Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

Unidad Zacatenco

Departamento de Computación

Calendarización para disminuir el tiempo de ejecución de integrales multidimensionales en entornos heterogéneos.

Tesis que presenta

Irene Elizabeth López Mares

para obtener el Grado de

Maestro en Ciencias en Computación

Directores de la Tesis

Dr. Amilcar Meneses Viveros
Dr. José Guadalupe Rodríguez García

Ciudad de México **Diciembre 2025**

Resumen

Existen diversos problemas que requieren del cálculo de integrales multidimensionales; sin embargo, la resolución numérica de estas tiene una alta complejidad. Además, el error numérico aumenta con el número de dimensiones.

Para disminuir la complejidad y el tiempo de ejecución se han desarrollado bibliotecas de integración con base en las cuadraturas Gaussianas y se han paralelizado. Sin embargo, el error numérico disminuye al aumentar el número de puntos de integración, lo cual consume demasiado tiempo computacional.

Una solución para reducir el error numérico, al disminuir el número de puntos, es aplicar la extrapolación de Romberg y aprovechar los entornos de ejecución heterogénea que hay en casi todos los equipos de cómputo.

En este trabajo de tesis se implementó la extrapolación de Romberg apoyada de un calendarizador para aprovechar las plataformas heterogéneas.

Los resultados obtenidos permiten concluir que el sistema de calendarización propuesto es funcional y adaptable a distintas plataformas, integrando de manera eficaz las técnicas de extrapolación y calendarización. La extrapolación de Romberg contribuyó a una reducción significativa en el tiempo de ejecución; sin embargo, el calendarizador no logró el rendimiento esperado, lo que sugiere la necesidad de realizar ajustes adicionales en la estrategia de asignación de tareas.

Abstract

There are various problems that require the calculation of multidimensional integrals; however, their numerical resolution is highly complex. Furthermore, the numerical error increases with the number of dimensions.

To reduce complexity and execution time, integration libraries based on Gaussian quadratures have been developed and parallelized. However, the numerical error decreases with increasing number of integration points, which is computationally time-consuming.

One solution to reduce numerical error by decreasing the number of points is to apply Romberg extrapolation and take advantage of the heterogeneous execution environments found on almost all computing equipment.

In this thesis, Romberg extrapolation was implemented with the support of a scheduler to take advantage of heterogeneous platforms.

The results obtained allow us to conclude that the proposed scheduling system is functional and adaptable to different platforms, effectively integrating extrapolation and scheduling techniques. Romberg extrapolation contributed to a significant reduction in execution time. However, the scheduler did not achieve the expected performance, suggesting the need for further adjustments to the task allocation strategy.

Agradecimientos

Agradezco al Centro de Investigación y Estudios Avanzados (CINVESTAV), por abrirme sus puertas y en especial al Departamento de Computación por brindarme apoyo y conocimiento durante toda la maestría.

Agradezco a mis asesores de tesis, el Dr. Amílcar Meneses Viveros y el Dr. José Guadalupe, por su valiosa guía y orientación durante el desarrollo de este trabajo. Gracias por acompañarme en cada etapa del proceso, por compartir su conocimiento y por ofrecerme su apoyo constante, que fue fundamental para la realización y consolidación de esta tesis.

Agradezco a profesores del departamento, por compartir su conocimiento, su experiencia y su pasión por la investigación, los cuales fueron una guía fundamental en mi formación académica. Así como al resto del personal, gracias por su apoyo.

Agradezco a mis padres y a mi hermano, por su amor incondicional, su apoyo constante y su confianza en mí a lo largo de este camino.

Agradezco a mis amigos, por acompañarme durante este trayecto, brindándome su compañía, ánimo y comprensión en los momentos más importantes.

Y por último, agradezco a la Secretaría de Ciencia, Humanidades, Tecnología e Innovación (SECITHI) y al Programa de Becas Elisa Acuña, por el apoyo económico otorgado durante mis estudios de maestría.

Índice general

Índice de figuras

Índice de tablas

Índice de algoritmos

Capítulo 1

Introducción

Actualmente existen diversos problemas en Física, Química e Ingeniería que requieren del cálculo de integrales multidimensionales. Sin embargo, no todas las integrales tienen solución analítica, por lo que se requieren métodos numéricos para su resolución.

En las últimas décadas, se ha propuesto y desarrollado una amplia variedad de métodos numéricos (Gibbs, 1915). Muchos de ellos se basan en cuadraturas gaussianas, las cuales, al aumentar el número de dimensiones, incrementan tanto el error numérico como la complejidad computacional. No obstante, la cuadratura de Gauss-Kronrod (Genz, 1972; Patterson, 1968; Piessens y Branders, 1974) se distingue por presentar el menor número de decimales donde oscila el error, gracias a la elección estratégica de los puntos que mejora la precisión de la integral. Sin embargo, la complejidad computacional sigue siendo elevada, por lo que se ha optado por disminuir el número de puntos y complementar el proceso con técnicas de extrapolación, con el fin de obtener estimaciones más precisas del resultado sin incrementar de manera significativa el costo computacional.

Por tal motivo, en este proyecto se propone utilizar los algoritmos presentados en el artículo (Quintero-Monsebaiz y cols., 2021), los cuales calculan las integrales mediante la cuadratura de Gauss-Kronrod combinada con extrapolaciones de Romberg (Kebaier, 2005; E. H. L. Liu, 2006), con el objetivo de implementarlos en una plataforma heterogénea utilizando estrategias de calendarización que permitan reducir el tiempo computacional.

1.1 Planteamiento del problema

Como se mencionó previamente, en la evaluación de integrales multidimensionales mediante cuadraturas gaussianas, aumentar el número de dimensiones provoca un incremento en la complejidad computacional. Sin embargo, las integrales multidimensionales ofrecen resultados con mayor precisión a diferencia de otros métodos, como se puede observar en la Figura 1.1

En la columna izquierda se muestran: la suma de Riemann ($\sum$), la integral unidimensional ($\int$) y la integral multidimensional ($\iiint$). Cada una representa un método

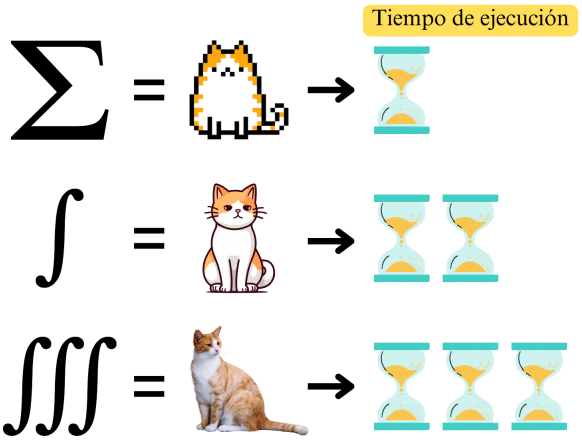


Figura 1.1: Precisión y tiempo de ejecución de los métodos de integración

.

de integración con distinto nivel de complejidad.

En la columna central, cada método está asociado a un gato, usado como metáfora del resultado obtenido:

- El gato pixelado corresponde a la suma, indicando un resultado menos preciso.
- El gato ilustrado representa la integral unidimensional, con un nivel de precisión intermedio.
- El gato realista corresponde a la integral multidimensional, simbolizando un resultado más preciso y detallado.

La idea es que entre más realista es el gato, mayor es la precisión del método asociado. Sin embargo, mayor precisión implica un mayor tiempo de ejecución requerido.

- La suma requiere poco tiempo (un reloj),
- La integral unidimensional requiere un tiempo moderado (dos relojes),
- La integral multidimensional demanda el mayor tiempo de cómputo (tres relojes).

Con el objetivo de reducir el tiempo de ejecución sin comprometer la precisión, los algoritmos descritos en (Quintero-Monsebaiz y cols., 2021) combinan la cuadratura de Gauss–Kronrod con la extrapolación de Romberg. Para evaluar su desempeño, estos algoritmos se sometieron a pruebas utilizando un conjunto de funciones de referencia del *benchmark* de integrales multidimensionales (Arumugam, Godunov, Ranjan, Terzic, y Zubair, 2013), lo cual permitió medir tanto su eficiencia computacional como su precisión.

$$f_1(x) = \left[\alpha + cos^2\left(\sum_{n=1}^{n} x^2\right)\right]^2, \tag{1.1}$$

$$f_2(x) = cos\left(\prod_{i=1}^{n} cos(2^{2i}x_i)\right), \tag{1.2}$$

$$f_3(x) = sen\left(\prod_{i=1}^{n} i \cdot arcsin(x_i^i)\right), \tag{1.3}$$

$$f_4(x) = sen\left(\prod_{i=1}^{n} arcsin(x_i)\right), \tag{1.4}$$

$$f_5(x) = \frac{1}{2\beta}\sum_{n=1}^{n} cos(\alpha x_i), \tag{1.5}$$

donde $\alpha = 0.1$ y $\beta = -0.054402111088937$. Las funciones serán calculadas en un hipercubo $[0,1]^d$, donde $d = 3, 4, 5, 6$.

La complejidad al evaluar cada función del *benchmark* en cada punto de la malla multidimensional es:

$$O(n^d), \tag{1.6}$$

donde n es el número de puntos y d es el número de dimensiones $d \geqslant 3$ (Quintero-Monsebaiz y cols., 2021).

El número de puntos evaluados en la integral está directamente relacionado con el error numérico; para que el resultado sea aceptable, dicho número debe ser lo suficientemente grande como para garantizar un error dentro de los márgenes tolerables (en función del dominio del problema puede ser $\leqslant 10^{-4}$). En (Quintero-Monsebaiz y cols., 2021) se realizaron pruebas con las funciones del *benchmark* usando hasta 56 puntos, sin embargo, como se muestra en la Tabla 1.1, el error sigue siendo demasiado alto para ser aceptable, incluso en CUHRE (Hahn, 2005), considerado uno de los algoritmos más empleados en la resolución de integrales multidimensionales. Por lo tanto, es necesario realizar pruebas con un mayor número de puntos hasta alcanzar un error numérico aceptable.

No obstante, al incrementar el número de puntos, la complejidad computacional también se eleva considerablemente, en especial para 5 y 6 dimensiones, lo que implica un alto costo en tiempo de ejecución. A pesar de lo anterior, es posible reducir el número de puntos a aproximadamente 40 mediante la implementación de extrapolaciones de Romberg. Por lo tanto, la complejidad resultante al emplear estas extrapolaciones es la siguiente:

$$O(\sum_{k=1}^{40} k^d) \approx O(40^d), \tag{1.7}$$

donde d es el número de dimensiones $\geqslant 3$ (Quintero-Monsebaiz y cols., 2021).

Además, el comportamiento al evaluar la fórmula de la extrapolación de Romberg (Ecuación 1.8) permite que los cálculos puedan realizarse en paralelo, debido a que los elementos $T_{i,0}$ se ejecutan de manera concurrente. Esta característica contribuye a la reducción significativa del tiempo de ejecución.

$$T_{i,k} = T_{i,k-1} + \frac{(T_{i,k-1} - T_{i-1,k-1})}{4^k - 1}, \tag{1.8}$$

donde $T_{i,0}$ es la evaluación de la integral para i puntos y $T_{i,k}$, para $k > 0$, es el k-ésimo ajuste (Kebaier, 2005; E. H. L. Liu, 2006).

Tabla 1.1: Error numérico relativo, σ_{rel}, CUHRE vs Kronrod. Las funciones corresponden a las Funciones 1.1 y 1.2 del *benchmark* de integrales multidimensionales. Extraído de (Quintero-Monsebaiz y cols., 2021).

	CUHRE		**Kronrod**	
Función	**Integral**	σ_{rel}	**Integral(56puntos)**	σ_{rel}
3 dimensiones				
1	20.03172186	e-06	20.03172187	e-12
2	0.930740275	e-07	0.930907702	e-04
4 dimensiones				
1	27.06839191	e-05	27.06839069	e-11
2	0.965374901	e-02	0.963093722	e-02
5 dimensiones				
1	27.63237484	e-03	27.63223859	e-10
2	0.978854949	e-02	0.979649828	e-02
6 dimensiones				
1	23.60730077	e-01	23.60655943	e-08
2	0.992661704	e-02	0.990171077	e-02

1.2 Propuesta

La hipótesis de este proyecto plantea que, a través de una estrategia de calendarización adecuada en entornos heterogéneos, se puede reducir el tiempo de ejecución de las integrales multidimensionales. Con el fin de validar esta hipótesis, se propone optimizar la ejecución de la extrapolación de Romberg seleccionando, para cada caso, el módulo de ejecución más adecuado (secuencial, CUDA, OpenMP o AVX), en función del número de dimensiones de la integral y del número de puntos empleados en la integración.

Antes de construir el calendarizador, primero es necesario realizar un conjunto de pruebas que permitan determinar cuántos puntos deben emplearse para alcanzar un error numérico aceptable. Para ello, se toma como referencia la Función 1.3 del *benchmark*, clasificada como la segunda más compleja. La justificación de esta elección se expone en la Sección 4.2.2. Se considera que, al controlar el error en esta función, también será posible lograr una precisión adecuada en el resto de las funciones menos complejas incluidas en el *benchmark*.

El calendarizador propuesto recibirá como entradas el número de dimensiones y el número de puntos de integración. Con base en datos previamente obtenidos sobre los tiempos de ejecución (véase Sección 4.3.1), determinará, para cada punto, qué módulo de la plataforma heterogénea resulta más adecuado para realizar el cálculo. De este modo, el calendarizador actuará como un componente de decisión capaz de asignar los puntos de la integral a los módulos que ofrezca el mejor rendimiento.

Además, se implementará un *codelet* (Augonnet, Thibault, Namyst, y Wacrenier, 2009), encargado de gestionar la invocación de los distintos módulos de ejecución. En otras palabras, si el calendarizador determina que la integral debe evaluarse en el GPU, el *codelet* deberá contener la referencia al módulo implementado en CUDA. Por el contrario, si la ejecución se realiza en la CPU, el *codelet* deberá incluir las referencias a los módulos secuencial, OpenMP (para la ejecución paralela) y AVX (para la ejecución vectorizada con instrucciones de Intel). De esta manera, el *codelet* actuará como un mecanismo flexible que encapsula los diferentes módulos de ejecución y permite que el calendarizador seleccione la alternativa más conveniente.

Sin embargo, para que el calendarizador pueda tomar decisiones informadas, es necesario contar con datos confiables sobre el tiempo de ejecución de cada módulo en distintos escenarios (variando dimensiones y número de puntos). Obtener estos datos de manera manual sería impráctico, debido tanto a la cantidad de configuraciones posibles como a la naturaleza heterogénea de las plataformas de ejecución. Para superar esta limitación, este proyecto plantea el desarrollo de heurísticas capaces de estimar el tiempo de ejecución con rapidez y suficiente precisión. Dichas heurísticas no solo reducen el tiempo requerido para obtener los datos de los módulos de la plataforma, sino que además permiten generalizar el enfoque, facilitando su adaptación y uso en diferentes entornos heterogéneos sin necesidad de repetir todo el proceso de medición desde cero.

1.3 Objetivos generales y específicos del proyecto

General

Disminuir el tiempo de ejecución en la extrapolación de Romberg para el cálculo integrales multidimensionales en entornos heterogéneos.

Particulares

1. Estudiar los tipos de calendarizadores y heurísticas, adecuados en entornos heterogéneos, que se pueden implementar en este problema.

2. Extender la biblioteca de integrales multidimensionales con el módulo de ejecución AVX.

3. Determinar la cantidad mínima (n) de puntos en la integral de Gauss-Kronrod para tener un error aceptable (igual o menor a 10^{-4}).

4. Construir el codelet para administrar la biblioteca de integrales multidimensionales.

5. Hacer pruebas con el codelet para estimar el tiempo de ejecución y el error para diferentes dispositivos y componentes.

6. Seleccionar e implementar el/los calendarizador(es) o heurística(s) para despacho de procesos.

7. Validar los tiempos de ejecución y el error del mecanismo de integración multidimensional propuesto.

1.4 Antecedentes

Este proyecto de tesis pretende profundizar e integrar los trabajos que han sido desarrollados en el departamento de Computación del Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional unidad Zacatenco, con el propósito de optimizar el cálculo de integrales multidimensionales. A continuación, se describen brevemente los trabajos en los que se basa esta propuesta:

1.4.1 Paralelización del cálculo de integrales multidimensionales

(Quintero-Monsebaiz y cols., 2021) aborda los esquemas de paralelización en arquitecturas multi-núcleo: CUDA y OpenMP; los cuales, se emplean para acelerar y mejorar la precisión de los algoritmos de integración multidimensional adaptativos, en especifico, el método adaptativo unidimensional Gauss-Kronrod que se generaliza a 3, 4, 5 y 6 dimensiones y el método de extrapolaciones de Romberg.

Este trabajo replantea el proceso tradicional de integración multidimensional con el propósito de aprovechar de manera efectiva las ventajas de las arquitecturas multinúcleo. Para ello, se incorporan estrategias de paralelización basadas en el hecho de que las cuadraturas multidimensionales pueden evaluarse mediante ciclos cuyas iteraciones son independientes entre sí. Esta característica permite distribuir el trabajo

entre varios núcleos de procesamiento y, con ello, mejorar significativamente el rendimiento computacional.

Las pruebas fueron realizadas en el conjunto de funciones de referencia del *benchmark* de integrales multidimensionales (Arumugam y cols., 2013), estas pruebas mostraron que el algoritmo es numéricamente preciso, con aceleraciones de hasta 800X en CUDA y 300X en la implementación OpenMP, en comparación con un algoritmo de integración multidimensional secuencial.

1.4.2 Calendarizadores en ambientes heterogéneos

Debido a los grandes volúmenes de datos que se han generado en los últimos años, se ha experimentado una creciente demanda en la creación y expansión de centros de datos, algunos de éstos han sido integrados por conjuntos de servidores con capacidades variables de procesamiento y almacenamiento. Este tipo de entornos representa un nuevo desafío en la optimización y uso eficiente de los recursos, por lo que calendarizar tareas se vuelve más complicado.

