrica y no paramétrica, y luego se escogieron cuatro regresores, el *Random Forest*, la Regresión Lineal Múltiple, una Red Neuronal Secuencial, y una Red LSTM, basándose la elección en las fortalezas de cada uno para modelar la predicción del tiempo, dada las características de las tareas y los recursos.

Se realizó una revisión exhaustiva de los diferentes tipos de regresión existentes, con el objetivo de entender a fondo los conceptos clave como la regresión paramétrica y no paramétrica. La regresión paramétrica se basa en la suposición de que la función subyacente que se está modelando puede ser descrita por un conjunto de parámetros, mientras que la regresión no paramétrica no hace tales suposiciones, lo que la hace más flexible y capaz de modelar una amplia variedad de relaciones. A partir de esta revisión, se seleccionaron cuatro métodos de regresión para la predicción del tiempo de ejecución. Estos métodos son: el *Random Forest*, la Regresión Lineal Múltiple, una Red Neuronal Secuencial, y una Red LSTM.

El *Random Forest* es un método de regresión basado en árboles de decisión que es conocido por su robustez y capacidad para manejar una amplia variedad de tipos de datos. La Regresión Lineal Múltiple es un método clásico que modela la relación entre dos o más características y una respuesta mediante el ajuste de una ecuación lineal a los datos observados.

Las Redes Neuronales Secuenciales y las Redes LSTM son métodos de regresión basados en redes neuronales que son especialmente útiles para modelar relaciones complejas y no lineales. Las Redes LSTM, en particular, son capaces de aprender dependencias a largo plazo, lo que las hace adecuadas para la predicción del tiempo de ejecución, que puede depender de las características de las tareas y los recursos a lo largo del tiempo.

Además de seleccionar los métodos de regresión, se profundizó en la teoría del entrenamiento de cada uno de ellos. Esto incluyó el estudio de técnicas para la detección de valores atípicos, el análisis de correlación, el análisis de la distribución de la variable objetivo, y métodos de codificación y normalización de los datos. Estos aspectos son fundamentales para garantizar que los modelos de regresión sean capaces de hacer predicciones precisas y útiles.

La elección de estos métodos fue con base en las fortalezas de cada uno para modelar la predicción del tiempo, dadas las características de las tareas y los recursos. Cada uno de estos métodos aporta una perspectiva única y valiosa a la tarea de predecir el tiempo de ejecución, y juntos proporcionan una solución robusta y versátil a este desafío.

En consecuencia, se diseñó un enfoque para entrenar y evaluar los regresores seleccionados. Este enfoque se dividió en dos fases. En la primera fase, se recopilaron datos de la ejecución de tareas en varios dispositivos. En la segunda fase, se llevó a cabo el entrenamiento de los regresores seleccionados utilizando los datos recopilados en la primera fase.

La primera fase implicó la propuesta de tres conjuntos de características: las características pre-ejecución de la tarea, las características pre-ejecución del recurso y las características de monitoreo. Cada característica propuesta reflejaba elementos que se consideraban importantes en su influencia sobre el tiempo de ejecución.

La idea detrás de la primera fase era variar la configuración de las características de la tarea y ejecutarla en diferentes unidades de procesamiento. Cada variación implicaba cambiar el tamaño de entrada **N**, la cantidad de hilos, el nivel de recursividad y la decisión de utilizar o no un dispositivo de almacenamiento secundario. Durante estas ejecuciones, se midieron características como la temperatura del dispositivo durante la ejecución y la frecuencia del procesador, entre otras.

Este enfoque permite recopilar un conjunto de datos diverso que refleja una amplia gama de condiciones de ejecución. Estos datos proporcionaron la base para el entrenamiento y la evaluación de los regresores en la segunda fase.

Una vez recopilados los datos en la primera fase, se procedió a la segunda fase. En esta faase, el conjunto de datos recopilados, pasó por varias etapas de procesamiento. Primero, se vincularon las características para formar un vector único que representara cada variación de configuración con sus respectivas mediciones en el dispositivo determinado.

A continuación, se realiza una agrupación de los datos con base en características críticas. Estas características críticas representaban parámetros que no todas las tareas incluyen, como la utilización de almacenamiento secundario, el nivel de recursividad y la estrategia de paralelización. A partir de esta agrupación, se obtienen varios conjuntos de datos.

Cada uno de estos conjuntos de datos es sometido a un análisis descriptivo, que incluye la detección de valores atípicos, un análisis de la distribución de la variable objetivo (en este caso, el tiempo de ejecución) y un análisis de correlación.

Posteriormente, se realizó un preprocesamiento de cada conjunto de datos, que incluyó la limpieza de los datos, la normalización y la codificación. Para evitar el sobreajuste en los modelos de regresión seleccionados, cada conjunto de datos se dividió en dos partes: datos de entrenamiento y datos de prueba.

Finalmente, se procedió al entrenamiento de los regresores seleccionados utilizando los datos de entrenamiento. Los datos de prueba se utilizaron para evaluar el rendimiento de cada regresor.

En esta propuesta de diseño, se valoró la posibilidad de mezclar la característica del consumo energético con el tiempo de ejecución, como elemento que podría tenerse en cuenta para obtener algún de tipo de prioridad que utilizara alguna heurística determinada para realizar la calendarización. Sin embargo debido a la complejidad que tiene mezclar estos dos elementos, se determinó por el momento, solamente incluir la predicción del tiempo de ejecución.

Para validar la propuesta de diseño, se llevó a cabo una prueba de concepto utilizando el algoritmo de Strassen como tarea. Donde se probó su ejecución variándole las configuraciones de sus características, en distintos dispositivos, para medirle las características señaladas en la propuesta de diseño. Luego se unieron los datos de cada configuración con su respectivas mediciones en los dispositivos, para luego agruparlos y resultado de la agrupación de características, se obtuvieron cinco conjuntos de datos. Estos conjuntos representaban las siguientes configuraciones: CUDA con OpenMP recursivo, CUDA recursivo, OpenMP recursivo, OpenMP no recursivo y Secuencial. Cada uno de estos conjuntos reflejaba la utilización o no de la recursividad y un tipo específico de estrategia de paralelización, además de la variante secuencial donde no se consideró la paralelización. Es importante destacar que, en el caso de Strassen, no se utilizó almacenamiento en disco.

Como resultado final, algunos regresores mostraron errores de predicción bastante bajos en cada variante de los conjuntos de datos obtenidos. Esto demuestra la efectividad de la propuesta y sugiere que puede ser una herramienta valiosa para la predicción del tiempo de ejecución en diferentes escenarios de calendarización.

En un esfuerzo adicional por optimizar el modelo, se realizaron pruebas adicionales para evaluar si era posible mantener una precisión de predicción aceptable utilizando únicamente las características pre-ejecución del algoritmo de Strassen, en lugar de la totalidad de las características propuestas. Sin embargo, los resultados de estas pruebas indicaron que esta estrategia simplificada no era viable, ya que condujo a errores de predicción significativamente más altos. Esto subraya la importancia de una caracterización completa y detallada para la precisión de la predicción en este contexto.

Además, se identificó que el modelo que proporcionaba los errores de predicción más bajos, dada la configuración de características obtenida, era el *Random Forest*. Este resultado destaca la eficacia de este algoritmo en particular para el conjunto de datos y las características específicas consideradas en este estudio. Sin embargo es importante mencionar que la red LSTM también brindó buenos resultados.

Por otra parte, se utilizó el modelo *Random Forest* para evaluar la influencia de cada característica en la construcción del modelo. Este análisis permitió entender mejor la contribución de cada característica a la predicción del tiempo de ejecución.

Determinar la importancia de las características en modelos como LSTM, regresión lineal múltiple y redes neuronales secuenciales puede ser un proceso complejo y que consume mucho tiempo. Aunque la regresión lineal múltiple proporciona coeficientes para cada característica que pueden interpretarse como su importancia relativa, esta interpretación puede ser engañosa si las características están correlacionadas entre sí, como es el caso de los datos recopilados en este trabajo. Además, en el caso de las redes neuronales, la importancia de las características se distribuye a través de la red y no está asociada directamente a ninguna característica individual. Por lo tanto, debido a las limitaciones de tiempo y a la complejidad inherente de estos modelos, no se pudo realizar un análisis detallado de la importancia de las características para estos modelos en el marco de este trabajo.

A partir de los resultados obtenidos, se concluyó que la elección de métodos de regresión como el *Random Forest*, la Regresión Lineal Múltiple, Redes Neuronales Secuenciales y Redes LSTM demostró ser efectiva para abordar la predicción del tiempo de ejecución en entornos heterogéneos. Sin embargo, entre estos métodos, tanto el *Random Forest* como la Red LSTM destacaron como los más sobresalientes en términos de precisión en la predicción.

La revisión detallada de técnicas de análisis, como la detección de valores atípicos, el análisis de correlación, la distribución de la variable objetivo y la normalización de datos, jugó un papel crucial en el fortalecimiento de los modelos de regresión. Estos aspectos contribuyeron significativamente a la capacidad de los modelos para realizar predicciones confiables y proporcionar información valiosa para la calendarización de tareas.

El diseño metodológico propuesto, que abarcó desde la recopilación de datos hasta el entrenamiento y la evaluación de los regresores, se mostró como un marco sólido y efectivo para abordar la predicción del tiempo de ejecución en contextos heterogéneos. La variación de características de tarea y ejecución en diversas unidades de procesamiento garantizó la robustez y la representatividad de los regresores entrenados.

A pesar de la consideración inicial de integrar el consumo de energía en la calendarización, se concluyó que esta interacción compleja requería un enfoque más profundo y quedó fuera del alcance de este estudio. Sin embargo, la metodología propuesta sentó las bases para futuras investigaciones en esta dirección.

La prueba de concepto realizada con el algoritmo de Strassen fortaleció la efectividad de los modelos de regresión en varias configuraciones. Si bien existieron limitaciones en términos de simplificación de características y evaluación exhaustiva de la importancia de cada una en ciertos modelos, los resultados destacaron la viabilidad de la propuesta y su potencial como herramienta valiosa en la predicción precisa del tiempo de ejecución en entornos de calendarización heterogéneos.

A partir de un análisis exhaustivo y riguroso, se ha demostrado que la combinación de **métodos de regresión**, junto con un **análisis detallado y un diseño metodológico sólido**, establece un enfoque altamente prometedor para lograr predicciones precisas del tiempo de ejecución. Tras evaluar diversas alternativas, se destacan dos enfoques particularmente efectivos: el **Random Forest** y la Red **LSTM**.

En el marco de este estudio, el **Random Forest** ha sobresalido como una opción excepcionalmente eficaz para la predicción de indicadores en la calendarización estática en entornos heterogéneos. No obstante, la Red **LSTM** también ha demostrado su valía al proporcionar resultados igualmente satisfactorios.

Esta investigación no solo proporciona resultados concluyentes sobre las técnicas de predicción evaluadas, sino que también sienta las bases para futuros avances en la optimización y planificación de tareas en contextos computacionales complejos. Los resultados obtenidos en esta investigación tienen un impacto directo en la capacidad de mejorar la eficiencia y el rendimiento en entornos donde la calendarización estática es un factor crítico.

En resumen, esta exploración exhaustiva de **desarrollar una caracterización de tareas y recursos** para predecir indicadores relevantes en la calendarización estática en entornos heterogéneos ha culminado en la identificación de enfoques altamente efectivos y en la apertura de nuevas perspectivas para futuras investigaciones en este campo.

Trabajo a futuro

Sin embargo resulta importante mencionar que siguen quedando elementos para mejorar. Por ejemplo, es conocimiento del investigador de este trabajo que resulta bastante complicado tener en ocasiones, los parámetros de ejecución de una determinada tarea para poder predecir su tiempo. Por tanto como propuesta a trabajos futuros se pensó en una forma de poder mejorar este punto. Además de proseguir con el siguiente paso, que sería probar el regresor dentro de un sistema real de calendarización, en donde al

momento que llegaran la/las tareas al sistema éste se generara un programa que estrayera las características de la tarea, extrayera las características del conjunto de recursos que se tuviera e intersectara ambas características para obtener la matriz de costo de ejecución que necesita el algoritmo de calendarización para empezar a funcionar.