Para enfrentar estas dificultades se han desarrollado diversas estrategias, basadas en indicadores que requieren la creación de una matriz de costos que vincule tareas y recursos. Esto implica una predicción precisa de los indicadores seleccionados, en función de las características de las tareas y los servidores. En (Salas-González, 2023) se desarrolla una caracterización detallada de tareas y recursos para crear un conjunto de atributos que sirven como entradas para diferentes algoritmos de Inteligencia Artificial (*Random Forest*, Regresión Lineal Múltiple, Redes Neuronales Secuenciales y Redes *Long Short-Term Memory*). 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.

1.5 Descripción del documento

El contenido de esta tesis se estructura de manera progresiva, partiendo de los fundamentos teóricos hasta llegar al análisis del sistema de calendarización propuesto.

El Capítulo 2 presenta los fundamentos teóricos necesarios para comprender el problema de la integración multidimensional y su complejidad computacional. Así como los conceptos de calendarización, plataformas heterogéneas y algunas heurísticas utilizadas en la asignación de tareas.

El Capítulo 3 describe la implementación del sistema de calendarización. Se detalla la estructura de la biblioteca de integrales multidimensionales utilizada como base. Posteriormente, se abordan las extensiones implementadas: el módulo de ejecución AVX, el *codelet* para administrar los módulos de la biblioteca y las tres estrategias de calendarización que gestionan el sistema: estática secuencial, dinámica secuencial y estática concurrente.

El Capítulo 4 presenta las pruebas realizadas para validar el funcionamiento del sistema, así como para evaluar la precisión de los resultados obtenidos en el cálculo

de las integrales y los tiempos de ejecución. Se llevaron a cabo pruebas preliminares para determinar la complejidad de las funciones del *benchmark* y estimar el número mínimo de puntos necesario para mantener un error aceptable. Además, se presentan las pruebas de rendimiento aplicadas a los calendarizadores en dos plataformas heterogéneas.

El Capítulo 5 contiene el análisis de los resultados obtenidos. Se comparan los tiempos de ejecución y la precisión de los módulos con y sin extrapolación de Romberg. Posteriormente, se evalúa el desempeño de los calendarizadores propuestos, con el objetivo de analizar su comportamiento y eficiencia.

En resumen, estos capítulos documentan el proceso de investigación, implementación, validación y análisis del sistema de calendarización.

Capítulo 2

Fundamentos

Este capítulo establece los fundamentos teóricos y tecnológicos que sustentan esta propuesta de tesis. En primer lugar, se describen algunos métodos numéricos disponibles para la resolución de integrales multidimensionales, destacando aquellos que ofrecen mayor precisión y eficiencia computacional. En la siguiente sección se presenta una revisión resumida sobre las bibliotecas especializadas para la resolución numérica de integrales multidimensionales: DCUHRE y CUHRE.

La tercera sección aborda las plataformas heterogéneas, poniendo especial énfasis en aquellas integradas por CPUs, que cuentan con núcleos de rendimiento y eficiencia, así como del conjunto de instrucciones vectoriales avanzadas (AVX), además de GPUs. Se discute cómo estas tecnologías contribuyen significativamente a la paralelización efectiva de tareas, logrando así importantes reducciones en los tiempos de ejecución. Posteriormente, en la cuarta sección se introducen los conceptos básicos asociados con los calendarizadores de tareas, clasificándolos en dos tipos: calendarizadores estáticos y dinámicos. Asimismo, se menciona la capacidad de estos sistemas para ejecutar tareas tanto de forma secuencial como concurrente.

En la quinta sección se realiza una revisión de trabajos relacionados, ofreciendo un análisis comparativo de diversos calendarizadores existentes en la literatura. En esta sección se resaltan las similitudes y diferencias entre los enfoques existentes y el enfoque propuesto en esta tesis, presentando finalmente una tabla comparativa de características clave.

Finalmente, la sexta sección introducirá las estrategias de Inteligencia Artificial (IA) orientadas a la calendarización de tareas en plataformas heterogéneas. Se destacarán particularmente las técnicas basadas en árboles de decisión y búsqueda tabú, pues éstas son las estrategias que serán implementadas dentro de este proyecto de tesis.

2.1 Integrales multidimensionales

La integración numérica por cuadratura es un método que busca aproximarse al valor de una integral definida. Para ello, se emplea una regla de cuadratura que utiliza un

conjunto finito de puntos evaluados en una función dada $f(x)$:

$$\int_a^b f(x)\,dx = \sum_{i=1}^{\infty} w_i f(x_i), \tag{2.1}$$

donde la integral se realiza sobre un intervalo cerrado $[a, b]$, $x_1, x_2, ..., x_n$ son los nodos en el intervalo de integración y $w_1, w_2, ..., w_n$ son los pesos asociados a cada nodo. La precisión de las fórmulas de integración depende de los pesos y las abscisas elegidos.

Una de las cuadraturas más simples se crea ubicando las abscisas sin espaciado especial y resolviendo $n + 1$ ecuaciones para obtener los pesos correspondientes. Este enfoque tiene el doble de precisión en comparación a la regla de Newton-Cotes. Esta cuadratura organiza los pesos y abscisas elegidos, con el propósito de integrar una clase de integrales que se pueden calcular mediante la multiplicación de un polinomio por una función de peso $W(x)$ (Abramowitz, 1974; Arfken, 1985; Szegő, 1975), es decir

$$\int_a^b W(x)f(x)\,dx = \sum_{i=1}^{n} w_i p_i(x_i), \tag{2.2}$$

donde $p_1(x), p_2(x), ..., p_n(x)$ son polinomios ortogonales cualesquiera de grado $2n - 1$ o menor.

En esta tesis se pretende utilizar la cuadratura de Gauss-Kronrod, sin embargo, primero es necesario abordar algunas propiedades generales de la regla de cuadratura de Gauss-Chebyshev (Abramowitz, 1974), donde los polinomios son ortogonales en el intervalo $[-1, 1]$, y $W(x) = (1-x^2)^{-\frac{1}{2}}$. Para obtener los polinomios de Chebyshev del primer tipo se emplea de manera directa la fórmula de recurrencia de tres términos:

$$\begin{aligned} T_0(x) &= 1, \\ T_1(x) &= x, \\ T_{n+1}(x) &= 2xT_n(x) - T_{n-1}(x). \end{aligned} \tag{2.3}$$

Por otro lado, para obtener los pesos se aplica la siguiente fórmula:

$$w_i = -\frac{\pi}{T_{n+1}(x_i)T_n'(x_i)}, \tag{2.4}$$

donde $T_n(x_i)'$ es la derivada del polinomio ortogonal en su cero x_i , lo cual es sencillo en su forma trigonométrica.

El procedimiento práctico para calcular la integral (Ecuación 2.2) mediante una cuadratura, comienza por generar el polinomio de Chebyshev utilizando la relación de puntos del árbol de recursión (Ecuación 2.3). Para posteriormente determinar los

ceros de Gauss-Chebyshev, mediante el método de Newton y finalmente se calculan los pesos asociados utilizando la Ecuación 2.4 (Dahlquist y Björck, 2008).

Sin embargo, las cuadraturas deterministas no adaptativas, como Gauss-Chebyshev, no son precisas cuando el integrando es altamente oscilatorio (Notaris, 2016); por lo que en este caso, los métodos deterministas adaptativos son una mejor opción para lidiar con la complejidad. En el método adaptativo se refina la cuadrícula en los intervalos donde el error local es mayor. Una de las cuadraturas adaptativas más populares es Gauss-Kronrod (Kronrod, 1964). En esta cuadratura, se eligen los pesos y las abscisas con el fin de maximizar la precisión, ya que hay $3n + 1$ variables, los algoritmos buscan secuencias óptimas de reglas, en donde cada secuencia reutiliza todas las abscisas de su predecesora. La fórmula de cuadratura de Gauss-Kronrod está dada por:

$$\int_a^b f(x)\,dx = \sum_{i=1}^{n} w_i f(x_i) + \sum_{i=1}^{n+1} v_i f(\xi_i), \tag{2.5}$$

donde x_i y w_i son los nodos y pesos de Gauss, respectivamente, mientras que las nuevas abscisas ξ_i y los pesos v_i son las abscisas y los pesos de Kronrod, que se eligen para maximizar la precisión a al menos $2n + 1$.

Kronrod evalúa la Ecuación 2.5 para $n > 40$ asegurando que ξ_i se encuentra en el intervalo $[1, -1]$ y w_i, v_i ambos definidos como positivos. Las abscisas ξ_i son las raíces del polinomio de Kronrod K_{n+1} que se evalúa por medio de una ecuación de recurrencia (Piessens y Branders, 1974), mientras que los pesos w_i, v_i se obtienen resolviendo un sistema lineal, para alcanzar la mayor precisión en la Ecuación 2.5. No obstante, el método propuesto por Kronrod para obtener los coeficientes de K_{n+1} requiere una ecuación que no se comporta bien (Szegö, 1935). En consecuencia, Patterson propuso una expansión de K_{n+1} en términos de polinomios de Legendre $P_n(x)$ (Patterson, 1968). Sin embargo, este método no aplica cuando se utilizan $N = 1, 9, 17, 22, 27, 35, 36, 37$ y 40 puntos. Por lo que, Pissens y Branders (Piessens y Branders, 1974) propusieron una expansión del polinomio $K_{n+1}(x)$ en términos de polinomios de Chebyshev para números pares e impares. Así, utilizando algunas propiedades de los polinomios de Chebyshev, los pesos se obtienen de la siguiente manera:

$$w_i = \frac{C_N}{2P'_N(x_i)K_{n+1}(x_i)} + \frac{2}{(N+1)(N+2)[P_{N+1}(x_i)]^2}, \tag{2.6}$$

$$v_i = \frac{N+2}{2(N+1)} \frac{C_N}{[P_{N+1}(\xi_i) - \xi_i P_{N+1}(\xi_i)]K'_{n+1}(\xi_i)}, \tag{2.7}$$

donde, P_{N+1} es un polinomio de Legendre de grado $N + 1$, y K'_{n+1} es un polinomio de Kronrod de grado $n + 1$; los órdenes de estos polinomios son diferentes.

El método de Piessens es aplicable para cualquier caso, incluyendo números pares e impares, además, desde el punto de vista computacional es menos costoso. Por ende, el tiempo de ejecución se puede reducir a la mitad en comparación con el método de Patterson.

Dado que esta tesis tiene como objetivo reducir el tiempo de ejecución y aumentar la precisión de la integración multidimensional, se empleará el método de Piessens para la generación de nodos y abscisas. Por lo que es necesario generalizar la metodología unidimensional a la integración de d-dimensiones.

Dicho lo anterior, se considera la evaluación de la integral d-dimensional sobre un hipervolumen Ω mediante una cuadratura Gaussiana aproximada para d-dimensiones (Keister, 1996)

$$\int_{\Omega} W(\mathbf{X}) f(\mathbf{X})\, d\mathbf{X}, \tag{2.8}$$

donde $\mathbf{X}$ es un vector de dimensión d.

Bajo este contexto, se necesitan n^d puntos en la cuadrícula hiperdimensional. La evaluación de la integral multidimensional (Ecuación 2.8) requiere un arreglo de pesos de rango d $W(\mathbf{X})$ que se obtiene por un producto directo del vector de pesos $\mathbf{w}_n^j$ para cada dimensión j, generando un tensor de pesos de rango d

$$\mathbf{w}_n^j = \{w_1^j, w_2^j, ..., w_p^j, ..., w_n^j\}, \qquad W_n^d = \bigoplus_{j=1}^{d} \mathbf{w}_n^j, \tag{2.9}$$

donde $\mathbf{w}_n^j$ es un vector que va desde 1 hasta el número de puntos de la cuadrícula n, y j recorre las dimensiones presentes en la integral; el tensor de pesos W_n^j es el producto directo de los vectores de pesos que cubren las d-dimensiones.

La generación de los nodos para la función n-dimensional $f(\mathbf{X})$ es similar a la generación de los pesos, es decir, se construye una matriz d-dimensional de puntos $\mathbf{x}$ mediante el producto directo de cada vector unidimensional. En el caso de las abscisas, se construye una matriz n-dimensional $\mathbf{X}$ evaluando la función en cada punto de la cuadrícula n-dimensional:

$$\mathbf{x}_n^j = \{x_1^j, x_2^j, ..., x_p^j, ..., x_n^j\}, \qquad X_n^d = \bigoplus_{j=1}^{d} \mathbf{x}_n^j. \tag{2.10}$$

Una vez construido el tensor de orden de rango d $\mathbf{X}$, se evalúa el integrando en este arreglo, y para calcular la integral d-dimensional se lleva a cabo la contracción de los arreglos de orden d $W(\mathbf{X})$ y $\mathbf{X}$ de la siguiente manera:

$$\int_{\Omega} W(\mathbf{x}) f(\mathbf{x})\, d\mathbf{x} = W_n^d * f(X_n^d) = \sum_{j=1}^{d} \sum_{i=1}^{n} W_i^j f(X_i^j), \tag{2.11}$$

donde $*$ denota la contracción de los tensores.

2.2 Bibliotecas para la resolución de integrales multidimensionales

A continuación se describen dos bibliotecas ampliamente utilizadas para la resolución de integrales multidimensionales: DCUHRE y CUHRE. Ambas forman parte del conjunto de algoritmos del paquete CUBA y se han consolidado como herramientas eficientes para la evaluación de funciones con comportamiento complejo.

2.2.1 DCUHRE

DCUHRE (Berntsen, Espelid, y Genz, 1991b) es una biblioteca implementada en FORTRAN 77 para el cálculo de integrales multidimensionales de doble precisión sobre regiones hiperrectangulares. Esta biblioteca proporciona un algoritmo adaptativo (Berntsen, Espelid, y Genz, 1991a), basado en una estrategia de subdivisión global adaptativa, que permite refinar selectivamente las subregiones de integración donde el integrando presenta mayor complejidad o variación local. Además, su estructura está diseñada para facilitar la implementación en dispositivos paralelos con memoria compartida.

El procedimiento del algoritmo consiste en elegir una subregión de la integral y determinar si requiere refinamiento adicional según el comportamiento del integrando. Si los integrandos, en un vector de integrandos, son significativamente diferentes entre sí, el algoritmo divide dicho vector en subvectores más pequeños, aplicando el proceso de integración de manera independiente a cada uno. Por el contrario, cuando los integrandos comparten suficiente similitud, es posible reutilizar la misma estrategia de subdivisión, con lo que se reduce el costo computacional en tiempo y memoria. Esta característica también permite aprovechar la ejecución paralela de las evaluaciones del integrando, logrando así una optimización adicional del rendimiento.

2.2.2 CUHRE

La biblioteca CUBA (Hahn, 2005) incluye el algoritmo CUHRE, un algoritmo determinista de integración multidimensional que utiliza reglas de cuadratura para estimar subregiones dentro de un esquema de subdivisión globalmente adaptativo. A diferencia de los métodos basados en Monte Carlo, CUHRE se fundamenta en aproximaciones polinomiales del integrando y en una estrategia de refinamiento. En cada iteración, la subregión con mayor error estimado es dividida por la mitad a lo largo del eje donde el integrando presenta la diferencia de cuarto orden más pronunciada, lo que permite identificar y refinar de manera eficiente las zonas con mayor dificultad numérica.

El procedimiento del algoritmo puede resumirse de la siguiente manera:

1. Seleccionar la región con el mayor error estimado.
2. Dividir dicha región a lo largo del eje correspondiente a la mayor diferencia de cuarto orden.

3. Aplicar la regla de cuadratura a las dos subregiones generadas.

4. Incorporar las subregiones a la lista general de regiones y actualizar los resultados totales de la integral y el error.

En dimensiones moderadas, CUHRE ofrece un rendimiento competitivo, especialmente cuando el integrando puede aproximarse adecuadamente mediante polinomios. No obstante, conforme aumentan las dimensiones del problema, el número de puntos requeridos por las reglas de cuadratura crece de manera significativa, lo que reduce su eficiencia y limita su aplicación en espacios de altas dimensiones.

2.3 Plataformas heterogéneas

Durante la última década, las plataformas heterogéneas se han vuelto populares, debido a que ofrecen un alto rendimiento y son eficientes en cuanto al consumo energético se refiere, en comparación con las plataformas homogéneas. Por tanto, estas plataformas han comenzado a ser ampliamente utilizadas en clústeres de alto rendimiento y estaciones de trabajo, lo que genera la necesidad de contar con una visión general y una comprensión profunda de las mismas.

Las plataformas heterogéneas integran procesadores con distintas arquitecturas, conjuntos de instrucciones y representaciones de datos en un mismo sistema (Salas-González, 2023), con el objetivo de optimizar el rendimiento y/o reducir el consumo de energía. A diferencia de las plataformas homogéneas convencionales, que suelen emplear CPUs multi-núcleos simétricos, las plataformas heterogéneas pueden estar compuestas por CPUs, GPUs e incluso FPGAs, esto permite que se usen tanto para tareas de propósito general como tareas más especificas que requieren mayor eficiencia. En este capítulo se abordarán los componentes de ejecución CPU y GPU, puesto que serán de vital importancia para este proyecto. Sin embargo, primero es necesario entender cómo surgieron estas plataformas.

En 1965, Gordon Moore formuló una observación conocida como la Ley de Moore (Moore, 1965), según la cual cada dos años se duplicará la cantidad de transistores integrados en un microprocesador, esta tendencia se mantuvo durante más de treinta años (Mollick, 2006). No obstante, aunque ahora es posible instalar aproximadamente 16,000 millones de transistores en un sólo microprocesador debido a su tamaño de 32 nm, no es económico. Además, existen ciertas limitaciones físicas que han frenado e incluso revertido ligeramente el crecimiento exponencial de la frecuencia (Brodtkorb, Dyken, Hagen, Hjelmervik, y Storaasli, 2010), una muy importante es la densidad de potencia.