En la figura 8.1 se presenta esta propuesta, que es una ampliación integral de las fases que se describieron en este trabajo. La fase III sería directamente utilizar la regresión para generar la matriz de costo, que sirva como entrada al algoritmo de calendarización que se esté utilizando. La fase IV sería una retroalimentación a la fase III, para resolver el problema que se describió anteriormente en donde resulta bastante complicado tener a priori las características en ejecución de una tarea, en una determinada unidad de procesamiento.

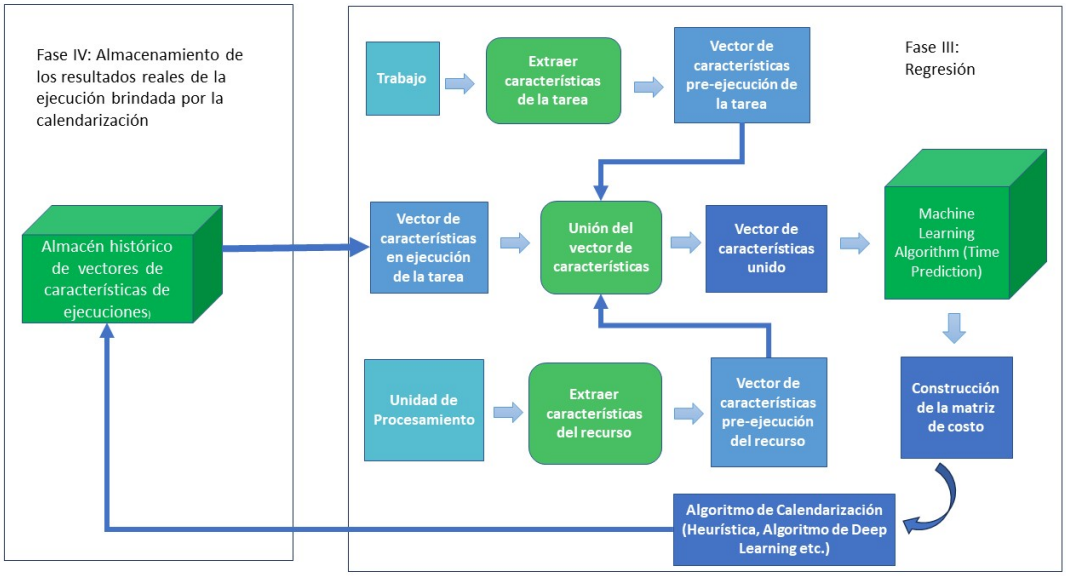

Figura 8.1: Diagrama de de integración para trabajos futuros

Esta fase IV sirve para almacenar los resultados históricos de las ejecuciones de las tareas en las distintas unidades de procesamiento. En este almacén se guardan los vectores integrados de las características pre-ejecución de la tarea, pre-ejecución de los recursos y las características en ejecución. Luego en las fase de retroalimentación se trata de obtener las características en ejecución de una determinada tarea a partir de sus características pre-ejecución y las pre-ejecución del recurso donde se va a ejecutar.Este proceso se podría realizar haciendo uso del algoritmo KNN de la siguiente forma:

Supongamos que se tiene un conjunto de entrenamiento de n observaciones que representan los datos históricos acumulados de observaciones pasadas. Cada observación consta de tres conjuntos de características: las características pre-ejecución de una tarea (denotadas por $\mathbf{x}$) y las características pre-ejecución de un recurso (denotadas por $\mathbf{z}$), así como las características en ejecución correspondientes (denotadas por $\mathbf{y}$). Entonces, el conjunto de entrenamiento es:

$$\{(\mathbf{x}_1, \mathbf{z}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{z}_2, \mathbf{y}_2), \ldots, (\mathbf{x}_n, \mathbf{z}_n, \mathbf{y}_n)\}$$

Ahora, supongamos que se tiene una nueva observación para la cual se quiere predecir las características en ejecución. Esta observación tiene características pre-ejecución de la

tarea $\mathbf{x}$ y características pre-ejecución del recurso $\mathbf{z}$.

Para hacer esta predicción con k-NN, primero se calcula la distancia entre la nueva observación $(\mathbf{x}, \mathbf{z})$ y cada observación en el conjunto de entrenamiento. Para esto se puede por ejemplo usar la distancia euclidiana:

$$D((\mathbf{x}, \mathbf{z}), (\mathbf{x}_i, \mathbf{z}_i)) = \sqrt{(\mathbf{x} - \mathbf{x}_i)^2 + (\mathbf{z} - \mathbf{z}_i)^2}$$

Luego, se selecciona las k observaciones en el conjunto de entrenamiento que tienen la menor distancia a la nueva observación. Estos representarían los k "vecinos más cercanos".

Finalmente, se hace una predicción para las características en ejecución de la nueva observación tomando el promedio de las características en ejecución de los k vecinos más cercanos. Si se denota las características en ejecución de los k vecinos más cercanos como $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_k$, entonces la predicción $\hat{\mathbf{y}}$ es:

$$\hat{\mathbf{y}} = \frac{\mathbf{y}_1 + \mathbf{y}_2 + \ldots + \mathbf{y}_k}{k}$$

Esta entonces sería la predicción de las características en ejecución para la nueva observación $(\mathbf{x}, \mathbf{z})$.

Como otro elemento que se detectó mejorable, es la capacidad de generalización del modelo, ya que solamente se probó una sola tarea con distintas configuraciones y distintas variantes de agrupación de características. La idea para trabajos futuros radica en hacer el modelo más robusto, probando entrenar los modelos con otras tareas, dispositivos y configuraciones.

Apéndice A

Métricas de evaluación de los regresores para cada variante

En este apéndice se pueden visualizar las gráficas correspondientes a las métricas mae, mse y R^2 para las variantes CUDA con OpenMP recursivo, CUDA recursivo, OpenMP no recursivo, OpenMP recursivo y Secuencial.

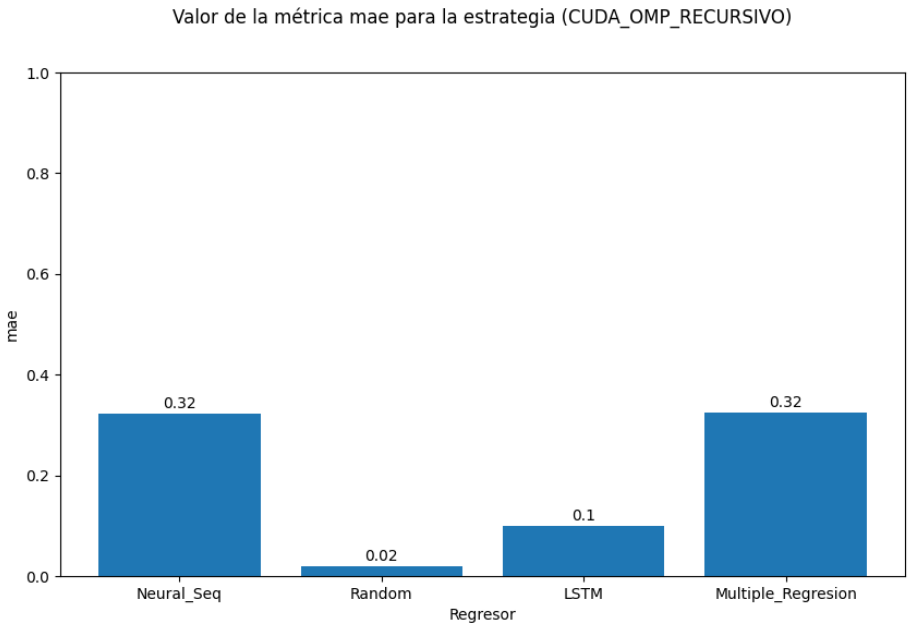

Figura A.1: Métrica de evaluación mae para el conjunto de regresores utilizados, en la estrategia CUDA con OpenMP recursivo

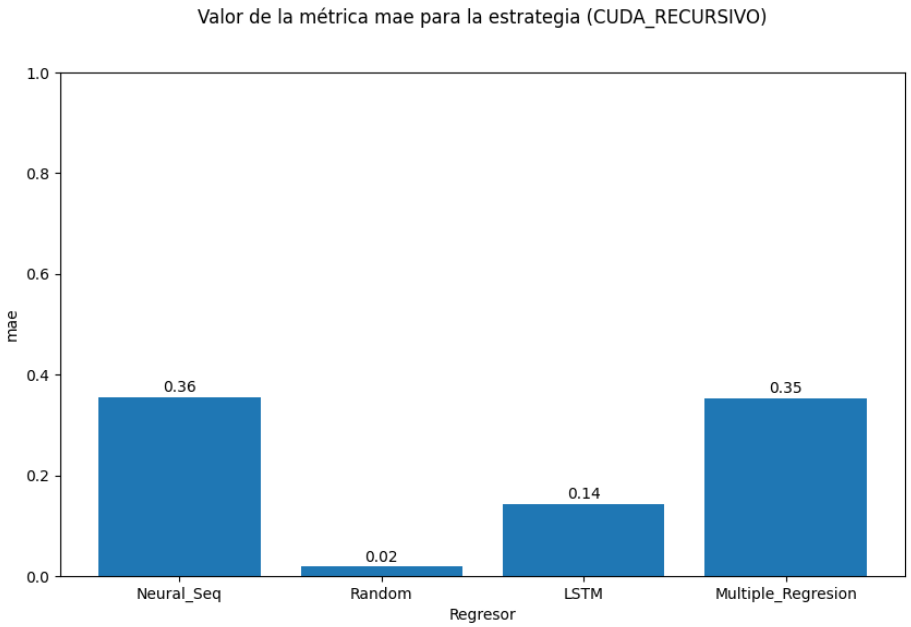

Figura A.2: Métrica de evaluación mae para el conjunto de regresores utilizados, en la estrategia CUDA recursivo

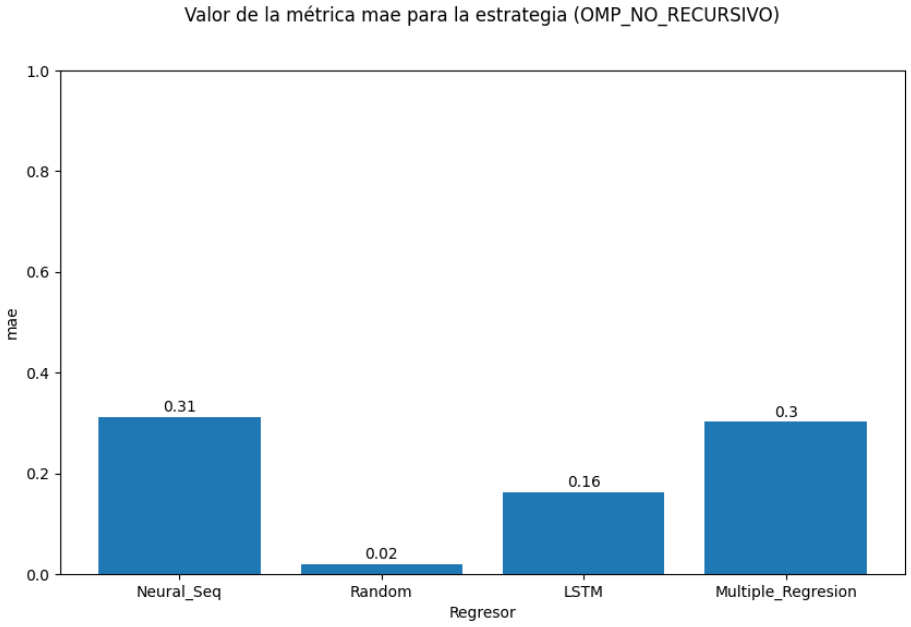

Figura A.3: Métrica de evaluación mae para el conjunto de regresores utilizados, en la estrategia OpenMP no recursivo

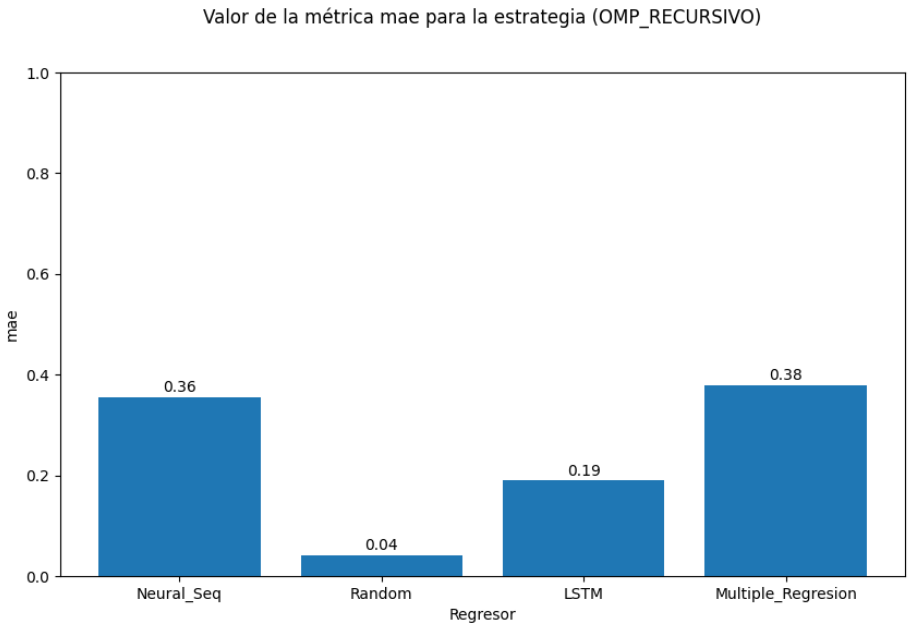

Figura A.4: Métrica de evaluación mae para el conjunto de regresores utilizados, en la estrategia OpenMP recursivo

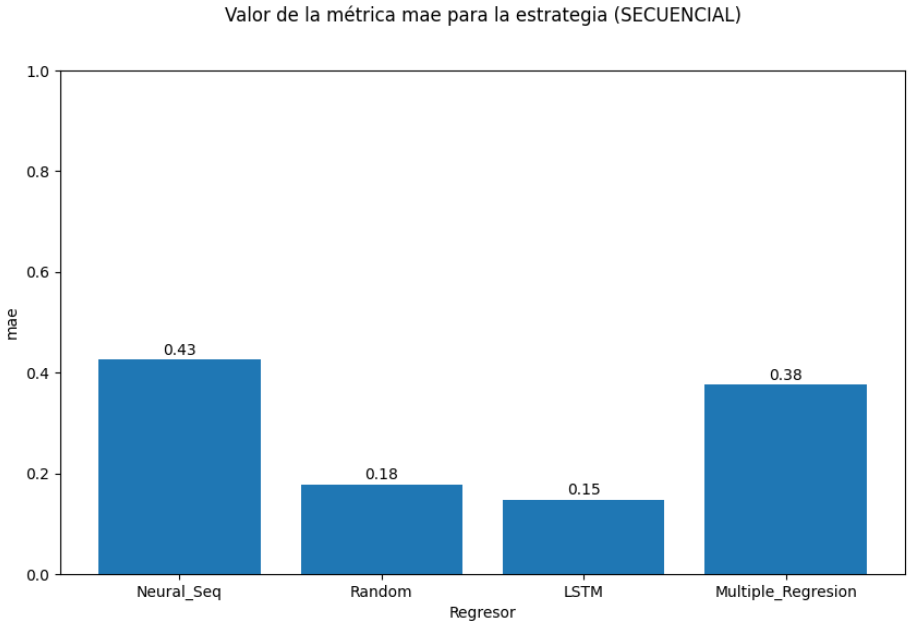

Figura A.5: Métrica de evaluación mae para el conjunto de regresores utilizados, en la estrategia Secuencial

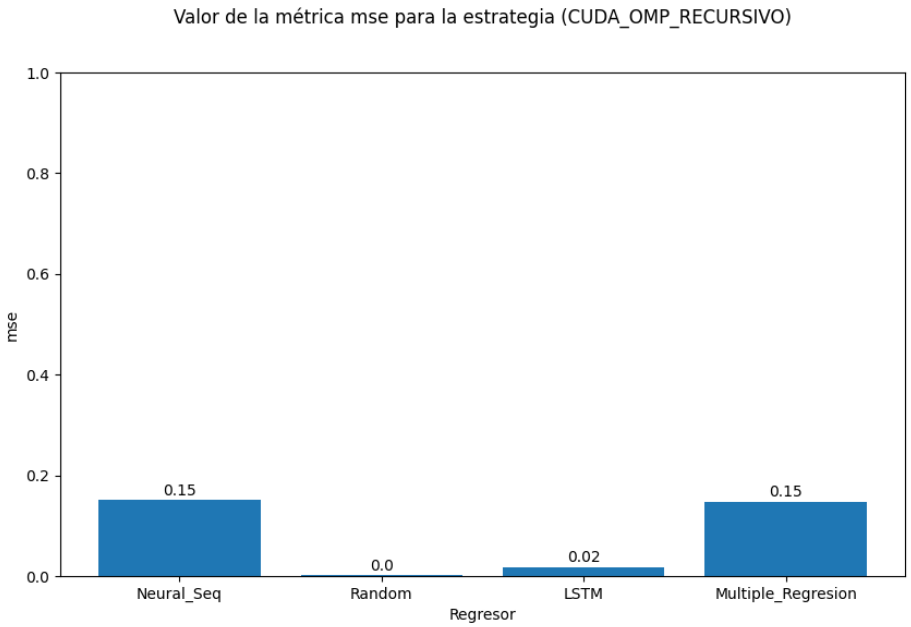

Figura A.6: Métrica de evaluación mse para el conjunto de regresores utilizados, en la estrategia CUDA con OpenMP recursivo

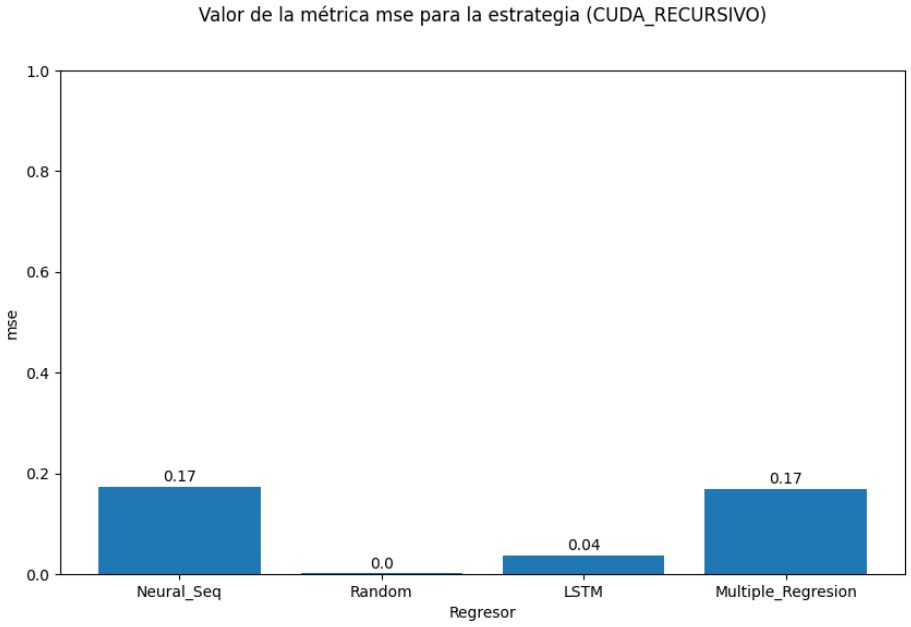

Figura A.7: Métrica de evaluación mse para el conjunto de regresores utilizados, en la estrategia CUDA recursivo

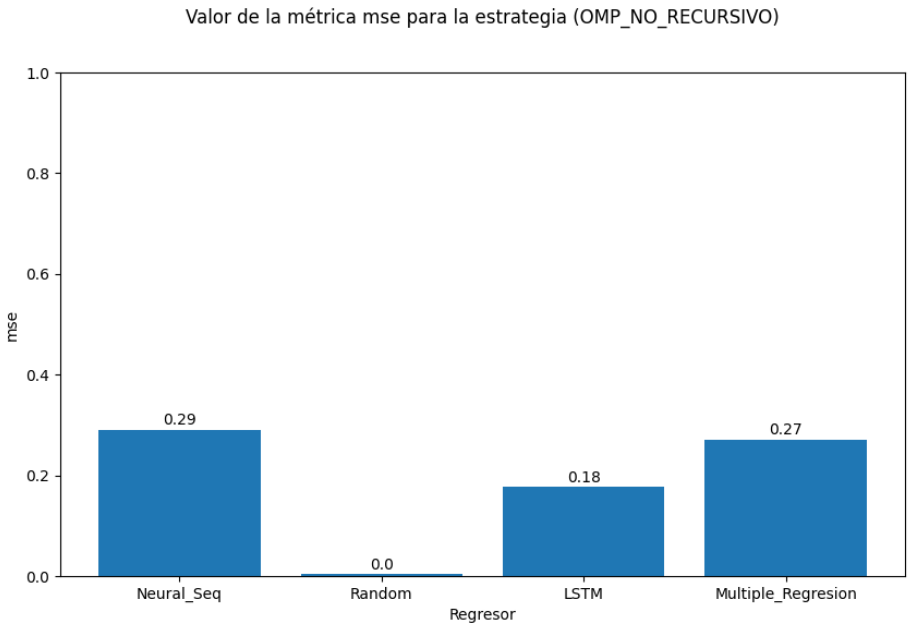

Figura A.8: Métrica de evaluación mse para el conjunto de regresores utilizados, en la estrategia OpenMP no recursivo

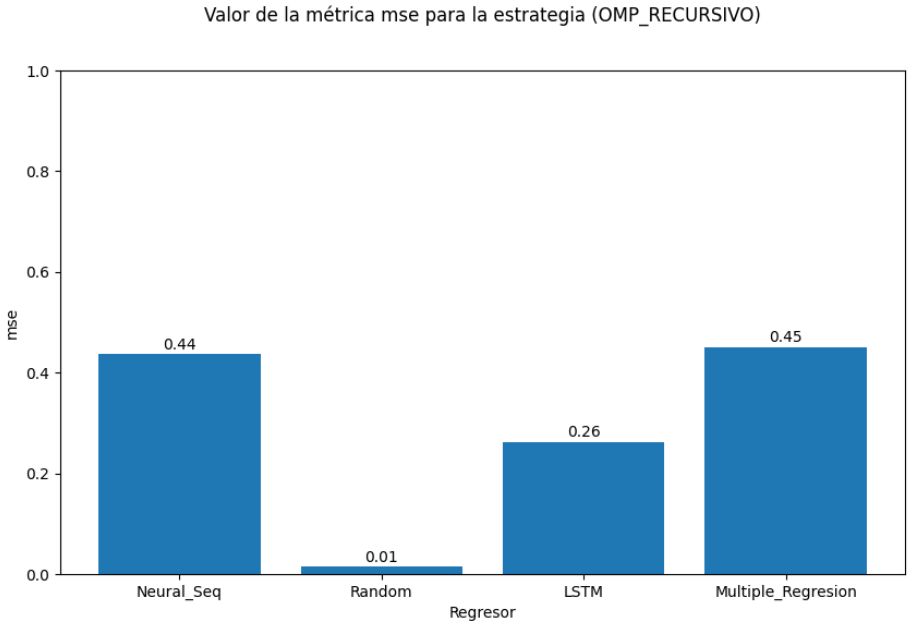

Figura A.9: Métrica de evaluación mse para el conjunto de regresores utilizados, en la estrategia OpenMP recursivo