A raíz de la Ley de Moore, se asumió que la frecuencia de los procesadores de un sólo núcleo también aumentaría de forma sostenida con el tiempo. Sin embargo, la densidad de potencia en los procesadores superó la de los disipadores de calor y en consecuencia, rebaso el limite del umbral de calor que el silicio puede soportar con los métodos de refrigeración convencionales. Aunque se desarrollaron nuevas tecnologías

de refrigeración, como nitrógeno líquido o hidrógeno (Brodtkorb y cols., 2010), estas soluciones resultaron inviables debido a sus elevados costos. Estas limitaciones han conducido a la evolución del diseño de los procesadores actuales, con el objetivo de mantener un equilibrio entre el rendimiento y los costes de desarrollo. En lugar de seguir aumentando la frecuencia, los procesadores fueron rediseñados para incorporar múltiples núcleos simétricos de baja frecuencia, comenzando con configuraciones de dos núcleos y evolucionando hacia arquitecturas de cuatro, ocho o más núcleos, permitiendo así la ejecución simultánea de múltiples programas.

Sin embargo, una desventaja de los procesadores multi-núcleo simétricos es que utilizan la misma cantidad de transistores y operan a frecuencias similares tanto para tareas simples como complejas, lo que implica un consumo energético constante, incluso cuando las tareas no requieren un procesamiento intensivo. Ante esto, las arquitecturas heterogéneas han surgido como una alternativa viable, combinando núcleos tradicionales de rendimiento con núcleos de eficiencia (Brodtkorb y cols., 2010). Los núcleos de eficiencia, generalmente usados en tareas mas sencillas, están diseñados para optimizar el rendimiento dentro de un presupuesto energético o de transistores determinado, lo cual generalmente implica un menor número de transistores, frecuencias más bajas y una funcionalidad reducida.

Además, Intel ha incorporado un módulo especializado en instrucciones vectorizadas dentro de los núcleos de rendimiento de sus procesadores, conocido como AVX (*Advanced Vector Extensions*). Este conjunto de instrucciones SIMD (*Single Instruction, Multiple Data*) busca mejorar el rendimiento en operaciones que pueden aprovechar el paralelismo a nivel de datos. AVX permite que los procesadores realicen una misma operación sobre múltiples datos al mismo tiempo, lo que lo hace especialmente útil en aplicaciones científicas, multimedia, de gráficos, criptografía y *machine learning*. AVX amplía la capacidad SIMD de los procesadores x86 mediante registros vectoriales de 256 bits, lo que permite procesar múltiples valores de punto flotante o enteros de manera simultánea.

Desde una perspectiva algorítmica, tareas altamente paralelizables, como las simulaciones de Monte Carlo, obtienen mayores ventajas al ejecutarse en procesadores heterogéneos, puesto que, la mayoría de las aplicaciones se componen de una combinación de tareas en serie y en paralelo. No obstante, la idoneidad de un procesador para una tarea específica dependerá de sus características arquitectónicas particulares (Asanović y cols., 2006; Hill y Marty, 2008).

Con el reciente énfasis en el cómputo de alto desempeño (*High Performance Computing*), se vuelve fundamental utilizar todos los recursos disponibles de los sistemas heterogéneos en cada ciclo de reloj. Tanto en el ámbito académico como en la industria, se reconoce que el rendimiento en serie ha alcanzado su punto máximo, lo que ha impulsado un enfoque creciente en el desarrollo de nuevos algoritmos capaces de aprovechar arquitecturas paralelas y heterogéneas. Por lo que, no sólo las CPUs han evolucionado, sino que también lo han hecho las GPUs.

La GPU originalmente fue diseñada para aplicaciones gráficas, especialmente en videojuegos, donde se encarga de renderizar imágenes 2D a partir de objetos geomé-

tricos tridimensionales, mediante un conjunto de procesadores que operan en paralelo para calcular el color de cada pixel (Brodtkorb y cols., 2010). Con el tiempo, las GPU han evolucionado en arquitecturas más generales, donde el renderizado gráfico es sólo una de sus múltiples aplicaciones. Actualmente, su rendimiento las convierte en una opción atractiva para tareas de HPC y el entrenamiento de redes neuronales en IA. No obstante, una de sus principales limitaciones es que, en la mayoría de los sistemas, las GPUs están conectadas a través del bus PCI Express, lo que puede convertirse en un cuello de botella en la transferencia de datos, limitando la velocidad de subida y descarga entre la CPU y la GPU. A pesar de este inconveniente, las GPUs son capaces de procesar grandes volúmenes de datos en un tiempo significativamente reducido, lo que las hace ideales para tareas que requieren el manejo de grandes cantidades de información.

En esencia, una GPU es un procesador multi-núcleo simétrico al que la CPU accede y controla exclusivamente (Brodtkorb y cols., 2010), lo que las convierte en un sistema heterogéneo. La GPU opera de forma asíncrona con respecto a la CPU, lo que permite la ejecución y la transferencia de memoria simultáneas.

El campo de la computación heterogénea abarca una amplia variedad de arquitecturas y áreas de aplicación, y actualmente no existe una teoría unificada que las integre por completo. En consecuencia, en este proyecto se emplearán plataformas heterogéneas compuestas por una CPU y una GPU para realizar pruebas. Si bien las capacidades específicas de cada componente varían según la plataforma, todas comparten una estructura general similar a la representada en la Figura 2.1.

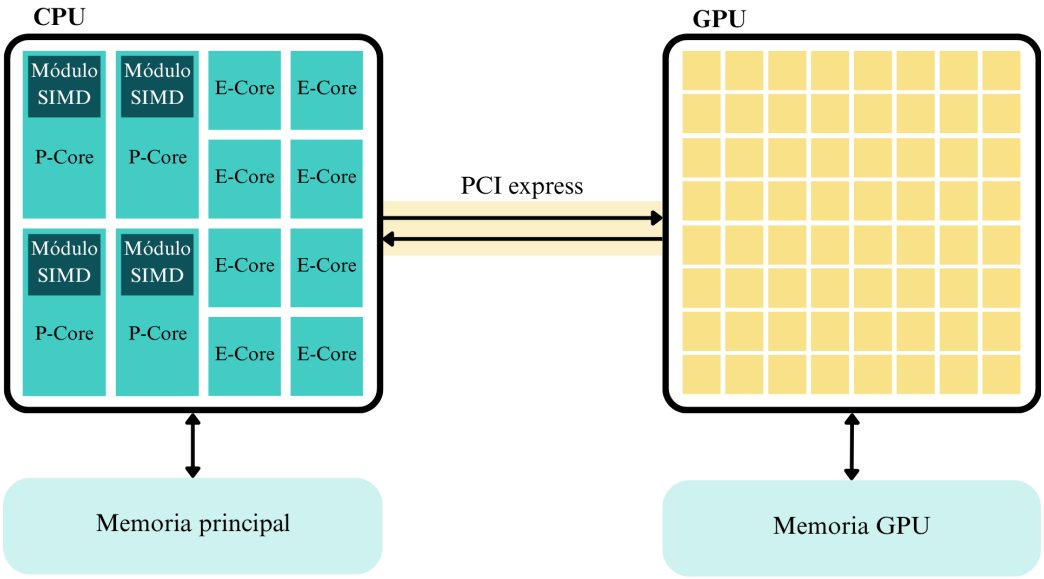


Figura 2.1: Estructura general de una plataforma heterogénea compuesta por una CPU y una GPU. P-Core hace referencia a los núcleos de rendimiento (*performance cores*) y E-Core a los núcleos de eficiencia (*efficient cores*)

.

2.4 Calendarizadores

El objetivo fundamental de la multi-programación es mantener la CPU ocupada la mayor parte del tiempo, maximizando así su utilización global y asegurando un mejor aprovechamiento de los recursos. Por otro lado, el tiempo compartido tiene como propósito alternar rápidamente entre múltiples procesos dentro de un mismo núcleo del procesador, generando la percepción de ejecución simultánea e interactiva para los usuarios. Para lograr estos objetivos, el calendarizador de procesos desempeña un papel central: selecciona, entre los procesos listos para ejecutarse, aquel que será ejecutado dentro del núcleo. Es importante destacar que cada núcleo físico sólo puede ejecutar un proceso a la vez, de modo que la eficiencia del sistema depende en gran medida de la política de calendarización aplicada.

Históricamente, los calendarizadores surgieron como un mecanismo para gestionar la competencia entre procesos en sistemas de propósito general. Sin embargo, conforme la arquitectura de los sistemas ha evolucionado, desde procesadores mononúcleo hasta plataformas multinúcleo, distribuidas y heterogéneas, también lo han hecho las estrategias de calendarización. En la actualidad, el calendarizador no sólo determina qué proceso ejecuta la CPU, sino que además puede gestionar múltiples tipos de recursos: núcleos de CPU, GPUs, FPGAs, aceleradores vectoriales y procesadores especializados, entre otros.

Diversas definiciones encontradas en la literatura describen al calendarizador como "un módulo que implementa políticas de planificación de procesos y asigna recursos en función de éstas" (Coulouris, Dollimore, Kindberg, y Blair, 2005), o como "el módulo responsable de seleccionar, entre los procesos en ejecución, aquél que debería recibir el próximo intervalo de tiempo de CPU" (Silberschatz, Galvin, y Gagne, 2012). Estas definiciones se enfocan principalmente en la asignación de tiempo de CPU; sin embargo, en el contexto de esta tesis es necesario ampliar esta perspectiva hacia plataformas heterogéneas.

Tal como señalan (Sterling, Anderson, y Brodowicz, 2017) y (Jeannot y Zilinskas, 2014), el calendarizador puede verse como "el componente de un sistema de alto rendimiento que decide cuáles son los trabajos más importantes que deben ejecutarse a continuación, y en qué recurso de *hardware*", así 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". Estas definiciones contemplan el uso de otro procesadores, como GPUs, FPGAs y aceleradores especializados, en el proceso de asignación, reflejando la importancia de distribuir eficientemente la carga de trabajo para obtener un mejor rendimiento y una menor demanda energética.

En función de sus características operativas, los calendarizadores suelen clasificarse en dos grandes categorías: calendarizadores estáticos y calendarizadores dinámicos.

2.4.1 Calendarizadores estáticos

Los calendarizadores estáticos asignan las tareas a los recursos antes de que comience la ejecución del programa (Lavaei, Noghabi, Chen, y Xue, 2018). Este tipo de planificación determina de manera anticipada dónde y cuándo será ejecutada cada tarea, basándose generalmente en información previa del sistema, estimaciones de carga o un modelo conocido de la aplicación.

Entre las ventajas que ofrecen este tipo de calendarizadores se encuentran (Buyya, Vecchiola, y Selvi, 2013; Dastjerdi, Gupta, Calheiros, Ghosh, y Buyya, 2016):

- Son más adecuados para cargas de trabajo de alta demanda y para entornos en los que la configuración del *hardware* no cambia con frecuencia.
- Permiten un control más preciso sobre la asignación de recursos y los tiempos de ejecución.
- Resultan apropiados para aplicaciones con requisitos de tiempo real y para cargas con alta demanda de procesamiento.

No obstante, también presentan limitaciones importantes (Sotiriades, Petraki, Kartsakli, Souravlias, y Bouganis, 2015; Al-Khateeb, Benkhlifa, y Bounceur, 2018):

- Su rigidez dificulta la adaptación a escenarios con variabilidad en la carga de trabajo o cambios en la disponibilidad de recursos, lo que puede llevar a desequilibrios en la carga.
- Pueden derivar en un uso ineficiente del sistema si la estimación inicial no refleja adecuadamente el comportamiento real de las tareas.

Los calendarizadores estáticos son especialmente útiles en entornos donde la predictibilidad es alta, pero su falta de flexibilidad limita su efectividad en plataformas heterogéneas con cargas dinámicas.

2.4.2 Calendarizadores dinámicos

A diferencia de los estáticos, los calendarizadores dinámicos toman decisiones en tiempo de ejecución, lo que les permite adaptarse a las condiciones cambiantes del sistema, variaciones en la carga de trabajo o fluctuaciones en el rendimiento del *hardware* (Lavaei y cols., 2018).

Entre sus principales ventajas destacan (Sotiriades y cols., 2015; Al-Khateeb y cols., 2018):

- Se ajustan rápidamente a cambios imprevistos en el sistema, redistribuyendo las tareas según la disponibilidad real de los recursos, mejorando así el rendimiento.
- Son especialmente efectivos en plataformas heterogéneas, donde las capacidades de los recursos pueden diferir significativamente.
- Mejoran la eficiencia energética al asignar tareas al recurso más adecuado en el momento oportuno.

Sin embargo, también presentan ciertas limitaciones (Al-Khateeb y cols., 2018; J. Liu, Li, Li, Qian, y Zhan, 2019):

- Requieren un mayor costo computacional, ya que implican la monitorización constante del sistema y la toma continua de decisiones en tiempo de ejecución.
- Pueden generar sobrecarga de comunicación entre recursos o nodos, lo cual influye en la latencia total y puede originar cuellos de botella.

A pesar de estas desventajas, los calendarizadores dinámicos son la opción preferida en sistemas modernos de alto rendimiento debido a su flexibilidad y capacidad de adaptación.

2.4.3 Ejecución secuencial y concurrente

Tanto los calendarizadores estáticos como los dinámicos pueden asignar tareas de dos maneras principales: de forma secuencial o de manera concurrente. En la ejecución secuencial, las tareas se procesan una a la vez en un único recurso, lo que resulta más simple de implementar y permite un mayor control sobre los flujos de ejecución. No obstante, limita el rendimiento cuando se dispone de múltiples unidades de procesamiento.

Por el contrario, la ejecución concurrente divide la carga en subtareas que pueden ejecutarse simultáneamente en distintos recursos del sistema, como los núcleos de CPU, GPUs o aceleradores vectoriales. Este enfoque permite obtener mejoras significativas en el rendimiento, especialmente en aplicaciones altamente paralelizables. Sin embargo, introduce una complejidad adicional, ya que requiere un balanceo de carga adecuado, la coordinación entre recursos y mecanismos para evitar que el paralelismo genere sobrecargas innecesarias.

En entornos heterogéneos, la elección entre ejecución secuencial o concurrente, así como el diseño del calendarizador que las gestiona, afecta directamente la eficiencia global del sistema, el tiempo de ejecución y el consumo energético.

En la Tabla 2.1 se ilustran los tres tipos de calendarizadores que se abordan en este proyecto para el cálculo de integrales multidimensionales en plataformas heterogéneas: el calendarizador estático secuencial, el calendarizador dinámico secuencial y el calendarizador estático concurrente. Cada uno de ellos representa un enfoque distinto respecto a la asignación de tareas y el uso de los recursos de cómputo, permitiendo evaluar sus ventajas, limitaciones y su impacto en el rendimiento bajo diferentes configuraciones de *hardware* y estrategias de paralelización. Así como un cuarto tipo de calendarizador de mayor complejidad que podría ser explorado en una tesis doctoral futura.

A continuación, se describen las características y el funcionamiento de cada uno.

Calendarizador estático secuencial

El calendarizador estático secuencial asigna todos los puntos de integración siguiendo una estrategia de planificación fija, determinada antes del inicio de la ejecu-

Tabla 2.1: Clasificación de calendarizadores que se abordan en este proyecto de tesis.

	Calendarizador	
	Secuencial	Concurrente
Estático	✓	✓
Dinámico	✓	×

ción. En este enfoque, cada grupo de puntos se distribuye de forma predeterminada, sin realizar ajustes durante el tiempo de ejecución.

Este tipo de calendarizador permite analizar el comportamiento base del sistema de calendarización cuando los puntos se procesan uno a la vez, sin paralelismo ni redistribución dinámica. Su principal ventaja es la simplicidad, pues no requiere mecanismos de sincronización ni monitoreo del estado del sistema. No obstante, su rendimiento se ve limitado por la falta de paralelismo y por su incapacidad para adaptarse a variaciones en la complejidad computacional de los distintos grupos de puntos.

Calendarizador dinámico secuencial

En contraste con el enfoque anterior, el calendarizador dinámico secuencial ajusta en tiempo de ejecución la asignación de puntos. Aunque la ejecución sigue siendo secuencial, el calendarizador decide dinámicamente el orden en que los puntos deben evaluarse.

Este enfoque resulta útil cuando los subintervalos presentan comportamientos numéricos heterogéneos, permitiendo equilibrar la carga de forma adaptativa incluso en un entorno sin paralelismo. Su desventaja principal radica en la sobrecarga introducida por la toma de decisiones en tiempo de ejecución, la cual puede ser significativa cuando el número de puntos es grande y el beneficio adaptativo es limitado.

Calendarizador estático concurrente

El calendarizador estático concurrente extiende el modelo estático hacia un entorno paralelo y heterogéneo. En este caso, los puntos se distribuyen entre múltiples módulos de ejecución (secuencial, CUDA, OpenMP, AVX), siguiendo una asignación determinada antes de iniciar la ejecución. Cada módulo recibe un conjunto previamente definido de puntos y los procesa de manera paralela, sin interacción ni reconfiguración durante la evaluación.

Este enfoque permite aprovechar de forma eficiente plataformas que cuentan con múltiples recursos, reduciendo significativamente el tiempo de ejecución. Sin embargo, su desempeño depende fuertemente de que la carga esté equilibrada desde un principio. Cuando la estimación de la carga asociada a cada módulo no es precisa o cuando los recursos presentan variaciones de rendimiento, este esquema puede derivar en

desbalances que reducen la ganancia paralela.

Calendarizador dinámico concurrente

Como parte del trabajo a futuro, se plantea el desarrollo de un calendarizador dinámico concurrente, el cual combinaría la capacidad adaptativa del enfoque dinámico con el aprovechamiento simultáneo de múltiples recursos. Este tipo de calendarizador ajustaría la asignación de puntos en tiempo de ejecución, redistribuyendo los puntos de integración entre los módulos disponibles conforme cambia la carga, el estado del sistema o el rendimiento de los módulos de ejecución.

Un calendarizador dinámico concurrente permitiría gestionar aplicaciones en escenarios heterogéneos más complejos, donde las características del *hardware* pueden variar con el tiempo o donde la distribución estática resulta insuficiente para garantizar un balance óptimo de carga. Además, facilitaría la explotación de paralelismo a nivel de tareas con un control fino del rendimiento, pudiendo integrar métricas adicionales como consumo energético, latencias de comunicación o prioridades específicas de los módulos de integración.

El desarrollo de este calendarizador se plantea como parte de una tesis doctoral futura, debido a la complejidad de los mecanismos necesarios para la toma de decisiones, la coordinación entre recursos y la integración de políticas de reasignación adaptativa en tiempo de ejecución.

2.5 Trabajos relacionados

En esta sección se presentan diversos trabajos relacionados con el tópico de esta tesis. Se incluyen tanto *frameworks* diseñados para facilitar el desarrollo de calendarizadores de tareas en entornos heterogéneos, como propuestas concretas de calendarizadores que emplean heurísticas o técnicas de inteligencia artificial con el objetivo de mejorar la asignación y ejecución de tareas.

2.5.1 Planificador inteligente para integración numérica multidimensional en ambientes heterogéneos

En (Morales y Puga, 2022) se presenta el diseño, implementación y evaluación de un calendarizador basado en aprendizaje automático, cuyo objetivo es mejorar el rendimiento en la evaluación de integrales multidimensionales.

Para ello, el calendarizador debe ser capaz de asignar dinámicamente las tareas a distintas arquitecturas de procesamiento heterogéneo, mediante la búsqueda tabú. Este mecanismo permite seleccionar, en tiempo de ejecución, el módulo más eficiente para cada punto de integración.

Los resultados muestran que el calendarizador mejora significativamente el tiempo de ejecución, reduce los desequilibrios de carga y aprovecha de manera más eficiente los recursos computacionales heterogéneos. Además, se observa que la búsqueda tabú

proporciona una selección precisa y de bajo costo computacional para cada punto de integración. Sin embargo, dichos resultados fueron simulados, por lo que no se puede asegurar que el calendarizador funcione de la misma manera en un entorno real.

2.5.2 FlexTensor: un *framework* de exploración y optimización de calendarización automática para el cálculo de tensores

El artículo (Zheng, Liang, Wang, Chen, y Sheng, 2020) presenta FlexTensor, un *framework* para la exploración y optimización de calendarizadores orientados al cálculo de tensores en sistemas heterogéneos. FlexTensor combina heurísticas con técnicas de aprendizaje automático para generar calendarizadores de alto rendimiento capaces de adaptarse a distintos tipos de *hardware*, incluyendo CPU, GPU y FPGA.

En los experimentos se probaron 12 tipos diferentes de cálculos tensonriales, FlexTensor demostró mejoras significativas de rendimiento en todos. En particular, obtuvo una aceleración promedio de 1.83× usando la GPU NVIDIA V100 contra cuDNN, una aceleración de 1.72× usando procesadores Intel Xeon contra MKL-DNN para el caso de convoluciones 2D y una aceleración de 1.5× usando la FPGA Xilinx VU9P respecto a las líneas base de OpenCL.

2.5.3 StarPU: una plataforma unificada para la calendarización de tareas

STARPU (Augonnet y cols., 2009) es una biblioteca de calendarización de tareas, diseñada para arquitecturas multi-núcleo heterogéneas. Su objetivo principal es proporcionar un modelo de ejecución uniforme que abstrae la complejidad del *hardware* y permite al programador concentrarse en la definición de tareas. Para ello, ofrece un *framework* de alto nivel que facilita el diseño de políticas de calendarización, así como una biblioteca que automatiza las transferencias de datos.

Además, ofrece una abstracción de tareas descargable y unificada llamada *codelet*. El *codelet* es capaz de gestionar múltiples implementaciones especializadas de una misma tarea para distintos componentes de una arquitectura heterogénea (e.g., una versión optimizada para GPU y otra para CPU), además de implementaciones paralelas (e.g., OpenMP). STARPU se encarga de determinar, calendarizar y ejecutar las implementaciones más adecuadas en cada componente disponible, explotando al máximo la heterogeneidad del sistema (por ejemplo, combinando CUDA y OpenCL). Asimismo, cuando las implementaciones lo permiten, STARPU puede ejecutar funciones en paralelo sobre varios componentes de manera simultánea, con el fin de maximizar el rendimiento global.

2.5.4 Algoritmo de calendarización de tareas basado en aprendizaje por refuerzo

El artículo (Song, Li, Tian, y Song, 2023) propone un algoritmo de calendarización de tareas en grafos acíclicos dirigidos (DAG) para entornos heterogéneos, combinando aprendizaje profundo con técnicas heurísticas. El enfoque consta de tres componentes principales: una red convolucional de grafos, una red de políticas que proponen los autores y un algoritmo heurístico de calendarización. El proceso inicia enviando las características de las tareas a la red convolucional de grafos, la cual aprende las propiedades estructurales del DAG y genera representaciones de alto nivel para cada tarea. Estas representaciones se entregan a la red de políticas, encargada de seleccionar la siguiente tarea a ejecutar. Posteriormente, la tarea elegida se asigna al procesador más adecuado, de acuerdo con el algoritmo heurístico propuesto. Este ciclo se repite hasta que todas las tareas han sido asignadas.

El algoritmo aprende y ajusta continuamente sus estrategias a medida que interactúa con el entorno, lo que permite mejorar su capacidad de decisión. La incorporación del componente heurístico contribuye a agilizar el proceso de selección de procesadores y a incrementar el rendimiento global de la calendarización.

2.5.5 Regla de Johnson para la calendarización de n tareas en dos máquinas

Una de las variantes clásicas del problema de calendarización de tareas es la asignación de dos máquinas para procesar un conjunto de n tareas, donde todas deben seguir el mismo orden de procesamiento. Para este escenario, una de las estrategias más utilizadas es la regla de Johnson (Garey y Johnson, 1976), cuyo objetivo es minimizar el tiempo de ejecución (*makespan*) requerido para completar todas las tareas.

La regla de Johnson establece una forma sistemática de ordenar las tareas para obtener la secuencia óptima. El procedimiento se basa en los siguientes pasos:

1. Registrar los tiempos de ejecución de cada tarea en ambas máquinas.
2. Seleccionar la tarea con el menor tiempo.
3. Determinar la posición del tarea asociada a ese menor tiempo:
    - Si el menor tiempo corresponde a la primera máquina, la tarea se coloca al inicio de la secuencia.
    - Si el menor tiempo corresponde a la segunda máquina, la tarea se coloca al final de la secuencia.
    - En caso de empate, la tarea se ejecuta en la primera máquina.
4. Repetir el proceso hasta ordenar todas las tareas.

Este método garantiza una secuencia óptima para el caso de dos máquinas, reduciendo eficazmente el *makespan* y mejorando el aprovechamiento del sistema.

2.5.6 Algoritmo híbrido heurístico-genético con parámetros adaptativos para la calendarización estática de tareas

El artículo (Ding, Wu, Xie, y Zeng, 2017) propone un algoritmo híbrido heurístico–genético con parámetros adaptativos (HGAAP), el cual combina un método de calendarización heurística con un algoritmo genético. Para acelerar la convergencia del proceso evolutivo, la generación inicial se construye utilizando un algoritmo heurístico de calendarización común. Además, las probabilidades de cruce y mutación se ajustan dinámicamente durante la ejecución, con el fin de favorecer la evolución y encontrar una mejor solución. El algoritmo propuesto también incorpora un mecanismo para eliminar individuos redundantes en cada generación, preservando así la diversidad de la población y evitando la convergencia prematura.

Los resultados experimentales, obtenidos a partir de un gran conjunto de DAGs generados aleatoriamente, demuestran que HGAAP produce soluciones de calendarización superiores a las obtenidas por algoritmos de referencia, incluido HEFT (Topcuoglu, Hariri, y W., 2002), considerado uno de los métodos heurísticos más efectivos para este tipo de problemas.

2.5.7 Calendarización de tareas

El artículo (AlEbrahim y Ahmad, 2017) propone un algoritmo de calendarización que asigna las tareas, representadas en el DAG, al procesador para minimizar el tiempo total de ejecución, considerando la restricción de cruce entre procesadores. El algoritmo inicia con una fase de priorización, en la cual las tareas se ordenan según un valor de prioridad que determina su relevancia dentro del grafo y su posición en la cola de ejecución. Posteriormente, en la fase de selección de procesador, se determina cuál procesador ejecutará cada tarea de la forma más eficiente.

Para realizar esta asignación, las tareas se ponderan mediante un peso calculado a partir del tiempo de ejecución estimado para cada procesador, los costos de comunicación entre tareas dependientes y el valor de priorización heredado de la tarea anterior. En la fase de selección, el algoritmo introduce una decisión aleatoria basada en un umbral asociado al cruce entre procesadores, el cual se calcula considerando los costos de comunicación entre tareas. Este mecanismo permite evitar asignaciones subóptimas debidas a decisiones deterministas demasiado rígidas y mejora el equilibrio en la distribución de tareas.

El algoritmo fue evaluado utilizando un conjunto de 750 DAGs generados aleatoriamente. Los resultados mostraron mejoras en el *makespan* de entre un 6 % y un 7 % en comparación con los algoritmos HEFT y PEFT (Arabnejad y Barbosa, 2014), considerados referentes en la calendarización sobre procesadores heterogéneos. Cabe destacar que el algoritmo propuesto mantiene la misma complejidad computacional que estos métodos, pero logra un rendimiento significativamente superior al reducir el tiempo total de ejecución.

2.5.8 Método para construir algoritmos de calendarización de tareas

El artículo (S. I. Kim y Kim, 2019) demuestra que un calendarizador de tareas inteligente es un componente clave para mejorar tanto el rendimiento como la eficiencia energética en entornos con procesadores multi-núcleo heterogéneos. Se realizó un análisis detallado de los algoritmos de calendarización existentes y del entorno de ejecución, con el propósito de identificar los elementos fundamentales necesarios para diseñar el mejor método de calendarización para un sistema de ejemplo.

A partir de este análisis, se identificaron seis componentes esenciales: la clasificación de tareas, la asignación de procesadores, el ordenamiento de colas, la migración de tareas, el escalamiento dinámico de voltaje y frecuencia (DVFS), y la estrategia de robo de tareas (*work stealing*). Con base en estos componentes, se evaluaron múltiples combinaciones posibles para determinar cuál configuración ofrecía los mejores resultados.

Los experimentos demostraron que la combinación óptima de componentes puede mejorar significativamente el rendimiento global del sistema, tanto en términos de tiempo de ejecución como de consumo energético. Estos resultados destacan la importancia de diseñar calendarizadores adaptativos y conscientes de la arquitectura, especialmente en sistemas heterogéneos donde las diferencias entre núcleos pueden influir de manera notable en el desempeño final.

2.5.9 Resumen de los trabajos relacionados

La tabla 2.2 compara los trabajos ya mencionados, destacando las características por las que son relevantes en esta tesis:

- Aprendizaje automático: se refiere al uso de modelos o técnicas que permiten que el sistema aprenda patrones a partir de datos para tomar decisiones o mejorar su desempeño. Para este contexto, puede incluir modelos predictivos para estimar el tiempo de ejecución, seleccionar estrategias de integración o ajustar dinámicamente los parámetros de ejecución.
- Integrales multidimensionales: indica si los trabajos relacionados consideran la integración multidimensional como un problema específico dentro del proceso de calendarización.
- Codelet: se refiere a la implementación de la unidad modular encargada de agrupar y referenciar múltiples funciones codificadas, permitiendo su gestión, selección y ejecución de forma independiente dentro de arquitecturas heterogéneas.
- Ejecución secuencial: corresponde a la capacidad de ejecutar un algoritmo de manera lineal, en un único hilo o núcleo, sin recurrir a paralelismo. Esta modalidad es relevante porque proporciona una línea base para comparar optimizaciones y resulta útil en dispositivos con recursos limitados o en etapas del algoritmo que no pueden paralelizarse.

- Ejecución paralela: describe la posibilidad de distribuir el cálculo entre múltiples unidades de procesamiento para acelerar las tareas.

  - CPU: paralelismo basado en múltiples hilos.
  - GPU: miles de núcleos capaces de realizar operaciones masivas en paralelo.
  - AVX: instrucciones vectoriales que permiten procesar varios datos simultáneamente en una sola operación.

  Este enfoque resulta especialmente útil para acelerar el cálculo de integrales complejas y otras tareas numéricas de alta demanda computacional.
- Reducción de tiempo: alude a la capacidad del método, algoritmo o calendarizador para disminuir el tiempo total de ejecución en comparación con enfoques previos o versiones no optimizadas. En los trabajos relacionados, este punto destaca las estrategias que lograron mejoras significativas mediante paralelismo, optimizaciones algorítmicas o una asignación eficiente de tareas.

2.6 Estrategias de Inteligencia Artificial para calendarizadores

Como se observo anteriormente, los calendarizadores son herramientas clave para optimizar la asignación de recursos, la programación de tareas y la gestión del tiempo. Bajo este contexto, las estrategias de IA ofrecen soluciones avanzadas que superan las limitaciones de los métodos tradicionales, permitiendo sistemas más adaptativos, dinámicos y capaces de aprender de la experiencia.

En esta tesis, se describirán dos técnicas de IA que serán empleas en los calendarizadores desarrollados durante este proyecto: árbol de decisiones y búsqueda tabú.

2.6.1 Árbol de decisiones

Los árboles de decisión (Kotsiantis, 2013) son modelos de clasificación que operan mediante una secuencia estructurada de pruebas simples. Cada prueba consiste en comparar un atributo numérico con un valor umbral o verificar si un atributo categórico pertenece a un conjunto específico de valores. Este enfoque genera un proceso de decisión lógico y transparente.

A diferencia de los modelos de tipo caja negra como las redes neuronales, los árboles de decisión ofrecen una mayor comprensión. Mientras que en una red neuronal las relaciones entre nodos se representan mediante pesos numéricos difíciles de interpretar, en un árbol de decisión las reglas que guían cada clasificación son explícitas y fácilmente comprensibles. Esto facilita su análisis y validación en los sistemas, donde son implementados.

Un ejemplo claro y simple se puede observar en la Figura 2.2, donde cada ruta es una regla lógica:

Tabla 2.2: Comparación de los trabajos relacionados.

Estado del arte	Aprendizaje automático	Integrales multidimensionales	Codelet	Ejecución secuencial	Ejecución paralela			Reducción de tiempo
					CPU	GPU	AVX	
Planificador inteligente para integración numérica multidimensional en ambientes heterogéneos (Morales y Puga, 2022)	✓	✓		✓	✓	✓		✓
FlexTensor (Zheng y cols., 2020)	✓			✓	✓	✓		✓
StarPU (Augonnet y cols., 2009)			✓	✓	✓	✓		✓
Algoritmo de calendarización de tareas basado en aprendizaje por refuerzo (Song y cols., 2023)	✓			✓	✓			✓
Regla de Johnson para la calendarización de n tareas en dos máquinas (Garey y Johnson, 1976)				✓	✓			✓
Algoritmo híbrido heurístico-genético con parámetros adaptativos para la calendarización estática de tareas (Ding y cols., 2017)	✓			✓				✓
Calendarización de tareas (AlEbrahim y Ahmad, 2017)	✓			✓				✓
Método para construir algoritmos de calendarización de tareas (S. I. Kim y Kim, 2019)				✓				✓
Propuesta	✓	✓	✓	✓	✓	✓	✓	✓

- Si A es verdadero y B es verdadero → Decisión 1
- Si A es verdadero y B es falso → Decisión 2
- Si A es falso → Decisión 3

Enfoques estadísticos, como las pruebas de hipótesis y diversas técnicas de remuestreo, han evolucionado en paralelo con métodos de aprendizaje automático, lo que ha dado lugar a herramientas basadas en árboles de decisión altamente flexibles y aplicables a una amplia gama de tareas estadísticas y de aprendizaje automático. Estos métodos destacan por su capacidad para trabajar con distintos niveles de medición y con diferentes calidades de datos, mostrando una notable robustez ante la

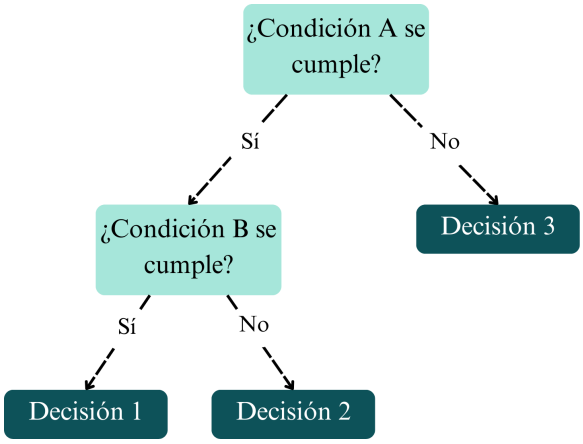


Figura 2.2: Ejemplo de un árbol de decisiones

.

presencia de valores faltantes. De hecho, muchas de sus variantes incorporan mecanismos explícitos para manejar datos incompletos en las etapas de entrenamiento y predicción (Shiue y Su, 2003).

Asimismo, múltiples estudios han demostrado que el empleo de árboles de decisión en sistemas de calendarización dinámica puede mejorar de manera significativa el rendimiento general, optimizando la asignación de tareas y reduciendo los tiempos de ejecución en arquitecturas heterogéneas (Shaw, Park, y Raman, 1992; Park, Raman, y Shaw, 1997; Arzi y Iaroslavitz, 2000; C. O. Kim, Min, y Yih, 2010).