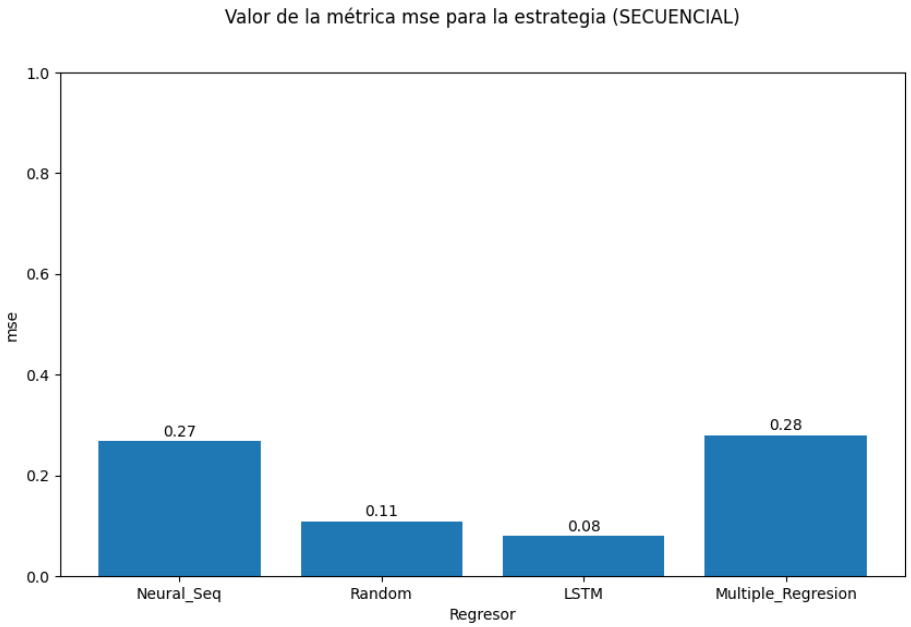

Figura A.10: Métrica de evaluación mse para el conjunto de regresores utilizados, en la estrategia Secuencial

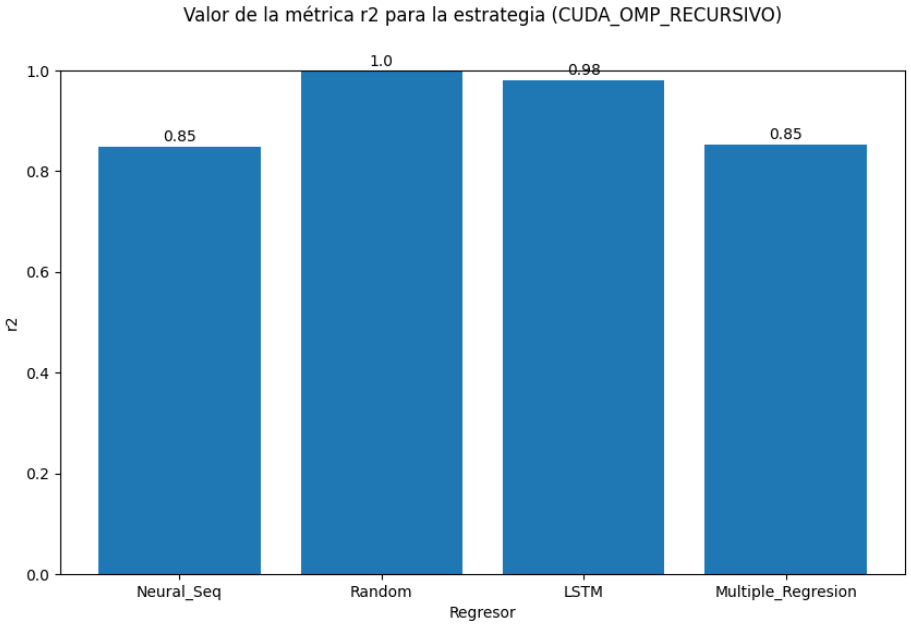

Figura A.11: Métrica de evaluación R^2 para el conjunto de regresores utilizados, en la estrategia CUDA con OpenMP recursivo

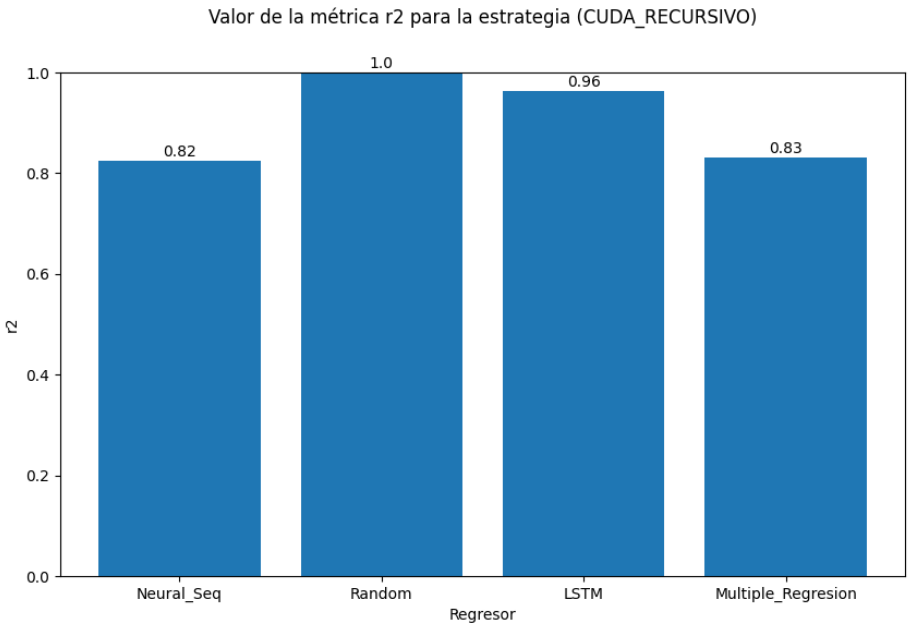

Figura A.12: Métrica de evaluación R^2 para el conjunto de regresores utilizados, en la estrategia CUDA recursivo

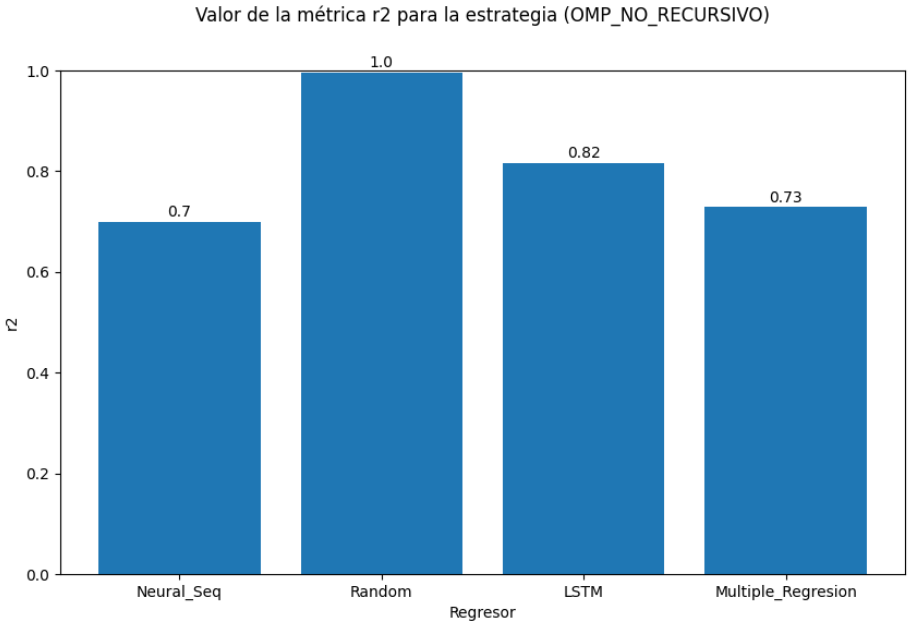

Figura A.13: Métrica de evaluación R^2 para el conjunto de regresores utilizados, en la estrategia OpenMP no recursivo

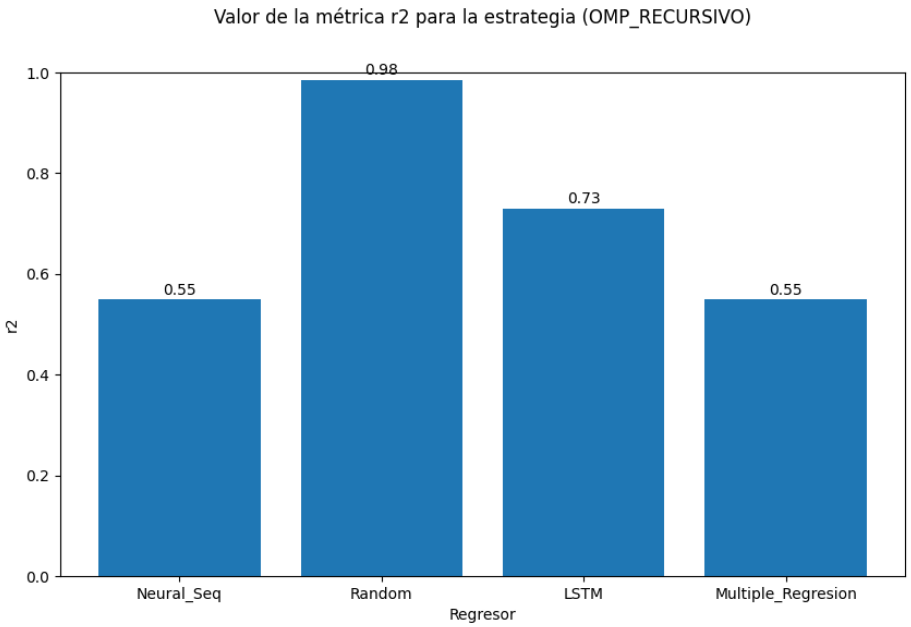

Figura A.14: Métrica de evaluación R^2 para el conjunto de regresores utilizados, en la estrategia OpenMP recursivo

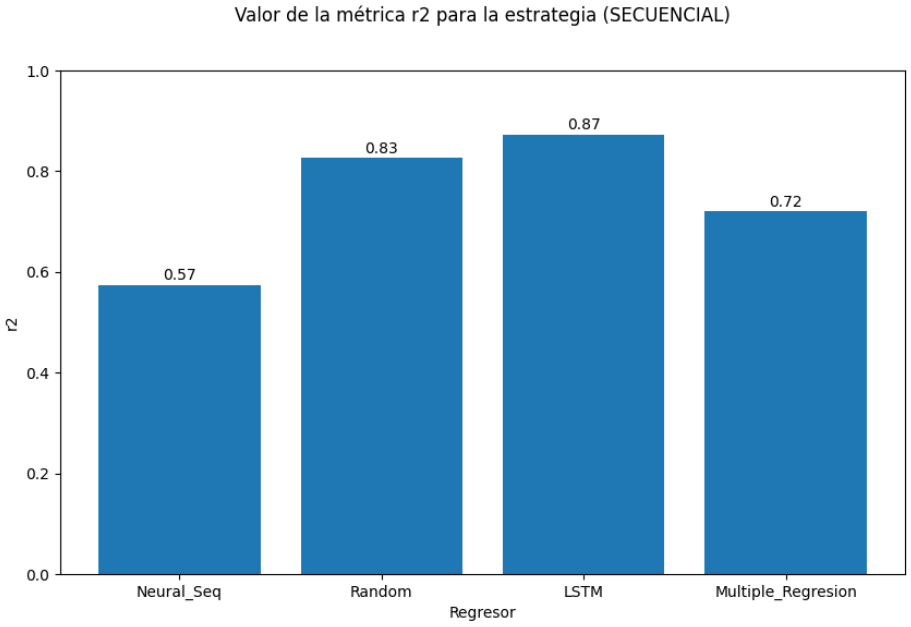

Figura A.15: Métrica de evaluación R^2 para el conjunto de regresores utilizados, en la estrategia Secuencial

Apéndice B

Gráficas de dispersión de la relación entre los valores predichos y los valores reales

En este apéndice se exponen las gráficas de dispersión de la relación entre los valores predichos y los valores reales para los modelos de Red Neuronal Secuencial y Regresión Lineal Múltiple en cada variante.