Las principales ventajas de emplear árboles de decisión en calendarizadores dinámicos incluyen:

1. El conocimiento aprendido a partir de ejemplos de entrenamiento no sólo permite clasificar con precisión los casos previamente observados, sino que también ofrece una elevada capacidad de generalización hacia instancias no vistas, manteniendo un desempeño consistente.

2. La función inferida puede representarse mediante un único árbol de decisión o mediante un conjunto de árboles con sus respectivas reglas. Esta representación explícita facilita su integración en los mecanismos de control del calendarizador y mejora la legibilidad del modelo.

3. Los métodos de aprendizaje basados en árboles de decisión presentan una robustez notable frente a errores de clasificación en los datos de entrenamiento, así como ante posibles imprecisiones en los valores de los atributos. Esto los convierte en una opción adecuada en entornos donde la calidad de los datos puede variar.

4. Estos métodos mantienen un rendimiento adecuado incluso cuando algunos atributos contienen valores faltantes o desconocidos, gracias a que incorporan es-

trategias específicas para manejar de manera estable la incompletitud de la información.

2.6.2 Búsqueda tabú

En 1986, Fred Glover introdujo un enfoque innovador para la optimización combinatoria denominado búsqueda tabú (Gendreau y Potvin, 2005). Esta técnica se basa en ampliar la exploración más allá de los óptimos locales, permitiendo realizar movimientos que no necesariamente mejoran la solución actual. Para evitar retroceder hacia combinaciones ya visitadas, la búsqueda emplea estructuras de memoria, conocidas como listas tabú, que registran los movimientos o soluciones recientes y restringen su reutilización durante un número determinado de iteraciones.

La búsqueda tabú puede entenderse como una evolución de los métodos clásicos de búsqueda local, comúnmente utilizados para encontrar soluciones óptimas en problemas complejos de combinatoria. Al igual que estos métodos, la búsqueda tabú recorre el espacio de soluciones avanzando en cada iteración hacia una solución vecina, definida por el operador de vecindad, el cual es un conjunto de transformaciones permitidas sobre la solución actual. La selección de la siguiente solución puede realizarse bajo el criterio de mejor mejora (elegir la mejor entre todas las soluciones vecinas) o de primera mejora (tomar la primera solución vecina que mejora la función objetivo).

Los métodos tradicionales dependen de una mejora continua de la función objetivo para guiar la búsqueda, lo que provoca que con frecuencia queden atrapados en óptimos locales. La principal aportación de la búsqueda tabú es su capacidad para escapar de estos óptimos: cuando el algoritmo detecta que ha alcanzado uno, se permite avanzar hacia la mejor solución no tabú en la vecindad, incluso si dicha solución es peor que la actual. Para evitar ciclos, se mantiene una memoria a corto plazo que marca ciertos movimientos o soluciones como tabú (prohibidos), a menos que se cumplan condiciones especiales conocidas como criterios de aspiración, que permiten ignorar la restricción si el movimiento lleva a una solución excepcionalmente buena.

Dado que este proceso puede continuar indefinidamente, las implementaciones prácticas requieren criterios explícitos de terminación. Entre los más comunes se encuentran: limitar el tiempo de ejecución total, fijar un número máximo de iteraciones o detener la búsqueda después de un número determinado de iteraciones sin obtener mejoras en el mejor valor objetivo registrado.

Un ejemplo claro y simple se puede observar en la Figura 2.3, en esta se muestra un conjunto de soluciones posibles (los nodos S1-–S5) conectadas mediante aristas que representan los movimientos permitidos entre soluciones, cada uno con un "costo", que puede interpretarse como: el costo de pasar de una solución a otra, la calidad relativa de la solución vecina o la penalización asociada al movimiento.

La búsqueda tabú comenzaría en un nodo (por ejemplo S1), luego exploraría sus vecinos (S2 y S3), elegiría uno aun si no mejora la solución, y prohibiría volver a un nodo recientemente visitado mediante la lista tabú.

La idea fundamental de utilizar información previa para guiar la búsqueda en

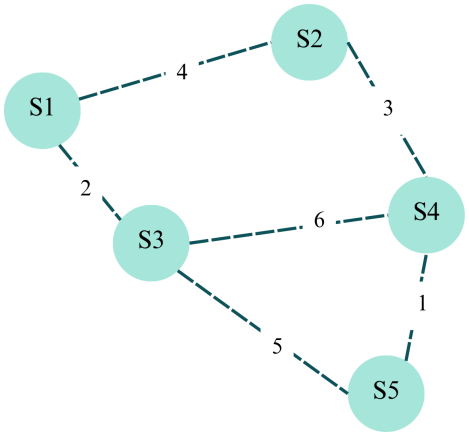


Figura 2.3: Ejemplo de la búsqueda tabú

.

procesos de optimización se relaciona con los métodos desarrollados en la década de 1970 dentro del campo de la Inteligencia Artificial (Nilsson, 1933). Es importante mencionar que Hansen propuso, en 1986, un enfoque conceptual cercano a la búsqueda tabú, conocido como "ascenso más pronunciado/descenso más suave" (Hansen, 1986), el cual también empleaba estrategias para escapar de óptimos locales.

Glover, por su parte, no concebía la búsqueda tabú como una heurística aislada, sino como una metaheurística: un marco general diseñado para supervisar y dirigir heurísticas internas, ajustándose de manera flexible a las características particulares del problema a resolver.

En este proyecto, se propone emplear la búsqueda tabú como un componente clave dentro del calendarizador con el objetivo de seleccionar el módulo de ejecución más adecuado para cada punto de la integración multidimensional. Esta elección se realizará de acuerdo a la cantidad de dimensiones y el número de puntos a evaluar.

Capítulo 3

Implementación

Este capítulo presenta las principales extensiones desarrolladas sobre la biblioteca de integrales multidimensionales, enfocadas en mejorar el rendimiento mediante el uso de arquitecturas heterogéneas y técnicas de optimización.

En la primera sección, se describe la biblioteca de integrales multidimensionales, los módulos de ejecución que la componen y cómo se planea expandirla. En la segunda sección, se presenta la arquitectura del sistema de calendarización propuesto, se describe de manera general cada elemento involucrado y como interactúan entre sí.

La tercera sección detalla la implementación de un nuevo módulo de ejecución para la biblioteca, el cual aprovecha las instrucciones vectoriales AVX. Este módulo permite procesar múltiples puntos de integración de manera simultánea dentro de cada hilo de la CPU, incrementando la eficiencia computacional en plataformas modernas con soporte SIMD.

En la cuarta sección, se introduce la unidad para administrar los módulos de ejecución denominada *codelet*. Funciona como un arreglo de apuntadores a funciones que encapsula los módulos (secuencial, CUDA, OpenMP y AVX), permitiendo su invocación estática o dinámica, independiente de la estrategia de calendarización y los detalles específicos de implementación de cada módulo.

Finalmente, se implementan tres estrategias de calendarización: estática secuencial, dinámica secuencial y estática concurrente; las cuales permiten seleccionar el módulo de ejecución más adecuado para evaluar cada punto de la integral multidimensional, con el objetivo de minimizar el tiempo total de ejecución.

Estas contribuciones permite transformar la biblioteca base en un sistema flexible, escalable y optimizado para su ejecución en plataformas heterogéneas.

3.1 Introducción a la implementación propuesta

Este trabajo de tesis toma como punto de partida la biblioteca de integrales multidimensionales (Quintero-Monsebaiz y cols., 2021), la cuál está basada en reglas de cuadratura de Gauss-Kronrod, junto con extrapolaciones de Romberg. Lo que permite

obtener resultados de alta precisión de manera eficiente, al aprovechar los componentes de la plataforma donde se ejecute.

La biblioteca está compuesta por tres módulos de ejecución principales:

- Módulo de ejecución secuencial (SEC): ejecuta la integración punto por punto de forma secuencial, utilizando únicamente un hilo de la CPU. Sirve de referencia para comparaciones de rendimiento con los otros módulos.
- Módulo de ejecución paralela con OpenMP (OMP): permite dividir el trabajo entre múltiples hilos de la CPU, reduciendo el tiempo de ejecución en procesadores multinúcleo.
- Módulo de ejecución paralela con GPU (CUDA): aprovecha la capacidad de procesamiento masivamente paralelo de las tarjetas gráficas para calcular múltiples puntos simultáneamente.

Cada módulo está diseñado para recibir como entrada el número de puntos y el número de dimensiones de la integral ($3 \leqslant dimensiones \leqslant 6$). Como salida, devuelve el valor estimado de la integral junto con el tiempo de ejecución. Además, el diseño modular de la biblioteca facilita su extensión, permitiendo agregar nuevos módulos con diferentes tecnologías de paralelización.

El objetivo de este capítulo es expandir la funcionalidad de dicha biblioteca, mediante la incorporación de tres elementos clave:

1. Módulo de ejecución con instrucciones vectorizadas AVX.
2. *Codelet*.
3. Calendarizadores.

La integración de estos nuevos elementos permite transformar la biblioteca original en una herramienta más versátil, escalable y adaptada a las plataformas heterogéneas. Donde la correcta asignación de tareas entre CPU y GPU puede tener un impacto significativo en el rendimiento final del sistema.

3.2 Arquitectura del sistema de calendarización

Dado que la biblioteca de integrales multidimensionales emplea la cuadratura de Gauss-Kronrod, la integral se evalúa a partir de un conjunto de puntos independientes. Cada punto corresponde a una evaluación de la función integrando $f(x)$, lo que permite enviarlos de manera individual a los distintos módulos de ejecución disponibles, como se observa en la Figura 3.1.

Sin embargo, no todos los puntos presentan el mismo costo computacional, ya que la complejidad de la evaluación depende del comportamiento local del integrando. Asimismo, cada módulo de ejecución (SEC, AVX, OMP, CUDA) exhibe un rendimiento diferente dependiendo del tipo de carga que recibe.

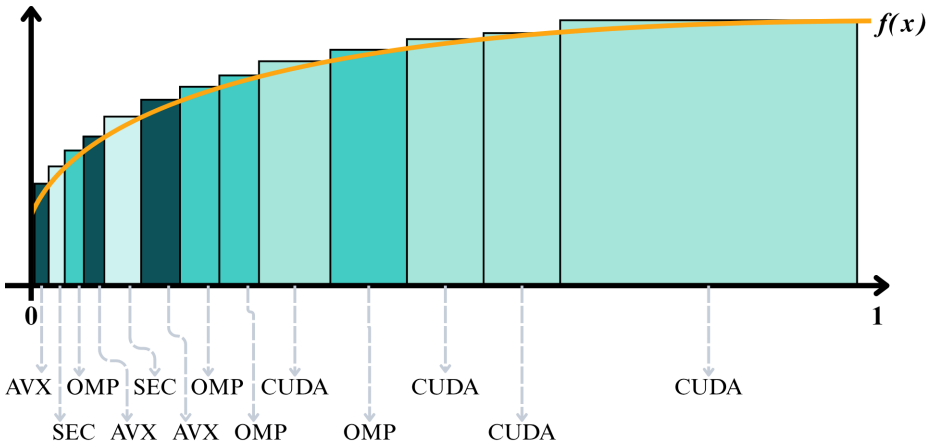


Figura 3.1: Integración por puntos en los módulos de ejecución.

Por esta razón, se implementó el uso de una estrategia de caracterización que permite seleccionar el módulo de ejecución más adecuado para cada punto. El objetivo es asignar cada punto al módulo que maximice su rendimiento, optimizando así el uso de recursos heterogéneos y mejorando la eficiencia global del proceso de integración multidimensional.

Para ello, se diseño el sistema de calendarización con una arquitectura modular que permite coordinar la ejecución de los módulos de la biblioteca de integrales multidimensionales, en función de su rendimiento, usando diferentes estrategias de calendarización. Esta integración se basa en la interacción entre tres bloques principales: el calendarizador, el *codelet* y la biblioteca.

El sistema recibe como parámetros de entrada:

- El nombre de la plataforma en la que se ejecutará el cálculo.
- La función a integrar $f(x)$, junto con sus argumentos (especificados en la sección 3.4.1).
- El número de dimensiones.
- El número de puntos de integración.

A partir estos parámetros, el calendarizador accede a las tablas de tiempos de ejecución correspondientes a la plataforma seleccionada. Dichas tablas contienen, para cada número de puntos, los tiempos estimados de ejecución de los módulos secuencial, CUDA, OpenMP y AVX (véase Sección 4.3.1). Posteriormente, el calendarizador identifica la tabla asociada a la dimensión de la integral y determina el módulo de ejecución más eficiente para cada punto. Finalmente, genera un identificador del módulo seleccionado y lo envía al codelet para su ejecución.

El codelet recibe el identificador generado por el calendarizador, junto con los datos de entrada del sistema de calendarización: número de puntos, dimensiones y función a integrar. Después, invoca directamente el módulo de ejecución solicitado, el cual se encarga de calcular la integral sobre la subregión asignada. Este ciclo se repite

para cada punto de integración. El resultado parcial producido por cada módulo es acumulado de manera controlada hasta obtener el resultado final de la integral.

El sistema entrega como salida el resultado de la integral y el tiempo de ejecución. En la Figura 3.2, el resultado se simboliza con la imagen de un gato realista, utilizada únicamente como ejemplo de una salida precisa, al ser el resultado de una integral multidimensional.

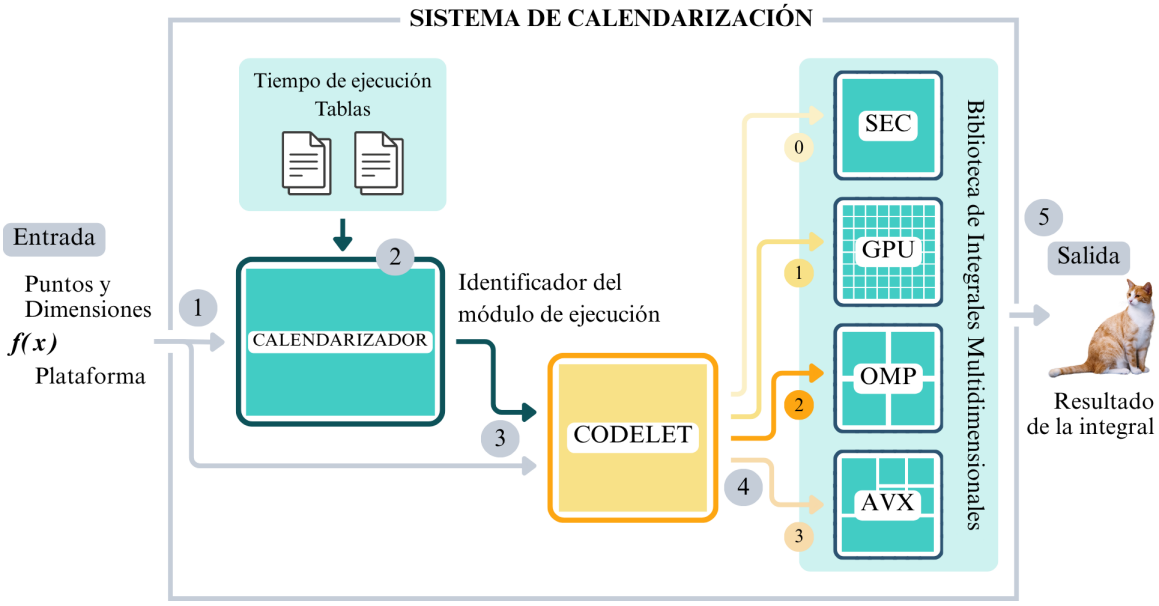


Figura 3.2: Arquitectura del sistema de calendarización.

En síntesis, el flujo de operación del sistema puede describirse de la siguiente manera:

1. Se proporcionan los parámetros de entrada: puntos, dimensiones, función y plataforma.
2. El calendarizador accede a las tablas de tiempo y selecciona el módulo de ejecución más eficiente para cada punto.
3. Los identificadores de los módulos y los argumentos de entrada (número de puntos, dimensiones y función a integrar) se envía al *codelet*.
4. Para cada punto, el *codelet* invoca el módulo de ejecución correspondiente dentro de la biblioteca y le pasa los argumentos necesarios.
5. Los módulos ejecutan el cálculo y retornan el resultado de la integral.

Esta estructura flexible permite que diferentes subregiones del dominio puedan ser evaluadas por distintos módulos, maximizando así el rendimiento general. El calendarizador, al ser un componente modular puede ser cambiado con facilidad, seleccionado el calendarizador (estático secuencial, dinámico secuencial, estático concurrente) que mas se adapte al entorno del problema.

3.2.1 Detalles de la implementación del sistema de calendarización

El sistema de calendarización fue implementado en el lenguaje de programación C, debido a su eficiencia en el manejo de memoria, control de bajo nivel y compatibilidad con bibliotecas de paralelismo y vectorización.

Para habilitar la ejecución paralela y vectorizada en los módulos y calendarizadores, se emplearon las siguientes bibliotecas estándar:

- `omp.h`: utilizada para la creación y gestión de regiones paralelas mediante la interfaz de programación OpenMP. Esta biblioteca permite distribuir las tareas de integración entre múltiples hilos de ejecución en CPU, optimizando el rendimiento en sistemas multinúcleo.
- `immintrin.h`: empleada para el uso de instrucciones vectoriales AVX. Permite realizar operaciones aritméticas sobre varios datos de manera simultánea, reduciendo el tiempo de ejecución.

De esta forma, el sistema logra aprovechar las capacidades de procesamiento paralelo y vectorizado disponibles en las plataformas heterogéneas, garantizando una ejecución eficiente y portable.

3.3 Módulo de ejecución AVX

El módulo de ejecución AVX calcula la integración multidimensional mediante la vectorización con instrucciones AVX. Este módulo fue desarrollado como un complemento para la biblioteca de integrales multidimensionales (Figura 3.3). La incorporación del módulo AVX responde a la necesidad de explotar el paralelismo a nivel de datos (SIMD), disponible en las arquitecturas modernas de las CPUs. Se busca una mejora significativa en el rendimiento sin recurrir a múltiples hilos, ni a componentes de ejecución diferentes a la CPU. Esta integración fortalece el conjunto de herramientas disponibles en la biblioteca, al ofrecer una opción adicional que aprovecha capacidades específicas del *hardware*.

Para desarrollar este módulo, se tomó como base el módulo de ejecución secuencial de la biblioteca de integrales multidimensionales y se modificó para optimizarlo mediante las instrucciones AVX. Esto permite procesar múltiples evaluaciones de la función en paralelo a nivel de registro.

El diseño del módulo considera las siguientes características:

- Flexibilidad para evaluar distintas funciones.
- Capacidad de integrarse con diversos calendarizadores sin alterar su funcionamiento interno.
- Aceleración mediante instrucciones SIMD para reducir el tiempo de ejecución sin comprometer la precisión numérica.

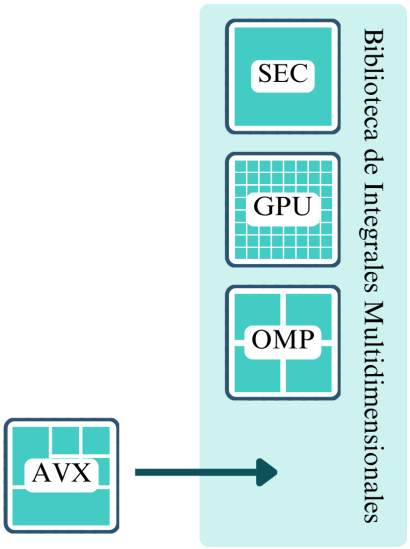


Figura 3.3: Incorporación del módulo de ejecución AVX a la biblioteca de integrales multidimensionales.

Este módulo representa un ejemplo claro de cómo las técnicas de optimización a nivel de *hardware* pueden ser aplicadas a algoritmos numéricos clásicos para mejorar su desempeño en escenarios de alto costo computacional.

3.3.1 Arquitectura del módulo de ejecución AVX

Para comprender la arquitectura del módulo de ejecución AVX, primero es necesario explicar el funcionamiento del módulo secuencial, el cual es la base. Este módulo se diseñó como una estructura genérica capaz de evaluar integrales multidimensionales mediante el método de cuadratura de Gauss–Kronrod. Su funcionamiento consiste en recorrer todos los puntos de integración en cada dimensión, multiplicando el valor de la función evaluada en dichos puntos por los pesos correspondientes de la cuadratura.

El módulo está conformado por cuatro funciones: `SEC3D`, `SEC4D`, `SEC5D` y `SEC6D`, cada una corresponde a una dimensión fija DIM (donde $3 \leq DIM \leq 6$). Estas funciones están implementadas mediante bucles `for` anidados, donde cada bucle representa una dimensión de la integral. Para realizar la integración, se recorren los índices de todos los bucles (Tabla 3.1); al llegar al bucle más interno, se evalúa la función para cada punto y se acumula el resultado, multiplicado por los pesos correspondientes[1].

La estructura general de cada función es la siguiente: `SEC3D` corresponde a tres dimensiones y, por lo tanto, está compuesta por tres bucles anidados (Figura 3.4a); `SEC4D` corresponde a cuatro dimensiones y contiene cuatro bucles (Figura 3.4b); `SEC5D` corresponde a cinco dimensiones y contiene cinco bucles (Figura 3.4c); y `SEC6D` co-

[1]Los pesos de cuadratura unidimensional se utilizan para aproximar el valor de la integral; su producto entre dimensiones pondera cada término de la suma y permite convertir la suma discreta en una aproximación de la integral continua.

rresponde a seis dimensiones, por lo que incluye seis bucles (Figura 3.4d).

Tabla 3.1: Relación de dimensiones (bucles) e índices.

Dimensiones	Índices
1	i
2	j
3	k
4	l
5	m
6	o

```
1  for i do
   |  // Primera dimensión
2  |  for j do
   |  |  // Segunda dimensión
3  |  |  for k do
   |  |  |  // Tercera dimensión
4  |  |  |  Evaluación de la integral
5  |  |  end
6  |  end
7  end
```

(a) SEC3D

```
1  for i do
   |  // Primera dimensión
2  |  for j do
   |  |  // Segunda dimensión
3  |  |  for k do
   |  |  |  // Tercera dimensión
4  |  |  |  for l do
   |  |  |  |  // Cuarta dimensión
5  |  |  |  |  Evaluación de la integral
6  |  |  |  end
7  |  |  end
8  |  end
9  end
```

(b) SEC4D

```
1  for i do
   |  // Primera dimensión
2  |  for j do
   |  |  // Segunda dimensión
3  |  |  for k do
   |  |  |  // Tercera dimensión
4  |  |  |  for l do
   |  |  |  |  // Cuarta dimensión
5  |  |  |  |  for m do
   |  |  |  |  |  // Quinta dimensión
6  |  |  |  |  |  Evaluación de la integral
7  |  |  |  |  end
8  |  |  |  end
9  |  |  end
10 |  end
11 end
```

(c) SEC5D

```
1  for i do
   |  // Primera dimensión
2  |  for j do
   |  |  // Segunda dimensión
3  |  |  for k do
   |  |  |  // Tercera dimensión
4  |  |  |  for l do
   |  |  |  |  // Cuarta dimensión
5  |  |  |  |  for m do
   |  |  |  |  |  // Quinta dimensión
6  |  |  |  |  |  for o do
   |  |  |  |  |  |  // Sexta dimensión
7  |  |  |  |  |  |  Evaluación de la integral
8  |  |  |  |  |  end
9  |  |  |  |  end
10 |  |  |  end
11 |  |  end
12 |  end
13 end
```

(d) SEC6D

Figura 3.4: Funciones del módulo de ejecución secuencial.

El algoritmo del módulo secuencial recibe como entrada:

- `x[]`: arreglo de puntos unidimensionales utilizados por la cuadratura.
- `w[]`: arreglo de pesos asociados a los puntos, uno por cada punto.
- `n`: número de puntos de integración.
- `*function`: apuntador a la función a integrar.

Dado que todas las funciones siguen el mismo esquema, variando únicamente en el número de dimensiones y, por ende, en la profundidad de los bucles, se tomó como

ejemplo la función `SEC6D` para explicar la implementación del módulo de ejecución secuencial. El Algoritmo 1 muestra la estructura general de dicha función.

Algoritmo 1: Cálculo de integrales multidimensionales usando el módulo de ejecución secuencial - 6 Dimensiones

Entrada: arreglo de puntos unidimensional `x[]`, arreglo de pesos `w[]`, número de puntos `n` y apuntador a función `*function`
Salida: valor aproximado de la integral `sum`

```
1  Inicializar sum ← 0;
   // Cada bucle interno corresponde a una dimensión adicional (máx. 6D)
2  for i = 0 to n − 1 do
       // Primera dimensión
3      w_i ← w[i];   ii[0] ← i;
4      for j = 0 to n − 1 do
           // Segunda dimensión
5          w_j ← w[j];   ii[1] ← j;
6          for k = 0 to n − 1 do
               // Tercera dimensión
7              w_k ← w[k];   ii[2] ← k;
8              for l = 0 to n − 1 do
                   // Cuarta dimensión
9                  w_l ← w[l];   ii[3] ← l;
10                 for m = 0 to n − 1 do
                       // Quinta dimensión
11                     w_m ← w[m];   ii[4] ← m;
12                     for o = 0 to n − 1 do
                           // Sexta dimensión
13                         w_o ← w[o];   ii[5] ← o;
14                         Evaluar func ← (*function)(x, ii, DIM);
15                         sum ← sum + func · w_i · w_j · w_k · w_l · w_m · w_o;
16                     end
17                 end
18             end
19         end
20     end
21 end
22 return sum;
```

El bucle más interno es la parte central del algoritmo (líneas 12–16), ya que en él se evalúa la función `function(x, i, j, k, l, m, o, DIM)` para obtener el valor del integrando $f(\mathbf{x}_{i,j,k,l,m,o})$, correspondiente al punto cuyas coordenadas están determinadas por los índices i, j, k, l, m, o, los cuales representan una posición dentro de la malla multidimensional. Una vez obtenido este valor, se multiplica por el producto

de los pesos asociados y el resultado se acumula en la variable `sum`.

Al finalizar todos los bucles, la función devuelve el valor de `sum` como aproximación de la integral en la región considerada. Dependiendo del número de dimensiones, el número de índices y pesos involucrados varía de manera correspondiente.

En el Algoritmo 1:

- `wi, wj, wk, wD`: representan los pesos de cuadratura unidimensional.
- `DIM`: representa el número de dimensiones.
- `ii[]`: representa el vector de coordenadas del punto de integración dentro de la malla multidimensional.

El módulo secuencial constituye la base conceptual y funcional sobre la que se construyó el módulo de ejecución AVX. A partir de su estructura, se implementaron optimizaciones que permiten aprovechar el paralelismo a nivel de datos sin modificar la lógica de cálculo original.

Detalles de la implementación del módulo de ejecución AVX

La paralelización a nivel de datos, en el módulo AVX, se realizó utilizando registros vectoriales de 256 bits, los cuales permiten trabajar con vectores de cuatro números de punto flotante de doble precisión (*double*). Esto significa que se puede realizar una misma operación aritmética sobre cuatro datos *double* en un sólo paso de la CPU.

La funciones del módulo de ejecución secuencial (`SEC3D, SEC4D, SEC5D, SEC6D`) se reorganizaron para operar en bloques de cuatro puntos simultáneamente, aprovechando los registros vectoriales de 256 bits (`__m256d`).

Para explicar este proceso, nuevamente se tomó como ejemplo la función `SEC6D` (Algoritmo 2). La entrada y la salida de la función permanecen sin cambios respecto al módulo de ejecución secuencial.

El bucle más interno, al ser la parte central del algoritmo, fue el que se modificó (líneas 14–25). Dicho bucle recorre los puntos en bloques de cuatro, dentro de él se incorpora un nuevo bucle que itera sobre esos cuatro puntos de manera secuencial para evaluar la función `(*function)()` de forma escalar (líneas 15–18), almacenando cada resultado en el arreglo `func[p]`.

Luego, estos valores, junto con los pesos o, se cargan en registros vectoriales mediante la instrucción `_mm256_loadu_pd` (líneas 19 y 20).

A continuación, se calcula el producto total de los pesos asociados a los bucles exteriores (`wprod` ← `wi` · `wj` · `wk` · `wl` · `wm`) y se construye un vector con cuatro réplicas de este valor utilizando la instrucción `_mm256_set1_pd` (línea 21). Dicho vector se multiplica elemento por elemento por el vector de pesos o mediante la instrucción `_mm256_mul_pd`, obteniendo así los pesos completos correspondientes al bloque procesado (línea 22).

```
w_avx ← _mm256_mul_pd(wk_avx, _mm256_set1_pd(wprod))
```

Algoritmo 2: Cálculo de integrales multidimensionales usando el módulo de ejecución AVX - 6 Dimensiones

Entrada: arreglo de puntos unidimensional `x[]`, arreglo de pesos `w[]`, número de puntos `n` y apuntador a función `*function`
Salida: valor aproximado de la integral `sum`

```
1  Inicializar sum ← 0;
2  aligned ← floor(n/4) · 4 ;                    /* Máximo múltiplo de 4 para vectorización */
   // Cada bucle interno corresponde a una dimensión adicional (máx. 6D)
3  for i = 0 to n − 1 do
       // Primera dimensión
4      w_i ← w[i];   ii[0] ← i
5      for j = 0 to n − 1 do
           // Segunda dimensión
6          w_j ← w[j];   ii[1] ← j
7          for k = 0 to n − 1 do
               // Tercera dimensión
8              w_k ← w[k];   ii[2] ← k
9              for l = 0 to n − 1 do
                   // Cuarta dimensión
10                 w_l ← w[l];   ii[3] ← l
11                 for m = 0 to n − 1 do
                       // Quinta dimensión
12                     w_m ← w[m];   ii[4] ← m
13                     Inicializar sum_avx[] ← 0 ;        /* Registro vectorial acumulador */
14                     for o = 0 to aligned − 1 step 4 do
                           // Evaluar escalarmente 4 puntos consecutivos de la integral
15                         for p ← 0 to 3 do
16                             ii[5] ← o + p;
17                             Evaluar func[p] ← (∗function)(x, ii, DIM);
18                         end
                           // Cargar bloques en registros vectoriales
19                         func_avx ← _mm256_loadu_pd(func) ;   /* Cargar los cuatro
                            valores del integrando */
20                         w_o_avx ← _mm256_loadu_pd(w[o : o + 3]) ; /* Carga los cuatro
                            pesos de o */
                           // Multiplicar y acumular en vector
21                         wprod ← w_i · w_j · w_k · w_l · w_m · w_o ; /* Producto de pesos exteriores
                            */
22                         w_avx ← _mm256_mul_pd(w_o_avx, _mm256_set1_pd(wprod)) ;
                           /* Multiplica y carga wprod · wo_avx */
23                         prod_avx ← _mm256_mul_pd(func_avx, w_avx)
24                         sum_avx ← _mm256_add_pd(sum_avx, prod_avx)
25                     end
                       // Reducir acumulador vectorial a escalar
26                     for p ← 0 to 3 do
27                         sum ← sum + sum_avx[p];
28                     end
29                 end
30             end
31         end
32     end
33 end
34 return sum;
```

Posteriormente, se realiza la multiplicación entre el vector que almacena los valores evaluados de la función y el vector de pesos (línea 23).

```
prod_avx ← _mm256_mul_pd(func_avx, w_avx)
```

Finalmente, los resultados ponderados se acumulan en la variable `sum`, representando la suma parcial de los valores obtenidos (líneas 24–28).

Las otras tres funciones (`SEC3D, SEC4D, SEC5D`) siguen exactamente el mismo procedimiento, variando únicamente en la profundidad de los bucles y en la cantidad de dimensiones que procesan.

En el Algoritmo 2:

- `wi, wj, wk, wl, wm, wo`: representan los pesos de cuadratura unidimensional.
- `DIM`: representa el número de dimensiones.
- `n`: representa el número de puntos de integración.
- `ii[]`: representa el vector de coordenadas del punto de integración dentro de la malla multidimensional.
- `wprod`: representa el producto parcial de pesos exteriores.
- `sum`: representa el acumulador escalar final del resultado.
- `func[4]`: representa el búfer temporal que guarda cuatro evaluaciones escalares consecutivas de `function`.
- `aligned`: representa el mayor múltiplo de 4 menor o igual que `n`; determina hasta dónde se puede vectorizar.
- `wk_avx, func_avx, w_avx, sum_avx`: representa los registros `__m256d` usados para cargas y cómputo vectorial.

3.4 Codelet

El *codelet* implementado en esta tesis actúa como una unidad de ejecución modular, permitiendo la selección estática o dinámica del módulo de ejecución a utilizar durante el proceso de calendarización. Este enfoque favorece la reutilización del código, la extensibilidad del sistema de calendarización y la independencia entre la lógica de calendarización y la biblioteca de integrales multidimensionales.

En términos funcionales, el *codelet* actúa como un vector de apuntadores a funciones (Figura 3.5), donde cada función representa una estrategia distinta (módulos de ejecución) para calcular integrales multidimensionales. De esta manera, el calendarizador no requiere conocer los detalles internos de cada implementación; simplemente envía un identificador numérico que representa el módulo mas adecuado para cierto punto de integración y el *codelet* se encarga de redirigir la ejecución al módulo correspondiente.

Esta arquitectura facilita la integración con múltiples tecnologías, desde un enfoque secuencial básico hasta implementaciones optimizadas en CUDA, OpenMP y AVX, sin alterar la estructura de la biblioteca, ni los calendarizadores.

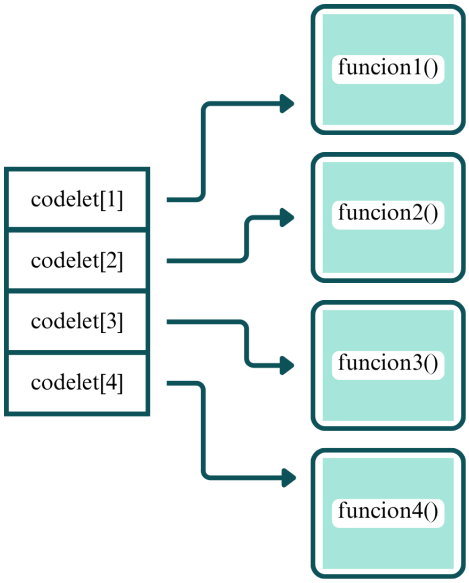


Figura 3.5: *Codelet* general.

3.4.1 Arquitectura del codelet

Como se explicó previamente, el *codelet* fue implementado como un vector de apuntadores a funciones, donde cada función representa un módulo de ejecución (Figura 3.6). Todos los módulos comparten los mismos argumentos y reciben los parámetros necesarios para realizar la integración sobre una región específica del dominio multidimensional. Al ser invocado el módulo ejecuta el procedimiento correspondiente y devuelven como salida el valor numérico resultante de la integración.

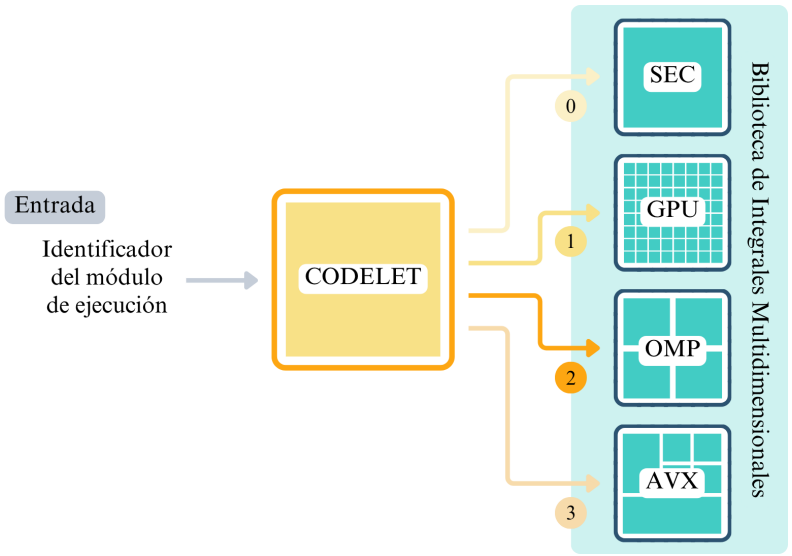


Figura 3.6: *Codelet* del sistema de calendarización propuesto.