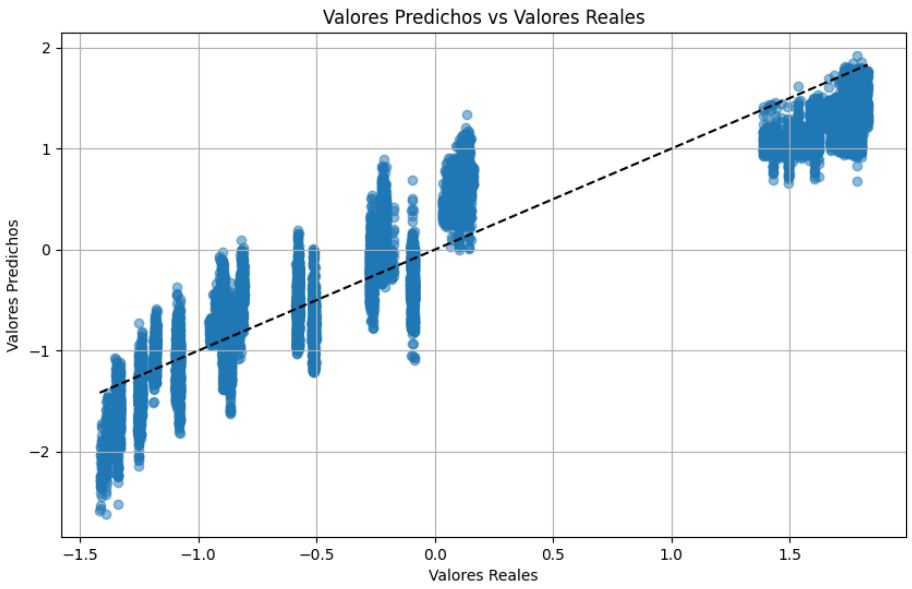

Figura B.1: Relación entre los valores predichos y reales para Regresión Lineal Múltiple en la variante CUDA con OpenMP recursivo

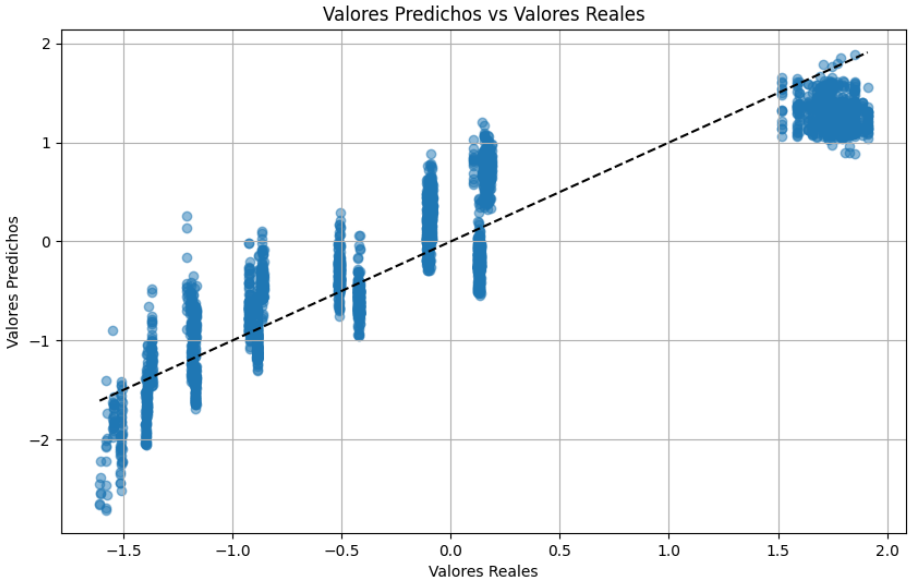

Figura B.2: Relación entre los valores predichos y reales para Regresión Lineal Múltiple en la variante CUDA recursivo

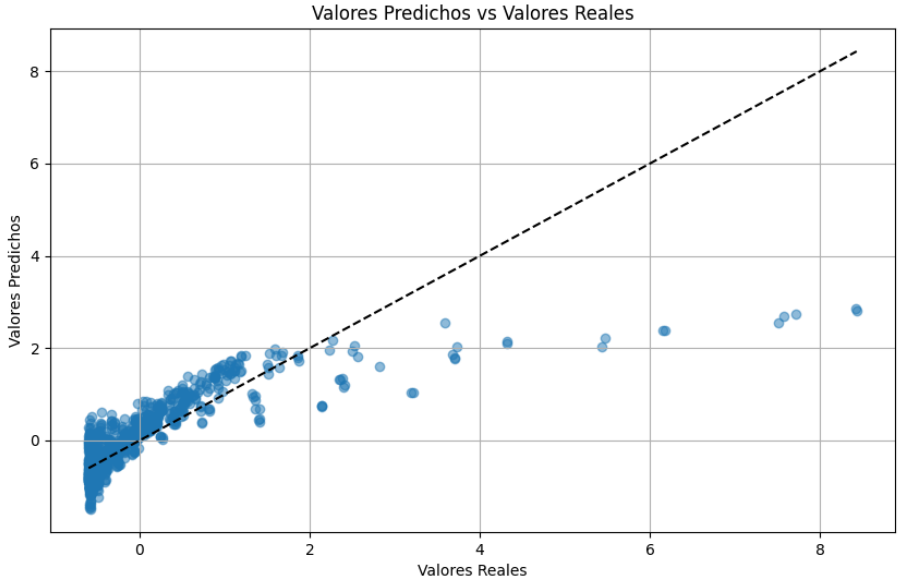

Figura B.3: Relación entre los valores predichos y reales para Regresión Lineal Múltiple en la variante OpenMP no recursivo

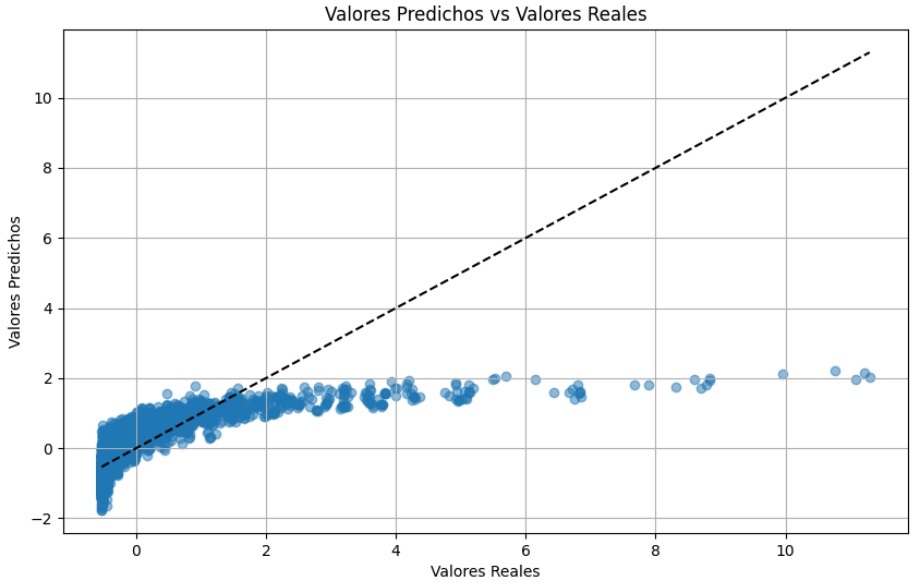

Figura B.4: Relación entre los valores predichos y reales para Regresión Lineal Múltiple en la variante OpenMP recursivo

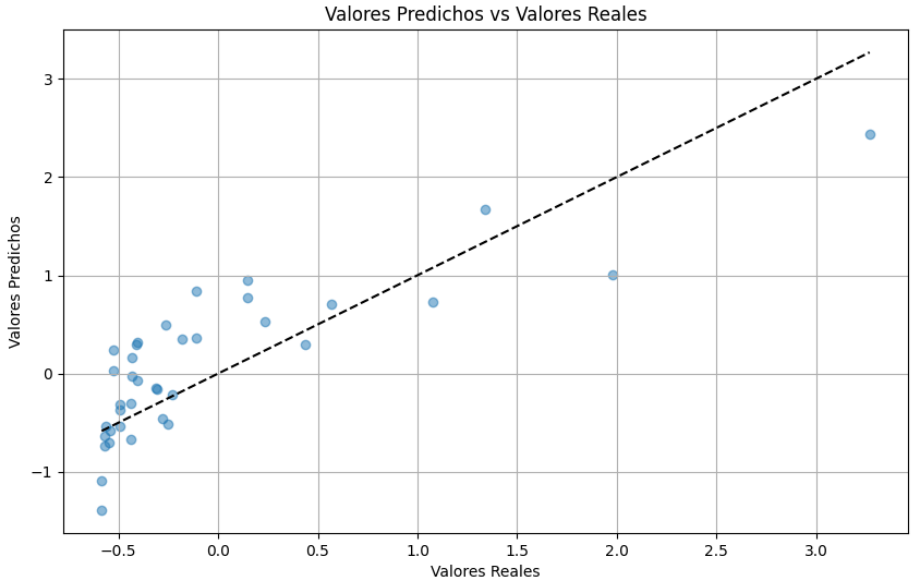

Figura B.5: Relación entre los valores predichos y reales para Regresión Lineal Múltiple en la variante secuencial

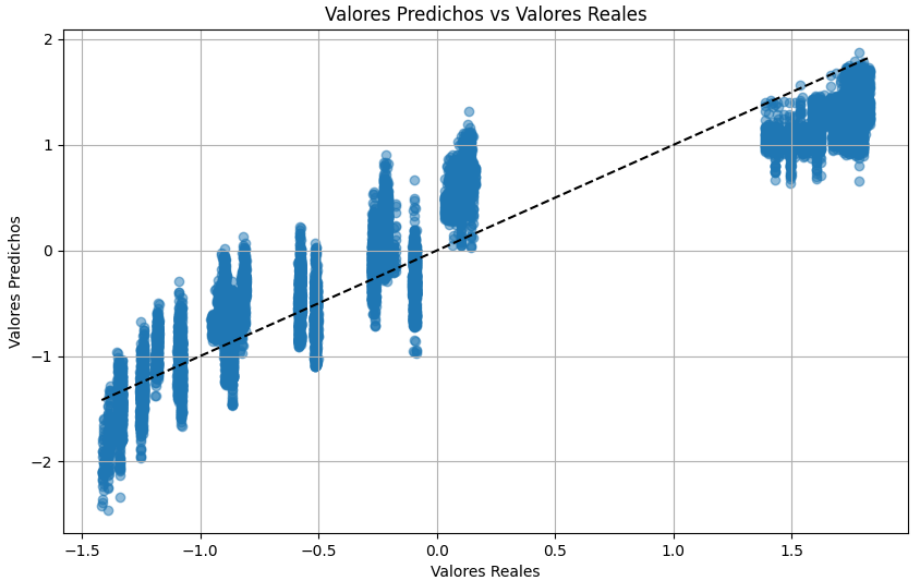

Figura B.6: Relación entre los valores predichos y reales para Red Neuronal Secuencial en la variante CUDA con OpenMP recursivo

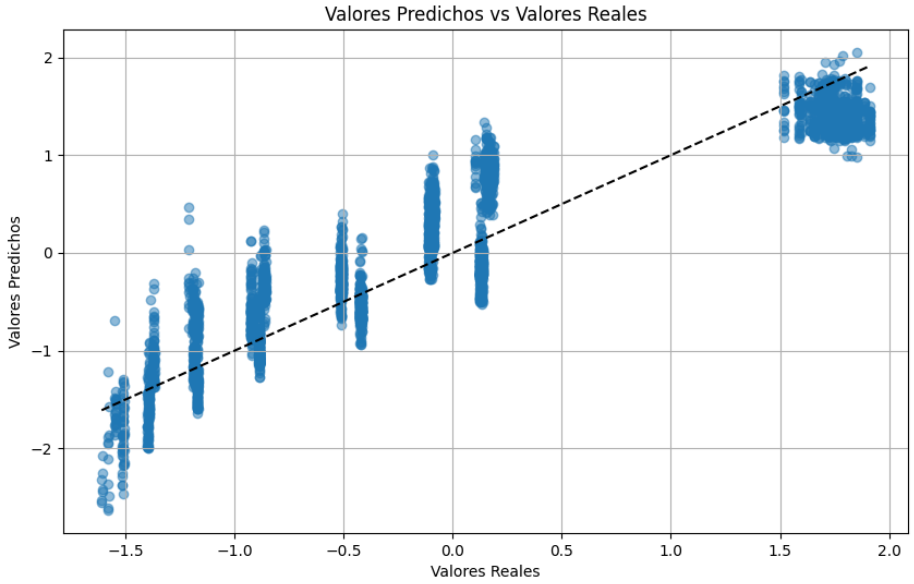

Figura B.7: Relación entre los valores predichos y reales para Red Neuronal Secuencial en la variante CUDA recursivo