Para ello, se definió el *codelet* como un tipo de dato mediante la instrucción:

```
typedef double (*codelet)(double*,int,int,int,double,double,double);
```

Esta declaración especifica que cada elemento de tipo `codelet` es un apuntador a un módulo que devuelve un valor de tipo `double` (el cual almacena el tiempo de ejecución) y recibe los siguientes argumentos:

- `double *`: apuntador a una variable donde se almacenará el resultado parcial de la integral.
- `int`: identificador de la función a integrar (`func_id`) dentro del conjunto de pruebas del *benchmark* (F1, F2, F3, F4 o F5).
- `int`: número de dimensiones (`DIM`).
- `int`: número de puntos de integración (`POINTS`).
- `double`: límite inferior del dominio de integración (`X1`).
- `double`: límite superior del dominio de integración (`X2`).
- `double`: tolerancia de error permitida (`ERR`).

En la función principal del sistema de calendarización, se declara el arreglo `GL[]` que contiene las referencias a las funciones correspondientes a cada módulo de ejecución. Cada posición del arreglo representa un módulo de ejecución.

```
codelet GL[] ← GL_SEC, GL_CUDA, GL_OMP, GL_AVX;
```

El *codelet* selecciona el módulo a ejecutar, a través de un identificador (Tabla 3.2).

Tabla 3.2: Asignación de módulos en el codelet.

Identificador	Módulo de ejecución
0	Secuencial
1	CUDA
2	OpenMP
3	AVX

Durante la ejecución del sistema de calendarización, el calendarizador se encarga de seleccionar el módulo más adecuado para cada punto de integración y lo almacena en la variable `mod_id` (identificador). Al invocar al *codelet*, el calendarizador le pasa el identificador como argumento, permitiendo que el sistema acceda al apuntador correspondiente dentro del vector de funciones y redirija automáticamente la ejecución al módulo seleccionado, mediante:

```
GL[mod_id](&sum, func_id, DIM, POINTS, X1, X2, ERR);
```

En esta llamada:

- `mod_id`: indica el módulo de ejecución seleccionado.

- `&sum`: es un apuntador donde se almacena el resultado parcial de la integral en una subregión.
- `func_id`: identifica la función del *benchmark* a integrar.
- `DIM`, `POINTS`, `X1`, `X2` y `ERR`: proporcionan los parámetros necesarios para el cálculo de la integral.

En resumen, el *codelet* funciona como un puente abstracto entre los calendarizadores y los módulos. Además, permite una ejecución flexible, limpia y eficiente de los módulos implementados. Este enfoque ofrece algunas ventajas:

- Modularidad: cada implementación puede desarrollarse y probarse de manera independiente.
- Extensibilidad: agregar un nuevo módulo sólo requiere incluir su apuntador y definir su lógica interna.
- Simplicidad de uso: el calendarizador delega completamente la ejecución sin preocuparse por los detalles del entorno heterogéneo.

3.5 Calendarizadores

El problema de la calendarización de tareas, aplicado en este contexto, consiste en decidir cómo asignar en qué módulo se evaluará cada punto o par de puntos de la integral, buscando minimizar el tiempo total de ejecución sin comprometer la precisión numérica.

Cuando se dispone de plataformas heterogéneas el tiempo de ejecución de la misma tarea puede variar significativamente de un componente a otro. Un calendarizador eficiente es clave para aprovechar al máximo los recursos disponibles, equilibrar la carga de trabajo y garantizar resultados confiables en el menor tiempo posible.

En esta tesis se implementaron y compararon tres calendarizadores para calcular las integrales multidimensionales, y se propone un cuarto enfoque como trabajo futuro:

- Calendarizador estático secuencial: asigna de forma fija el módulo más adecuado para cada punto de integración en función de datos de rendimiento previamente obtenidos.
- Calendarizador dinámico secuencial: toma decisiones en tiempo de ejecución, lo que permite adaptarse a condiciones cambiantes del sistema. Esto mejora la utilización de los recursos frente al estático, aunque con un coste adicional de gestión.
- Calendarizador estático concurrente: extiende el modelo secuencial estático considerando la evaluación por pares de los puntos de integración y buscando combinaciones óptimas en distintas plataformas para aprovechar la paralelización entre módulos heterogéneos.

- Calendarizador dinámico concurrente: combina la asignación dinámica y la ejecución concurrente. La idea es ajustar la planificación en tiempo de ejecución mientras se coordinan varios grupos de puntos de la integral en paralelo sobre diferentes dispositivos, con el fin de maximizar el rendimiento en escenarios heterogéneos más complejos. Sin embargo, en este trabajo de tesis no se aborda este calendarizador, sólo se propone como un trabajo a futuro.

Esto permite analizar cómo varía el desempeño y la eficiencia al aplicar distintas estrategias de calendarización sobre un conjunto de integrales multidimensionales y, al mismo tiempo, sentar las bases para un modelo más avanzado que podría explorarse en un proyecto de doctorado.

3.5.1 Estático secuencial

El calendarizador estático secuencial fue diseñado como la estrategia más simple y directa para asignar tareas de integración en el sistema de calendarización. Su funcionamiento se basa en una política de planificación, en la que se determina de antemano qué módulo de ejecución (secuencial, CUDA, OpenMP o AVX) es el más eficiente para evaluar cada punto de integración, basándose en los tiempos de ejecución de cada módulo que fueron previamente recolectados y almacenados en tablas, cada una de estas tablas presenta los tiempos de ejecución correspondientes a cada punto dentro del intervalo [1, 40]. La sección 4.3.1 aborda con mas detalle estas tablas.

El calendarizador recibe como entrada los siguientes argumentos:

- `PLAT`: nombre de la plataforma.
- `DIM`: número de dimensiones.
- `n`: número de puntos de integración.

Durante la fase de inicialización, el algoritmo del calendarizador selecciona la tabla que corresponde a la plataforma y al número de dimensiones que recibió como entrada, cada fila contiene los tiempos correspondientes a los distintos módulos de ejecución para un número determinado de puntos. El algoritmo compara estos valores y almacena el identificador del módulo, que reporta el menor tiempo de ejecución en cada caso, en el arreglo `array`.

El tamaño del arreglo es igual al número total de puntos (en este caso, 40). Cada posición de este arreglo representa un punto de integración específico y su contenido corresponde al identificador del módulo que debe ejecutar el cálculo de la integral en dicho punto. Esta estructura permite que, durante la ejecución, el sistema de calendarización consulte directamente el arreglo para invocar el módulo más eficiente sin realizar comparaciones adicionales en tiempo de ejecución.

El Algoritmo 3 ilustra el proceso principal de esta planificación. Donde:

- `n`: representa el número de puntos de integración.
- `m`: representa el número de módulos de ejecución disponibles (secuencial, CUDA, OpenMP y AVX).

- `array[n]`: almacena el identificador del módulo que reporta el menor tiempo de ejecución para cada punto.

Algoritmo 3: Selección del módulo más adecuado por número de puntos

Entrada: nombre de la plataforma `PLAT`, número de dimensiones `DIM`, número de puntos de integración `n`
Salida: arreglo con los identificadores de los módulos más adecuados por punto `array[]`

```
1  Construir el nombre del archivo: f_name ← PLAT/DIM.txt;
2  Leer el archivo f_time;
3  Reservar memoria para array de tamaño n;
4  for i ← 0 to n − 1 do
5  |  k ← 0;
6  |  Leer el primer valor de la fila y asignarlo a m_val;
7  |  for j ← 1 to m − 1 do
8  |  |  Leer el siguiente valor de la fila y asignarlo a val;
9  |  |  if val < m_val then
10 |  |  |  m_val ← val;
11 |  |  |  k ← j;
12 |  |  end
13 |  end
14 |  array[i] ← k;
15 end
16 Cerrar el archivo f_time;
```

Evaluación en pares de la integral multidimensional usando el calendarizador estático concurrente

Una vez completada esta etapa, la función principal del sistema de calendarización inicia el proceso de evaluación de la integral multidimensional (Algoritmo 4). Para ello, se recorre cada punto de integración mediante un ciclo `for`. En cada iteración, para el punto `i`, se consulta en el arreglo `array[i]` qué módulo de ejecución debe encargarse del cálculo; el identificador obtenido se almacena en la variable `mod_id`.

Con el identificador, el *codelet* (`GL[]`) invoca al módulo con los parámetros necesarios para su ejecución:

- `sum`: variable para acumular resultado parcial de la integración.
- `func_id`: identificador de la función.
- `DIM`: dimensión de la integral.
- `i+1`: índice del punto actual.
- `X1` y `X2`: límites de integración.
- `ERR`: error tolerado.

El resultado parcial correspondiente se guarda en el arreglo `t[]` para su posterior análisis.

Algoritmo 4: Evaluación punto a punto de la integral multidimensional usando el calendarizador estático secuencial

Entrada: número de puntos `n`, arreglo de los identificadores de módulos `array[]`, parámetros de integración `sum, func_id, DIM, n, X1, X2, ERR`

Salida: resultados parciales de la integral `t[]`

```
1 for i ← 0 to n − 1 do
2 |   mod_id ← array[i];
3 |   GL[mod_id](&sum, func_id, DIM, i + 1, X1, X2, ERR);
4 |   Guardar el resultado parcial en t[i · n] ← sum;
5 end
```

En este proceso:

- `array[i]`: indica qué módulo de ejecución debe calcular la integral para el punto i.
- `GL[mod_id]`: invoca módulo de ejecución (SEC, CUDA, OMP, AVX) correspondiente (véase Sección 3.4.1).

De esta manera, el calendarizador estático secuencial garantiza que cada punto de integración sea procesado por el módulo más adecuado.

Este calendarizador no realiza ningún análisis dinámico en tiempo de ejecución, lo que lo hace ideal en situaciones donde el comportamiento computacional de los módulos es predecible y las condiciones de ejecución no varían significativamente.

Esta estrategia presenta las siguientes ventajas:

- Bajo costo computacional: no se requieren decisiones en tiempo de ejecución.
- Determinismo: la asignación de módulos es fija y repetible.
- Simplicidad de implementación: no requiere sincronización ni estructuras de control complejas.

Sin embargo, su principal desventaja radica en su incapacidad para adaptarse a cambios en el entorno de ejecución, lo cual puede derivar en subutilización de recursos si las condiciones reales difieren de las medidas durante la generación de la tabla, en otras palabra, no se aprovecharía al máximo la capacidad potencial de cada módulo.

3.5.2 Dinámico secuencial

El calendarizador dinámico secuencial presenta una estrategia similar al calendarizador estático secuencial, con la diferencia fundamental de que no realiza una planificación previa completa. En lugar de almacenar los identificadores de los módulos antes

de la ejecución, este calendarizador mantiene en memoria una copia completa de la tabla con los tiempos de ejecución, y realiza la comparación cada vez que el *codelet* solicita una nueva evaluación durante las iteraciones del bucle *for* en la función principal del sistema de calendarización.

El calendarizador recibe como entrada los mismos argumentos que el calendarizador estático secuencial: nombre de la plataforma, número de dimensiones y número de puntos de integración. En la fase de inicialización (Algoritmo 5), lee y almacena en memoria la tabla con los tiempos de ejecución, lo que le permite al sistema comparar estos valores de forma dinámica durante la evaluación de la integral, con el fin de seleccionar el módulo más adecuado en cada punto.

Algoritmo 5: Inicialización del calendarizador dinámico secuencial

Entrada: nombre de la plataforma `PLAT`, número de dimensiones `DIM`, número de puntos de integración `n`
Salida: tabla de tiempos almacenada en memoria `table[]`

```
1 Construir el nombre del archivo: f_name ← PLAT/DIM.txt;
2 Leer el archivo f_time;
3 Reservar memoria para la matriz table[n · m];
4 for i ← 0 to n − 1 do
5 |   for j ← 0 to m − 1 do
6 |   |   Leer valor desde el archivo y almacenarlo en table[i][j];
7 |   end
8 end
9 Cerrar el archivo f_time;
```

En este proceso:

- `n`: representa el número de puntos de integración.
- `m`: representa el número de módulos de ejecución disponibles (secuencial, CUDA, OpenMP y AVX).
- `table`$[n \cdot m]$: almacenada la tabla con los tiempos de ejecución.

El proceso de evaluación de la integral, en la función principal del sistema de calendarización (Algoritmo 6), es casi igual al proceso del Algoritmo 4. Sin embargo, a diferencia del calendarizador estático secuencial que usa un arreglo, este calendarizador invoca de forma dinámica una función especializada: `sched()` (Algoritmo 7). Esta función recibe como entrada el punto de integración `i` que será evaluado y con base en la tabla almacenada en memoria, determina el módulo más adecuado para la ejecución, devolviendo a la salida el identificador de dicho módulo para almacenarlo en la variable *mod_id*.

Para ello, la función `sched()` selecciona, dentro de la tabla de tiempos `table[]`, la fila `i` correspondiente al punto de integración recibido como entrada. Luego, mediante un bucle, compara los tiempos registrados para cada módulo en dicha fila y selecciona

Algoritmo 6: Evaluación punto a punto de la integral multidimensional usando el calendarizador dinámico secuencial

Entrada: número de puntos `n`, parámetros de integración `sum, func_id, DIM, n, X1, X2, ERR`

Salida: resultados parciales de la integral `t[]`

```
1 for i ← 0 to n − 1 do
2 |   mod_id ← sched[i];
3 |   GL[mod_id](&sum, func_id, DIM, i + 1, X1, X2, ERR);
4 |   t[i · n] ← sum;
5 end
```

aquel con el menor tiempo de ejecución; este módulo será el encargado de evaluar la integral en el punto `i`. El valor retornado, `mod`, indica el identificador que recibirá el *codelet* durante el proceso de evaluación de la integral.

El Algoritmo 7 muestra la implementación de la función `sched(i)`. Donde:

- `n`: representa el punto solicitado en la región actual.
- `m`: representa el número de módulos de ejecución disponibles (secuencial, CUDA, OpenMP y AVX).
- `table[`$n \cdot j$`]`: contiene el tiempo estimado para el módulo de ejecución j en el punto n.
- `mod`: representa el identificador del módulo con el menor tiempo para ese punto.

Algoritmo 7: Selección dinámica del módulo más adecuado: `sched()`

Entrada: número del punto de integración `n`, número de módulos `m`, tabla con los tiempos de ejecución `table[]`

Salida: identificador del modulo de ejecución `mod`

```
1 i ← table[n][0];
2 mod ← 0;
3 for j ← 1 to m do
4 |   if table[n][j] < i then
5 |   |   i ← table[n][j];
6 |   |   mod ← j;
7 |   end
8 end
9 return mod;
```

Este enfoque evita la necesidad de generar un arreglo auxiliar, como en el calendarizador estático, y permite una toma de decisiones más ajustada al contexto inmediato de ejecución.

Esta estrategia presenta las siguientes ventajas:

- Mayor adaptabilidad: se selecciona el módulo más adecuado en el momento preciso en que se requiere.
- Reutilización de la tabla completa: permite evaluar nuevos escenarios sin generar el arreglo auxiliar.

Sin embargo, la estrategia no está exenta de desventajas:

- Sobrecarga ligera: al hacer comparaciones cada vez que se requiere un cálculo, se incurre en un pequeño costo adicional respecto al enfoque estático.
- Requiere acceso constante a memoria: es necesario mantener la tabla de tiempos en memoria para acceder a ella en cada evaluación.

En resumen, el calendarizador dinámico secuencial ofrece un balance entre simplicidad y adaptabilidad, siendo especialmente útil en escenarios donde la distribución de cargas es irregular o donde las decisiones deben responder a condiciones cambiantes en la ejecución.

3.5.3 Estático concurrente

En el calendarizador estático concurrente, al igual que el calendarizador estático secuencial, la distribución de las tareas se realiza de forma predeterminada antes de la ejecución y se mantiene fija durante todo el proceso, no varía dinámicamente en función de la carga. Sin embargo, la diferencia radica en su diseño, que aprovecha la ejecución concurrente de tareas en plataformas heterogéneas, siguiendo un enfoque similar a la regla de Johnson (Garey y Johnson, 1976) para calendarización en parejas.

Con el objetivo de aprovechar la concurrencia a nivel de tareas, se implementó un esquema de ejecución por lotes de dos puntos (pares) con espera activa. En cada iteración del bucle `for` de la función principal del sistema de calendarización (Algoritmo 8), se seleccionan dos puntos consecutivos de la integral multidimensional y se organiza la ejecución para calcular la integral en esos puntos, mediante la creación de dos hilos independientes usando OpenMP.

Para evitar conflictos de acceso a recursos compartidos, se crean candados (*locks*) independientes para cada módulo de ejecución disponible. Estos candados garantizan exclusión mutua en el uso de los recursos, evitando conflictos cuando dos hilos intentan acceder simultáneamente al mismo módulo.

Antes de iniciar la evaluación de un punto, el hilo correspondiente intenta adquirir el candado del módulo asignado:

- Si el módulo está libre, el candado se adquiere de inmediato y el hilo comienza la evaluación del punto.
- Si el módulo ya está ocupado, el hilo debe esperar hasta que el candado sea liberado. De este modo, se garantiza que dos puntos asignados al mismo módulo se ejecuten de forma secuencial, incluso dentro de un bloque que intenta ejecutarse de manera concurrente.

Este comportamiento reproduce una espera activa controlada sobre el módulo, pero sin bloquear el avance de otras tareas en módulos diferentes.