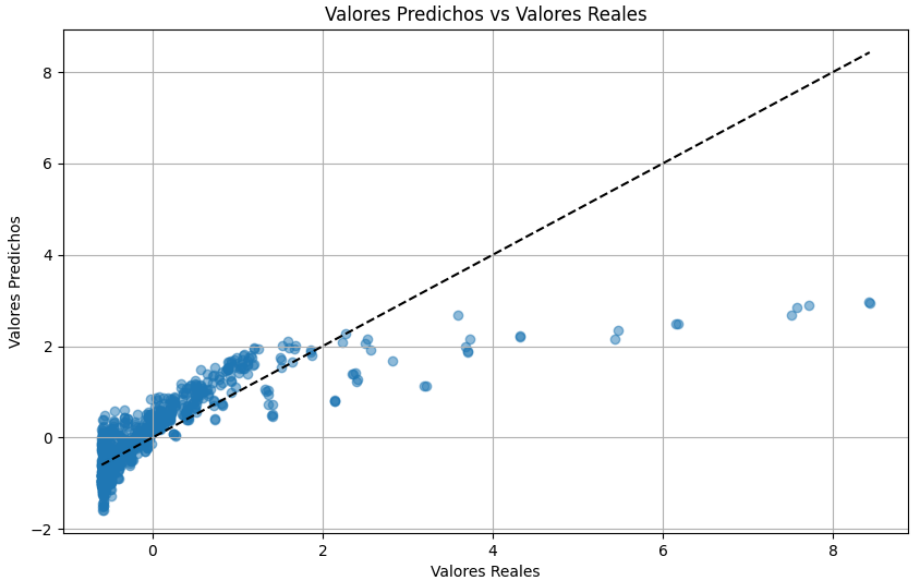

Figura B.8: Relación entre los valores predichos y reales para Red Neuronal Secuencial en la variante OpenMP no recursivo

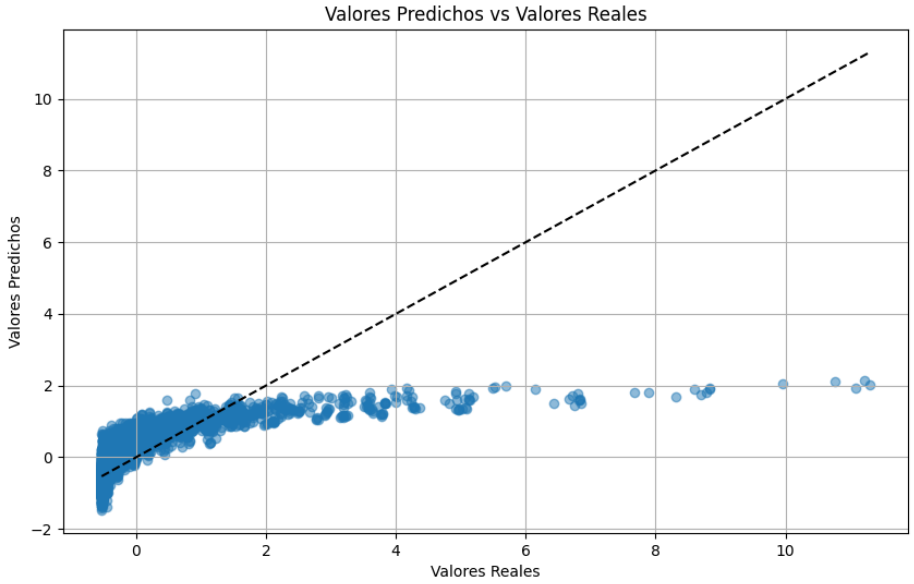

Figura B.9: Relación entre los valores predichos y reales para Red Neuronal Secuencial en la variante OpenMP recursivo

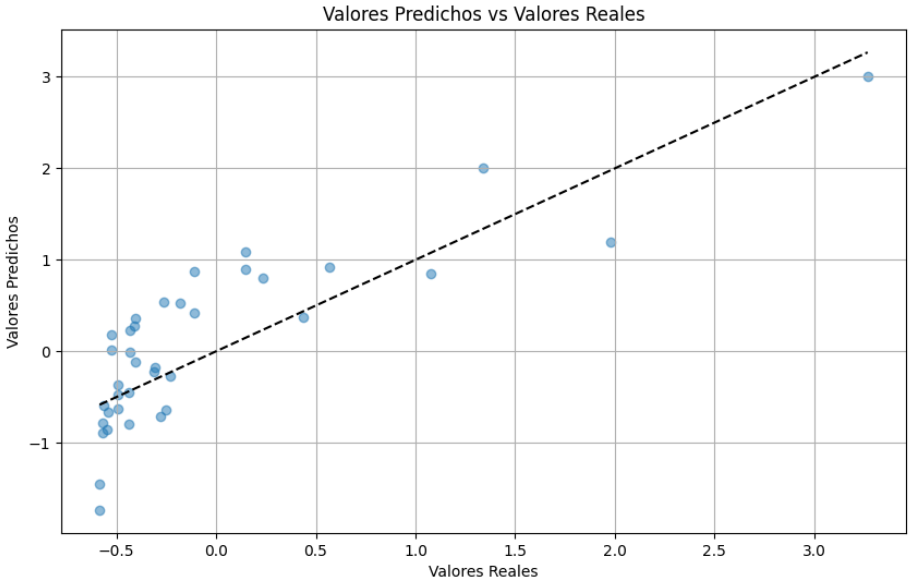

Figura B.10: Relación entre los valores predichos y reales para la Red Neuronal Secuencial en la variante secuencial

Apéndice C

Datos de ejecución de los servidores

En este apéndice se expone la tabla donde se muestran los datos de ejecución del experimento descrito en el capítulo 7.

Servidor	Tipo de memoria	Tamaño de memoria (GB)	Tipo de Disco	Cantidad de núcleos físicos	Tipo de procesador	Variante secuencial	Variante OpenMP Recursivo	Variante OpenMP no Recursivo
1	DDR4	16	HDD	6	Intel(R) Core (TM) i7-8700 CPU @ 3.20GHz	X	X	X
2	DDR4	16	HDD	6	Intel(R) Core (TM) i7-8700 CPU @ 3.20GHz	X	X	X
3	DDR4	16	HDD	6	Intel(R) Core (TM) i7-8700 CPU @ 3.20GHz		X	
4	DDR4	16	HDD	6	Intel(R) Core (TM) i7-8700 CPU @ 3.20GHz	X	X	X
5	DDR4	16	HDD	6	Intel(R) Core (TM) i7-8700 CPU @ 3.20GHz		X	X
6	DDR4	16	HDD	6	Intel(R) Core (TM) i7-8700 CPU @ 3.20GHz		X	X
7	DDR4	16	HDD	6	Intel(R) Core (TM) i7-8700 CPU @ 3.20GHz		X	X
8	DDR4	16	HDD	6	Intel(R) Core (TM) i7-8700 CPU @ 3.20GHz		X	X
9	DDR4	16	HDD	6	Intel(R) Core (TM) i7-8700 CPU @ 3.20GHz		X	X
10	DDR4	16	HDD	6	Intel(R) Core (TM) i7-8700 CPU @ 3.20GHz		X	X
11	DDR3	8	HDD	6	Intel(R) Core (TM) i7-3930K CPU @ 3.20GHz		X	X
12	DDR3	8	HDD	6	Intel(R) Core (TM) i7-3930K CPU @ 3.20GHz		X	X
13	DDR3	8	HDD	6	Intel(R) Core (TM) i7-3930K CPU @ 3.20GHz		X	X
14	DDR3	8	HDD	6	Intel(R) Core (TM) i7-3930K CPU @ 3.20GHz		X	X
15	DDR3	8	HDD	6	Intel(R) Core (TM) i7-3930K CPU @ 3.20GHz		X	X
16	DDR3	8	HDD	6	Intel(R) Core (TM) i7-3930K CPU @ 3.20GHz		X	X
17	DDR3	8	HDD	6	Intel(R) Core (TM) i7-3930K CPU @ 3.20GHz		X	X
18	DDR3	8	HDD	6	Intel(R) Core (TM) i7-4930K CPU @ 3.40GHz		X	X
19	DDR3	8	HDD	6	Intel(R) Core (TM) i7-4930K CPU @ 3.40GHz		X	X
20	DDR3	8	HDD	6	Intel(R) Core (TM) i7-4930K CPU @ 3.40GHz		X	X
21	DDR3	8	HDD	6	Intel(R) Core (TM) i7-4930K CPU @ 3.40GHz		X	X

Figura C.1: Datos de los servidores donde se ejecutaron las distintas variantes del algoritmo de Strassen

22	DDR3	8	HDD	6	Intel(R) Core (TM) i7-4930K CPU @ 3.40GHz		X	
23	DDR3	8	HDD	6	Intel(R) Core (TM) i7-4930K CPU @ 3.40GHz		X	
24	DDR3	8	HDD	6	Intel(R) Core(TM) i7-4930K CPU @ 3.40GHz		X	
25	DDR3	8	HDD	6	Intel(R) Core(TM) i7-4930K CPU @ 3.40GHz		X	
29	DDR3	8	HDD	6	Intel(R) Core(TM) i7-3970X CPU @ 3.50GHz		X	
30	DDR3	8	HDD	6	Intel(R) Core(TM) i7-3930K CPU @ 3.20GHz		X	
31	DDR4	16	HDD	6	Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz	X	X	X

Figura C.2: Datos de los servidores donde se ejecutaron las distintas variantes del algoritmo de Strassen

Índice de figuras

Índice de tablas

Bibliografía

Bibliografía

A. Hema Singh, B. R. M. (2012a). Study on functional analysis and comparison of ddrx sdram series. *International Journal of Advanced Technology Engineering Research (IJATER)*, 2:19–22.

A. Hema Singh, B. R. M. (2012b). Table 1- ddr3 feature comparison. *International Journal of Advanced Technology Engineering Research (IJATER)*, 2:19–22.

Abdollahi, M. and Rajabi, A. (2019). Static scheduling algorithm based on hybrid genetic approach for precedence-constrained tasks on heterogeneous computing systems. *Journal of Computational Science*, 34:101003.

Agarwal, A., Colak, S., Jacob, V., and Pirkul, H. (2006). Heuristics and augmented neural networks for task scheduling with non-identical machines. *European Journal of Operational Research*, 175:296–317.

Agarwal, S. (2013). Data mining: Data mining concepts and techniques. pages 203–207.

Aggarwal, C. C. (2018). *Neural Networks and Deep Learning: A Textbook*. Springer Publishing Company, Incorporated, 1st edition.

Ahmad, I. and Yu-Kwong Kwok, Y.-K. K. (1994). A new approach to scheduling parallel programs using task duplication. In *1994 Internatonal Conference on Parallel Processing Vol. 2*, volume 2, pages 47–51.

Al-Khateeb, H., Benkhlifa, E., and Bounceur, A. (2018). An overview of task scheduling in cloud computing: Concepts and challenges. *Journal of Network and Computer Applications*, 113.

Ali, S., Siegel, H., Maheswaran, M., and Hensgen, D. (2000). Task execution time modeling for heterogeneous computing systems. pages 185–199.

Alpaydin, E. (2010). *Introduction to machine learning*. MIT press.

Amalarethinam, G. and P, M. (2011). An overview of the scheduling policies and algorithms in grid computing. *International Journal of Research and Reviews In Computer Science*, 2.

Arabnejad, H. (2012). List based task scheduling algorithms on heterogeneous systems-an overview.

Arpaci-Dusseau, R. H. and Arpaci-Dusseau, A. C. (2018). *Operating Systems: Three Easy Pieces*. CreateSpace Independent Publishing Platform, North Charleston, SC, USA.

Bang, L., Aydin, A., and Bultan, T. (2015). Automatically computing path complexity of programs. pages 61–72.

Beloglazov, A., Abawajy, J., and Buyya, R. (2012). Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Generation Computer Systems*, 28(5):755–768.

Bhatti, M. K., Oz, I., Amin, S., Mushtaq, M., Farooq, U., Popov, K., and Brorsson, M. (2018). Locality-aware task scheduling for homogeneous parallel computing systems. *Computing*, 100(6):557–595.

Bielecki, J. and Śmiałek, M. (2022). Estimation of execution time for computing tasks. *Cluster Computing*.

Bittencourt, L. F., Goldman, A., Madeira, E. R., da Fonseca, N. L., and Sakellariou, R. (2018). Scheduling in distributed systems: A cloud computing perspective. *Computer Science Review*, 30:31–54.

Błaszczyk, P. and Proficz, J. (2017). Static scheduling algorithms for energy-aware real-time multiprocessor systems: A survey. *Journal of Systems Architecture*, 77:3–17.

Blumzon, C. and Pănescu, A.-T. (2019). *Data Storage*, volume 257.

Bohn, C. A. and Lamont, G. B. (2002). Load balancing for heterogeneous clusters of pcs. *Future Generation Computer Systems*, 18(3):389–400. Cluster Computing.

Buyya, R., Broberg, J., and Goscinski, A. (2013). *Mastering Cloud Computing*. Morgan Kaufmann.

Cabello Sánchez, J. U. (febrero de 2017). *XSCALA: a framework for supporting parallel task programming on hybrid heterogeneous computing systems = XSCALA: un marco de desarrollo para soportar programación paralela por tareas en sistemas de cómputo híbridos heterogéneos*. PhD thesis, CINVESTAV.

Cai, X. and Chen, H. (2019). A survey on machine learning for resource management in cloud computing. *Journal of Parallel and Distributed Computing*, 127:77–91.

Carpenter, P., Utz, U.-H., Narasimhamurthy, S., and Suarez, E. (2022). Heterogeneous High Performance Computing. Technical Report 6, ETP4HPC.

CHAPIN, S. J. (1996). Distributed and multiprocessor scheduling. *ACM Computing Surveys*, 28.

Chen, B. M., Lee, T. H., Peng, K., and Venkataramanan, V. (2010). *Hard Disk Drive Servo Systems*. Springer Publishing Company, Incorporated, 2nd edition.

Corke, G. (2021). Intel core vs amd ryzen for cad, bim & beyond. https://aecmag.com/workstations/11th-gen-intel-core-vs-amd-ryzen-5000-for-cad-bim-beyond/. visited on 23/06/2023.

Cormen, T. H. (2013). *Algorithms Unlocked*. The MIT Press.

Coulouris, G., Dollimore, J., Kindberg, T., and Blair, G. (2011a). *Distributed Systems: Concepts and Design*. Addison-Wesley Publishing Company, USA, 5th edition.

Coulouris, G., Dollimore, J., Kindberg, T., and Blair, G. (2011b). *Distributed Systems: Concepts and Design*. Pearson Education.

Dastjerdi, A. V., Gupta, H., Calheiros, R. N., Ghosh, S. K., and Buyya, R. (2018). *Fog Computing: Principles, Architectures, and Applications*. Springer.

Dongarra, J., Beckman, P., Aerts, P., and et al. (2011). The international exascale software project roadmap. *International Journal of High Performance Computing Applications*, 25(1):3–60.

Dongarra, J., Jeannot, E., Saule, E., and Shi, Z. (2007). Bi-objective scheduling algorithms for optimizing makespan and reliability on heterogeneous systems. *Annual ACM Symposium on Parallelism in Algorithms and Architectures*.

Dongarra, J. and Lastovetsky, A. L. (2009). *High Performance Heterogeneous Computing*. Wiley-Interscience, USA.

Dr. Priyankasharma, M. (2021a). Comparative analysis study on ssd, hdd, and sshd. *Turkish Journal of Computer and Mathematics Education (TURCOMAT)*, 12:3635–3641.

Dr. Priyankasharma, M. (2021b). Figure. 1 solid state drive hardware pcb. *Turkish Journal of Computer and Mathematics Education (TURCOMAT)*, 12.

Dr. Priyankasharma, M. (2021c). Table 1. comparison table of sshd, ssd, hdd. *Turkish Journal of Computer and Mathematics Education (TURCOMAT)*, 12.

Drozdowski, M. (2009). *Scheduling for Parallel Processing*. Computer Communications and Networks. Springer London, 1 edition.

Edwin S.Hou, N. A. and Ren, H. (1994). Dfrn: a new approach for duplication based scheduling for distributed memory multiprocessor systems. In *IEEE Trans.Parallel and Distributed Systems*, volume 5, pages 113–120.

El-Rewini, H. and Lewis, T. (1990). Scheduling parallel program tasks onto arbitrary target machines. *Journal of Parallel and Distributed Computing*, 9(2):138–153.

El-Shorbagy, M. (2016). A hybrid genetic algorithm for job shop scheduling problems. *International Journal of Advancement in Engineering, Technology and Computer Sciences (IJAETCS)*, 3:6–17.

Elahi, A. (2018). *Computer Systems: Digital Design, Fundamentals of Computer Architecture and Assembly Language*.

Fadika, Z., Dede, E., Hartog, J., and Govindaraju, M. (2012). Marla: Mapreduce for heterogeneous clusters. In *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, pages 49–56.

Fernández, R. (17/09/2022). Facturación del mercado de servidores en el mundo 2016-2021. https://es.statista.com/estadisticas/966137/facturacion-del-mercado-de-servidores-en-el-mundo/.

Flynn, M. and Luk, W. (2011). *Computer System Design: System on a Chip,*.

Freund, R. F. (1989). Optimal selection theory for superconcurrency. In *Proceedings of the 1989 ACM/IEEE Conference on Supercomputing*, Supercomputing '89, page 699–703, New York, NY, USA. Association for Computing Machinery.

Gao, Y. and Zhang, P. (2016). A survey of homogeneous and heterogeneous system architectures in high performance computing. In *2016 IEEE International Conference on Smart Cloud (SmartCloud)*, pages 170–175.

Gilda, K. (2013a). Comparative study of static task scheduling algorithms for heterogeneous systems. *International Journal on Computer Science and Engineering*, 5:166–173.

Gilda, K. (2013b). Comparative study of static task scheduling algorithms for heterogeneous systems. *International Journal on Computer Science and Engineering*, 5:166–173.

Goglin, B. (2014). Managing the topology of heterogeneous cluster nodes with hardware locality (hwloc). In *2014 International Conference on High Performance Computing Simulation (HPCS)*, pages 74–81.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley.

Goldberg, R. P. (1972). *Architectural Principles for Virtual Computer Systems*. PhD thesis, Harvard University, Cambridge, MA.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*, volume 1. MIT press.

Grasso, I., Pellegrini, S., Cosenza, B., and Fahringer, T. (2013). libwater: Heterogeneous distributed computing made easy. *Proceedings of the International Conference on Supercomputing*.

Guo, L., Zhao, S., Shen, S., and Jiang, C. (2012). Task scheduling optimization in cloud computing based on heuristic algorithm. *Journal of Networks*, 7.

Guo, Y. and Wang, L. (2012). Energy-aware scheduling for parallel applications on heterogeneous clusters. In *2012 20th Euromicro International Conference on Parallel, Distributed and Network-Based Processing*, pages 99–103. IEEE.

Gupta, M., Bhargava, L., and Sreedevi, I. (2020). Artificial neural network based task scheduling for heterogeneous systems. pages 74–79.

H. Singh, A. Y. (1996). Dfrn: a new approach for duplication based scheduling for distributed memory multiprocessor systems. In *Proc. Heterogeneous Computing Workshop*, pages 86–97.

Hager, G. and Wellein, G. (2010). *Introduction to High Performance Computing for Scientists and Engineers*. CRC Press, Inc., USA, 1st edition.

Hastie, T., Tibshirani, R., and Friedman, J. (2001). The elements of statistical learning. *Aug, Springer*, 1.

Hennessy, J. L. and Patterson, D. A. (2011). *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers, 5th edition.

Hu, Y., Li, J., and He, L. (2020). A reformed task scheduling algorithm for heterogeneous distributed systems with energy consumption constraints. *Neural Computing and Applications*, 32.

Huang, X. and Chen, X. (2013). Heterogeneous computing: Strategies and tactics for scheduling algorithms. *Journal of Information and Computational Science*, 10(13):3927–3935.

Härdle, W. K. and Simar, L. (2012). *Applied Multivariate Statistical Analysis*, pages 367–384.

Ilavarasan, E. and Thambidurai, P. (2007). Low complexity performance effective task scheduling algorithm for heterogeneous computing environments. *Journal of Computer Science*, 3(2):94–103.

Ilavarasan, E., Thambidurai, P., and Mahilmannan, R. (2005). High performance task scheduling algorithm for heterogeneous computing system. In Hobbs, M., Goscinski, A. M., and Zhou, W., editors, *Distributed and Parallel Computing*, pages 193–203, Berlin, Heidelberg. Springer Berlin Heidelberg.

Iverson, M., Ozguner, F., and Potter, L. (1999). Statistical prediction of task execution times through analytic benchmarking for scheduling in a heterogeneous environment. *IEEE Transactions on Computers*, 48(12):1374–1379.

Jacob, B., Brown, M., Fukui, K., Trivedi, N., et al. (2005). Introduction to grid computing. *IBM redbooks*, pages 3–6.

Jeannot, E. and Zilinskas, J. (2019). *High-Performance Computing: From Grids and Clouds to Exascale*. CRC Press.

Jeffers, J., Reinders, J., and Sodani, A. (2016). *Intel Xeon Phi Processor High Performance Programming: Knights Landing Edition 2nd Edition*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2nd edition.

Khuri, A. (2013). Introduction to linear regression analysis, fifth edition by douglas c. montgomery, elizabeth a. peck, g. geoffrey vining. *International Statistical Review*, 81.

Kiran, M., Hashim, A., Lim, M., and Yap, Y. (2009). Execution time prediction of imperative paradigm tasks for grid scheduling optimization. *International Journal of Computer Science and Network Security*, 9.

Kirk, D. (2007). Nvidia cuda software and gpu parallel computing architecture. volume 7, pages 103–104.

Kuchaki Rafsanjani, M. and Khatibi Bardsiri, A. (2012). A new heuristic approach for scheduling independent tasks on heterogeneous computing systems. *International Journal of Machine Learning and Computing*, 2:371–376.

Kuhn, M. and Johnson, K. (2019). *Feature Engineering and Selection*.

Lavaei, J., Namin, A. S., and Goudarzi, M. (2020). Online optimization of heterogeneous datacenters with resource-efficient workloads. *IEEE Transactions on Cloud Computing*, 8(4):1346–1359.

Lavaei, J., Noghabi, B., Chen, Q., and Xue, G. (2018). Online optimization of heterogeneous datacenters with resource-efficient workloads. *IEEE Transactions on Cloud Computing*, 8(4):1346–1359.

Leung, J., Agrawal, D. P., and Rao, S. (2000). Scheduling in distributed computing systems: issues and approaches. *IEEE Transactions on Parallel and Distributed Systems*, 11(8):726–738.

Lewis, T. and Kruatrachue, B. (1988). Grain size determination for parallel processing. *IEEE Software*, 5(01):23–32.

Li, Y., Ma, J., Xie, Z., Hu, Z., Shen, X., and Zhang, K. (2023). A scheduling method for heterogeneous signal processing platforms based on quantum genetic algorithm. *Applied Sciences*, 13(7).

Lin, Z., Li, C., Tian, L., and Zhang, B. (2022). A scheduling algorithm based on reinforcement learning for heterogeneous environments. *Applied Soft Computing*, 130:109707.

Liu, F. T., Ting, K., and Zhou, Z.-H. (2009). Isolation forest. pages 413 – 422.

Liu, J., Li, J., Li, D., Qian, D., and Zhan, D. (2019). Task scheduling algorithm for heterogeneous computing based on multi-objective genetic algorithm and dynamic priority strategy. *Journal of Parallel and Distributed Computing*, 124.

Liu, Y., Wang, Y., and Zhang, J. (2012). New machine learning algorithm: Random forest. volume 7473, pages 246–252.

Loh, W.-Y. (2011). Classification and regression trees. *WIREs Data Mining and Knowledge Discovery*, 1(1):14–23.

Lopes, R. (2011). *Kolmogorov-Smirnov Test*, pages 718–720.

Mahmoud, H. (2019). Parametric versus semi and nonparametric regression models.

Mammone, A., Turchi, M., and Cristianini, N. (2009). Support vector machines. *WIREs Computational Statistics*, 1(3):283–289.

MarketsandMarkets (2018). High-performance computing market by component (solutions [servers, storage, networking devices, software], and services], deployment type, organization size, server price band, application area, and region - global forecast to 2023. https://www.marketsandmarkets.com/Market-Reports/high-performance-computing-market-631.html. Accessed: March 2023.

Mattson, T. (2001). An introduction to openmp. pages 3–3.

Moore, D., McCabe, G., and Craig, B. (2012). *Introduction to the practice of statistics*.

Nelli, F. (2018). *Python Data Analytics: With Pandas, NumPy, and Matplotlib*.

Nemhauser, G. (1967). Introduction to dynamic programming.

Nemirovsky, D., Arkose, T., Markovic, N., Nemirovsky, M., Unsal, O., and Cristal, A. (2017). A machine learning approach for performance prediction and scheduling on heterogeneous cpus. pages 121–128.

Nielsen, F. (2016). *Introduction to MPI: The Message Passing Interface*, pages 21–62.

Nikoletseas, S. E. and Rolim, J. (2013). *Handbook of Research on Heterogeneous Next-Generation Networking: Innovations and Platforms*. IGI Global.

Null, L. and Lobur, J. (2006a). *The Essentials of Computer Organization And Architecture*. Jones and Bartlett Publishers, Inc., USA.

Null, L. and Lobur, J. (2006b). *FIGURE 6.1 The Memory Hierarchy*. Jones and Bartlett Publishers, Inc., USA.

Null. Linda, L. J. (2018). *Four Instructions Going Through a Six Stage Pipeline*. Jones Bartlett Learning.

Oki, Y., Mikami, H., Nishida, H., Umeda, D., Kimura, K., and Kasahara, H. (2021). *Performance of Static and Dynamic Task Scheduling for Real-Time Engine Control System on Embedded Multicore Processor.*, volume 11998 of *Lecture Notes in Computer Science. 11998*. Springer International Publishing.

Olive, D. (2017). *Linear regression*.

Palis, M. and Liou, J. (1997). A comparison of general approaches to multiprocessor scheduling. In *Parallel Processing Symposium, International*, page 152, Los Alamitos, CA, USA. IEEE Computer Society.

Paneerselvam, S., Raajan, P., and Sai, S. (2010). Genetic algorithm and particle swarm optimization approaches to solve combinatorial job shop scheduling problems.

Park, G.-L., Shirazi, B., and Marquis, J. (1997). Dfrn: a new approach for duplication based scheduling for distributed memory multiprocessor systems. In *Proceedings 11th International Parallel Processing Symposium*, pages 157–166.

Patterson, D. A. and Hennessy, J. L. (2013). *Computer Organization and Design: The Hardware/Software Interface*. Elsevier.

Perner, P. (2002). Data mining - concepts and techniques. *KI*, 16:77.

Pham, T.-P., Durillo, J. J., and Fahringer, T. (2020). Predicting workflow task execution time in the cloud using a two-stage machine learning approach. *IEEE Transactions on Cloud Computing*, 8(1):256–268.

Pinedo, M. (2016). Scheduling: theory, algorithms, and systems. *Springer Science & Business Media*.

Press, R. (2023). Table 1. ddr5 changes and advantages over ddr4 dimms.

Qin, X. and Jiang, H. (2005). A dynamic and reliability-driven scheduling algorithm for parallel real-time jobs executing on heterogeneous clusters. *Journal of Parallel and Distributed Computing*, 65(8):885–900.

Quinn, M. J. (2004). *Parallel Computing*. McGraw-Hill.

Rajaraman, R. (2012). *Algorithm design and applications*. Pearson Education India.

Ramaswamy, S., Rastogi, R., and Shim, K. (2000). Efficient algorithms for mining outliers from large data sets. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, SIGMOD '00, page 427–438, New York, NY, USA. Association for Computing Machinery.

Ray, U., Hazra, T., and Ray, U. (2016). Matrix multiplication using strassen's algorithm on cpu gpu.

Refaeilzadeh, P., Tang, L., and Liu, H. (2009). Cross-validation. *Encyclopedia of Database Systems*, 532–538:532–538.

Rjoub, G., Bentahar, J., Wahab, O., and Batineh, A. (2020). Deep and reinforcement learning for automated task scheduling in large-scale cloud computing systems.

Robinson, A. (2008). Introduction to nonparametric regression. kunio takezawa, john wiley & sons inc., hoboken, new jersey, 2006. no. of pages: xviii+538. price: $123.00. isbn: 978-0-471-74583-9. *Statistics in Medicine*, 27(16):3214–3215.

Roque Díaz, E. A. (2021). Análisis de proporcionalidad energética en servidores de hpc. Master's thesis, CINVESTAV.

Ryckbosch, F., Polfliet, S., and Eeckhout, L. (2011). Trends in server energy proportionality. *Computer*, 44(9):69–72.

Sakellariou, R. and Zhao, H. (2004). A hybrid heuristic for dag scheduling on heterogeneous systems.

Sarstedt, M. and Mooi, E. (2014). *Regression Analysis*, pages 193–233.

Schutten, J. (1996). List scheduling revisited. *Operations Research Letters*, 18(4):167–170.

Shanthini, J. and Shankarkumar, K. R. (2012). Article: Anatomy study of execution time predictions in heterogeneous systems. *International Journal of Computer Applications*, 45(7):39–43. Full text available.

Shen. P. John, L. M. (2013). *Modern Processor Design Fundamental of Superscalar Processor*. WAVELAND, 4180 IL Route 83, Suite 101 Long Grove, IL 60047-9580.

Silberschatz, A., Galvin, P. B., and Gagne, G. (2012). *Operating System Concepts*. Wiley.

Silberschatz, A., Galvin, P. B., and Gagne, G. (2021a). *Figure 3.2 Diagram of process state,"Operating System Concepts"*, page 108. Wiley.

Silberschatz, A., Galvin, P. B., and Gagne, G. (2021b). *Figure 4.1 Single-threaded and multithreaded processes.,"Operating System Concepts"*, page 108. Wiley.

Silberschatz, A., Galvin, P. B., and Gagne, G. (2021c). *Figure 4.3 Concurrent execution on a single-core system.,"Operating System Concepts"*, page 108. Wiley.

Silberschatz, A., Galvin, P. B., and Gagne, G. (2021d). *Figure 4.4 Parallel execution on a multicore system.,"Operating System Concepts"*, page 108. Wiley.

Silberschatz, A., Galvin, P. B., and Gagne, G. (2021e). *Operating System Concepts*. Wiley.

Song, W., Chen, X., Li, Q., and Cao, Z. (2022). Flexible job shop scheduling via graph neural network and deep reinforcement learning. *IEEE Transactions on Industrial Informatics*, pages 1–11.

Sotiriades, E., Petraki, E., Kartsakli, E., Souravlias, D., and Bouganis, C.-S. (2015). A survey of task scheduling in multicore and accelerator-based systems. *ACM Computing Surveys*, 48(1).

Stakem, P. (2012). Architecture programming of the intel x86 architecture.

Sterling, T., Anderson, M., Brodowicz, M., and et al. (2018). *High Performance Computing: Modern Systems and Practices*. Elsevier.

Sukhpal Singh, I. C. (2016). A survey on resource scheduling in cloud computing: Issues and challenges. *Journal of Grid Computing*, 14:1–18.

Taktak-Meziou, M., Chemori, A., Ghommam, J., and Derbel, N. (2015a). Fig. 1: View of the main components of a typical hdd. *Transactions on Systems, Signals Devices*, 10:1–23.

Taktak-Meziou, M., Chemori, A., Ghommam, J., and Derbel, N. (2015b). Fig. 2: Illustration of the main operating functions of a hdd servo-system. *Transactions on Systems, Signals Devices*, 10:1–23.

Tanenbaum, A. S. and Bos, H. (2014a). *Location of type 1 and type 2 hypervisors.* Prentice Hall Press, USA, 4th edition.

Tanenbaum, A. S. and Bos, H. (2014b). *Modern Operating Systems*. Prentice Hall Press, USA, 4th edition.

Tanenbaum, A. S. and Steen, M. v. (2006a). *Distributed Systems: Principles and Paradigms (2nd Edition)*. Prentice-Hall, Inc., USA.

Tanenbaum, A. S. and Steen, M. v. (2006b). *An example of a cluster computing system.* Prentice-Hall, Inc., USA.

Tanenbaum, A. S. and Steen, M. v. (2006c). *A layered architecture for grid computing systems.* Prentice-Hall, Inc., USA.

Tariq, R., Aadil, F., Malik, M. F., Ejaz, S., Khan, U., and Khan, M. (2019). Directed acyclic graph based task scheduling algorithm for heterogeneous systems: Proceedings of the 2018 intelligent systems conference (intellisys) volume 2. pages 936–947.

Taylor, P. (8/09/2022). Volume of data/information created, captured, copied, and consumed worldwide from 2010 to 2020, with forecasts from 2021 to 2025. https://www.statista.com/statistics/871513/worldwide-data-created/#:~:text=Over%20the%20next%20five%20years,to%20more%20than%20180%20zettabytes.

Technology, T. K. (2020). ¿qué está impulsando el crecimiento de los centros de datos? https://media.kingston.com/kingston/pdf/ktc-\\solutions-servers-data-centers-what-is-driving-the-growth-of-data-\\centers-infographic-latam.pdf.

Teraiya, J., Shah, A., Pant, M., Sharma, T., Verma, O., Singla, R., and Sikander, A. (2020). Analysis of dynamic and static scheduling algorithms in soft real-time system with its implementation. Marwadi University, Gujarat, Rajkot, India.

Topcuoglu, H., Hariri, S., and Wu, M.-Y. (2002a). Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, 13(3):260–274.

Topcuoglu, H., Hariri, S., and Wu, M.-Y. (2002b). Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, 13(3):260–274.

Topcuoglu, H., Hariri, S., and Wu, M.-Y. (2002c). Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, 13(3):263.

Ullman, J. (1975). Np-complete scheduling problems. *Journal of Computer and System Sciences*, 10(3):384–393.

Van Houdt, G., Mosquera, C., and Nápoles, G. (2020). A review on the long short-term memory model. *Artificial Intelligence Review*, 53.

Wu, M.-Y. and Gajski, D. (1990). Hypertool: a programming aid for message-passing systems. *IEEE Transactions on Parallel and Distributed Systems*, 1(3):330–343.

Xu, Y., Li, K., Khac, T. T., and Qiu, M. (2012). A multiple priority queueing genetic algorithm for task scheduling on heterogeneous computing systems. In *2012 IEEE 14th International Conference on High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems*, pages 639–646.

Yang, S., Li, X., Li, X., and Liu, X. (2019). Quantifying energy and performance efficiency for homogeneous and heterogeneous hpc systems. *Sustainable Computing: Informatics and Systems*, 22:54–65.

Yang, T. and Gerasoulis, A. (1994). Dsc: scheduling parallel tasks on an unbounded number of processors. *IEEE Transactions on Parallel and Distributed Systems*, 5(9):951–967.

Zaman, S. K., Maqsood, T., Ali, M., Bilal, K., Madani, S., and Khan, A. u. R. (2019). A load balanced task scheduling heuristic for large-scale computing systems. *Computer Systems Science and Engineering*, 1:1–12.

Zhang, L., Li, K., Li, C., and Li, K. (2016). Bi-objective workflow scheduling of the energy consumption and reliability in heterogeneous computing systems. *Information Sciences*, 379.

Zhang, X. and Zhong, Y. (2014). An effective hybrid genetic algorithm for heterogeneous multi-objective task scheduling. In *2014 26th International Conference on Microelectronics*, pages 1–4. IEEE.