El calendarizador inicializa los candados correspondientes a cada módulo (líneas 1–3):

```
for i ← 0 to m − 1 do
   Inicializar candado mod_lock[i];
```

donde `m`: representa el número de módulos de ejecución disponibles (secuencial, CUDA, OpenMP y AVX).

Después, se lanza una región paralela de OpenMP con dos hilos, cada uno responsable de procesar diferentes puntos de integración (línea 6). El acceso al siguiente punto a procesar se controla mediante una operación atómica, de modo que cada hilo obtiene un identificador único sin riesgo de colisiones:

```
#pragma omp atomic capture
{ idx ← next_idx; next_idx++; }
```

Para cada punto de integración (líneas 7–12):

- Se consulta el módulo asignado en el arreglo `array[idx]`.
- Se adquiere el candado correspondiente al módulo antes de su ejecución. Si el módulo está ocupado, el hilo debe esperar activamente hasta que el candado sea liberado.
- Una vez que el módulo está disponible, se ejecuta la función correspondiente mediante el **codelet** (`GL[mod_id](...)`).
- Se libera el candado para que otro hilo puedo acceder al módulo.
- El resultado parcial `local_sum` se almacenan en el arreglo global `t[]`.

Finalmente, al concluir la ejecución de todos los puntos, los candados se destruyen para liberar los recursos asociados (líneas 14–16):

```
for i ← 0 to m − 1 do
   Destruir candado mod_lock[i];
```

Este mecanismo permite que el sistema de calendarización ejecute de manera concurrente distintos puntos de integración sobre módulos diferentes, manteniendo la coherencia en el uso de los recursos. De esta forma, se logra una concurrencia controlada sin introducir condiciones de carrera o bloqueos innecesarios. El Algoritmo 8 muestra la implementación de lo descrito anteriormente. Donde:

- `idx`: representa el identificador del hilo, que a la vez esta asociado al punto de integración.
- `n`: representa el número de puntos de integración.
- `m`: representa el número de módulos de ejecución disponibles (secuencial, CUDA, OpenMP y AVX).

- `array[i]`: indica qué módulo de ejecución debe calcular la integral para el punto `i`.
- `mod_id`: identificador del módulo de ejecución.
- `local_sum`: almacena el resultado parcial de la integración.
- `t[]`: guarda los resultados parciales acumulados para su posterior análisis.

Algoritmo 8: Evaluación en pares de la integral multidimensional usando el calendarizador estático concurrente

Entrada: número de puntos `n`, arreglo de los identificadores de módulos `array[]`, parámetros de la integración `sum, func_id, DIM, n, X1, X2, ERR`

Salida: resultados parciales de la integral `t[]`

for $i \leftarrow 0$ **to** $m - 1$ **do**
  Inicializar candado $mod\_lock[i]$;
end
Paralelizar con 2 hilos usando $OpenMP$;
for *cada hilo* **do**
  `// Obtener los identificadores de forma atómica`
  $\#pragma\ omp\ atomic\ capture\ \{idx \leftarrow next\_idx;\ next\_idx++;\}$
  Determinar módulo asignado $mod\_id \leftarrow array[idx]$;
  Inicializar variable local $local\_sum \leftarrow 0$;
  Adquirir candado $mod\_lock[mod\_id]$;
  $\texttt{GL}[mod\_id](\&local\_sum, func\_id, DIM, idx + 1, X1, X2, ERR)$;
  Liberar candado $mod\_lock[mod\_id]$;
  Guardar los resultados parciales $t[i \cdot n] \leftarrow local\_sum$;
end
for $i \leftarrow 0$ **to** $m - 1$ **do**
  Destruir candado $mod\_lock[i]$;
end

Optimización mediante búsqueda tabú

Previo a la ejecución, el calendarizador genera la asignación de módulos usando una estrategia basada en búsqueda tabú (véase Sección 2.6.2). Esta técnica permite explorar el espacio de soluciones (asignaciones posibles) en busca de una que minimice el tiempo total de ejecución, conocido como *makespan*. La implementación combina:

1. Lectura de la tabla con los tiempos de ejecución.
2. Construcción de las dos mejores opciones para evaluar cada punto de integración (mejor y segundo mejor módulo).
3. Búsqueda metaheurística basada en búsqueda tabú para encontrar una asignación (uso de la segunda mejor opción en algunos puntos) que reduzca el *makespan*.

4. Función simulación del *makespan* que respeta las restricciones de emparejado y exclusión por módulo.

Selección de los dos mejores módulos de ejecución por cada punto de integración

En la fase de inicialización, el calendarizador almacena en memoria la tabla con los tiempos de ejecución de la misma forma que lo hace el Algoritmo 5. Después el Algoritmo 9 identifica los dos mejores módulos de ejecución para cada punto de la integral, basándose en los tiempos de ejecución registrados en la tabla.

Algoritmo 9: Selección de los dos mejores módulos por punto: `build_top2()`

Entrada: número del punto de integración `n`, número de módulos `m`, tabla con los tiempos de ejecución `table[]`
Salida: arreglo con los dos mejores módulos y sus tiempos por cada punto `array_top2[]`

```
1  for i ← 0 to n − 1 do
2      Inicializa los identificadores de los módulos b ← −1, s ← −1;
3      Inicializa el mejor bt ← ∞, y segundo mejor tiempo st ← ∞;
4      for j ← 0 to m − 1 do
5          t ← table[i][j];
6          if t < bt then
7              st ← bt; s ← b ;          /* Segundo mejor tiempo y módulo */
8              bt ← t; b ← j ;                  /* Mejor tiempo y módulo */
9          end
10         else if t < st then
11             st ← t; s ← j;
12         end
13     end
14     Guardar en array_top2[i]: best_mod ← b, second_mod ← s,
        best_t ← bt, second_t ← bt;
15 end
```

Para cada punto, el algoritmo construye un registro con:

- `best_mod`: identificador del módulo con el menor tiempo de ejecución para ese punto.
- `best_t`: tiempo de ejecución de `best_mod`.
- `second_mod`: segundo módulo con el menor tiempo de ejecución para ese punto (si existe; -1 si no).
- `second_t`: tiempo de ejecución de `second_mod`.

Como salida devuelve un arreglo (`array_top2`) de longitud `n`, que contiene todos los registros. En el algoritmo:

- `n`: representa el números de puntos.
- `m`: representa el número de módulos de ejecución disponibles (secuencial, CUDA, OpenMP y AVX).
- `table`[$n \cdot j$]: contiene la tabla con los tiempos de ejecución;

Asignación óptima de módulos de ejecución usando búsqueda tabú

Posteriormente, el Algoritmo 10 determina, por cada punto, cuál de los dos módulos seleccionados será el encargado de evaluarlo, con el propósito de optimizar la asignación mediante una metaheurística de búsqueda tabú. El objetivo principal es minimizar el tiempo total de ejecución (*makespan*) del sistema, explorando diversas combinaciones de asignación y evitando quedar atrapado en óptimos locales.

El algoritmo recibe como entrada:

- `table`[$n \cdot m$]: tabla con los tiempos de ejecución.
- `array_top2[n]`: arreglo con los dos mejores módulos por punto.
- `tenure`: parámetros de configuración del algoritmo tabú.

El procedimiento comienza al crear un arreglo de asignaciones (`assign`), donde cada posición indica si cada punto `i` usa el mejor módulo (0) o el segundo mejor (1). En la asignación inicial cada punto se asocia a su mejor módulo disponible y se calcula el *makespan* llamando a la función `simulate_makespan` (Algoritmo 11), ese valor se guarda como el costo actual y mejor costo inicial (líneas 3 y 4):

```
cur_cost, best_cost ← simulate_makespan(table, array_top2, assign);
```

El algoritmo examina los posibles movimientos para cada punto `i` de la integral (líneas 10–20). Primero, reasigna cada punto a su segundo mejor módulo:

```
assign[i] ← not(assign[i]);
```

Cada movimiento temporal se prueba llamando nuevamente a la función `simulate_makespan` para estimar el nuevo *makespan*. Si el cambio mejora el tiempo total y no está prohibido por la lista tabú, se considera como el mejor movimiento candidato (líneas 17–19):

```
if cost < best_move_cost or best_move == -1 then
   Actualizar best_move ← i, best_move_cost ← cost;
```

Durante la búsqueda, cada movimiento realizado se marca como tabú durante un cierto número de iteraciones, evitando así regresar a soluciones ya exploradas o generar ciclos. Este periodo está determinado por la variable `tenure`, que indica cuántas iteraciones deben pasar antes de que el movimiento vuelva a ser considerado (línea 28).

Algoritmo 10: Búsqueda tabú para asignación óptima de módulos: `tabu_assign_top2()`

Entrada: tabla con los tiempos de ejecución `table[]`, arreglo con los dos mejores módulos y sus tiempos por cada punto `array_top2[]`, parámetros de configuración del algoritmo tabú `tenure`

Salida: registro que contiene el tiempo de ejecución estimado y el arreglo con la mejor asignación de módulos encontrada `tabu_search`

```
1  Inicializar arreglo assign ← 0 todos los puntos usan su mejor módulo
2  Inicializar contador de prohibiciones tabu[N] ← 0;
3  Calcular costo inicial cur_cost ← simulate_makespan(table, array_top2, assign);
4  Guardar como mejor costo best_cost ← cur_cost;
5  for iter ← 1 to max_iters do
6      if supera límite sin mejora then
7          terminar búsqueda
8      end
9      Inicializar best_move ← −1, best_move_cost ← ∞;
10     for i ← 0 to n − 1 do
11         if módulo es tabú o no existe segundo módulo then
12             continuar
13         end
14         Intercambiar asignación assign[i] ← not(assign[i]);
15         Evaluar costo cost ← simulate_makespan(table, array_top2, assign);
16         Revertir cambio assign[i] ← not(assign[i]);
17         if cost < best_move_cost or best_move == −1 then
18             Actualizar best_move ← i, best_move_cost ← cost;
19         end
20     end
21     if no hay movimientos válidos: best_move == −1 then
22         for cada punto i do
23             Decrementar contadores tabu[i] − −
24         end
25     end
       // Aplicar mejor movimiento encontrado
26     Invertir asignación: assign[best_move] ← not(assign[best_move]);
27     Actualizar cur_cost ← best_move_cost;
28     Marcar tarea como tabú tabu[best_move] = tenure;
29     Decrementar todos los contadores tabú mayores a 0;
       // Si mejora el mejor costo global, actualizar best_cost y guardar
          asignación
30     if cur_cost < best_cost then
31         Actualizar best_cost ← cur_cost;
32         Copiar asignación actual a best_a[i] ← assign[i];
33     end
34 end
35 Guardar en tabu_search ← TSOut(best_cost, best_a);
```

En este caso, se estableció `tenure` ← `7`, con el propósito de favorecer la exploración de nuevas regiones del espacio de búsqueda sin perder eficiencia ni agilidad en el proceso.

```
tabu[best_move] ← tenure;
```

En cada iteración, los contadores tabú se reducen gradualmente hasta liberar las soluciones marcadas como tabú (líneas 21–25):

```
if no hay movimientos válidos: best_move == -1 then
   for cada punto i do
     Decrementar contadores tabu[i]-;
```

Si se encuentra una asignación con menor *makespan*, se guarda como mejor asignación global (líneas 30–33).

```
if cur_cost < best_cost then
   best_cost ← cur_cost;
   best_a[i] ← assign[i];
```

Si no hay mejora durante varias iteraciones consecutivas, el algoritmo puede detenerse anticipadamente (líneas 6–8).

```
if supera límite sin mejora then
   Terminar búsqueda;
```

Finalmente, el algoritmo devuelve un registro `tabu_search` que contiene el tiempo de ejecución estimado y el arreglo con la mejor asignación de módulos encontrada.

Función de simulación del makespan

La función `simulate_makespan` (Algoritmo 11) tiene como objetivo estimar el tiempo total de ejecución (*makespan*) que resultaría de una determinada asignación de los puntos de integración a los módulos. Es decir, simula cómo se comportaría el sistema de calendarización de manera concurrente, usando solamente los tiempos medidos, lo que permite que el Algoritmo 10 pueda evaluar diferentes estrategias de asignación hasta encontrara la más optima.

Para simular el esquema de ejecución por lotes de dos puntos (pares) con espera activa, se incluyen las siguientes restricciones:

- Exclusión por módulo de ejecución: un mismo módulo no puede ejecutar dos tareas simultáneamente.
- Restricción de paralelismo máximo: sólo 2 puntos pueden estar en ejecución al mismo tiempo.

Como entradas, el Algoritmo 11 de la función recibe:

- `n`: número de puntos de integración.

- `m`: número de módulos de ejecución disponibles (secuencial, CUDA, OpenMP y AVX).
- `array_top2[n]`: arreglo con los dos mejores módulos por punto.
- `assign[n]` arreglo con la asignación de los módulos para cada punto.

El algoritmo inicializa los tiempos de liberación de cada módulo, es decir, el instante en el que cada módulo estará disponible para ejecutar la evaluación de un nuevo punto:

```
mod_free[mod_id] ← 0
```

Después, simula el comportamiento del sistema de calendarización al ejecutar, por pares, los puntos de la integral. La selección del módulo de ejecución para cada punto se realiza con base en el plan de asignación contenido en el arreglo `assign[]`, que determina si se emplea el módulo con el mejor tiempo estimado o el módulo con el segundo mejor tiempo. Para ello, ejecuta el bucle principal `while` mientras haya puntos pendientes por calcular `next < n` o haya puntos que todavía no terminan de calcularse `running > 0` (líneas 3–25). Dentro del bucle principal hay otro bucle que intenta calcular nuevos puntos, pero con una restricción: sólo puede se pueden evaluar dos puntos al mismo tiempo (líneas 4–17).

```
While{running < 2 and next < N}{...}
```

Si se cumplen las condiciones, se toma el siguiente punto `use_second ← assign[next]` y se verifica si debe ejecutarse en su mejor o segundo mejor módulo, de acuerdo con el valor almacenado en `assign[next]`. A partir de esta selección, se obtiene el identificador del módulo elegido, `mod_id`, junto con su tiempo de ejecución esperado, `dur` (líneas 5 y 6).

La evaluación del punto sólo puede empezar si el módulo está libre y el tiempo actual ya lo permite, para ello se calcula el momento en el que puede iniciar (línea 7):

```
start ← max(mod_free[mod_id], cur_time);
```

Para evitar conflictos entre procesos, si ya hay un punto siendo evaluado (`running == 1`), se verifica que la nueva evaluación que se lanzará no use el mismo módulo que usa la que ya está activa (líneas 8–10):

```
if d == used_mod then break;
```

En el caso de que sea el mismo módulo, la nueva evaluación no puede comenzar aún, por lo que el bucle interno se detiene hasta que dicho módulo se libere. Si no existen conflictos, se calcula el tiempo de finalización de la nueva evaluación (`endt ← start + dur`) y se determina en cuál de los dos *slots* disponibles se colocará: `A` o `B` (líneas 11 y 12). El sistema dispone únicamente de dos espacios de ejecución en paralelo, siguiendo un esquema por pares.

```
endA ← endt;
modA ← d;
```

o

```
endB ← endt;
modB ← d;
```

Se marca el módulo como ocupado hasta ese momento `mod_free[d] ← endt` y se incrementan los contadores (líneas 14–15):

```
running++; (una tarea más en ejecución)
next++; (pasar a la siguiente tarea)
```

Si esto ocurre, la variable `progressed` se pone en 1, indicando que hubo progreso en la simulación (línea 15). Si no se pudo lanzar ninguna tarea `progressed = 0`, eso significa que todos los módulos requeridos están ocupados. Por lo tanto, el algoritmo avanza al siguiente evento, es decir, al momento en que un punto termina de calcularse (líneas 18–24:

```
next_event ← min(endA, endB);
cur_time ← next_event;
```

Si una evaluación termina (por ejemplo, `endA ← cur_time`), se marca como libre (líneas 21–23):

```
endA ← ∞;
modA ← -1;
running-;
```

Así, el simulador "avanza" hasta que haya espacio para lanzar nuevas tareas. Cuando todas las tareas han sido ejecutadas `next >= n` y no hay tareas corriendo `running == 0`, el bucle termina.

Finalmente, el valor de `cur_time` representa el tiempo total que habría tardado el sistema en completar todas las tareas, cumpliendo con las restricciones establecidas. Dicho valor se devuelve como el *makespan* simulado.

En el Algoritmo 11:

- `mod_free[d]`: representa el instante en que el módulo `mod_id` queda libre (inicialmente 0).
- `cur_time`: representa el tiempo actual de simulación.
- `running`: representa el número de puntos evaluados en curso (0, 1 o 2).
- `next`: representa el índice del siguiente punto sin evaluar.
- `endA`, `endB`: representan los tiempos de finalización de las dos evaluaciones actualmente en ejecución.
- `modA`, `modB`: representa el módulo asociado a cada slot en ejecución.

Algoritmo 11: Simulación del makespan: `simulate_makespan()`

Entrada: número de puntos de integración `n`, número de módulos `m`, arreglo con los dos mejores dispositivos por punto `array_top2[]`, arreglo con la asignación de los módulos para cada punto `assign[]`

Salida: tiempo total de ejecución `cur_time`

```
1  Inicializar arreglo mod_free[m] ← 0 para cada módulo;
2  Inicializar cur_time ← 0, running ← 0, next ← 0;
3  while running > 0 or next < n do
4      while running < 2 and next < n do
5          use_second ← assign[next] ;        /* Determinar el módulo a usar */
6          Obtener módulo mod_id y duración dur;
7          Calcular start ← max(mod_free[mod_id], cur_time);
8          if mod_id == used_mod then
9              break ;                /* Romper bucle para esperar disponibilidad */
10         end
11         Calcular tiempo de finalización endt ← start + dur;
12         Registrar mod_id y endt en un slot libre: A o B
13         Actualizar mod_free[d] ← endt;
14         Registrar tarea como en ejecución running + +;
15         Incrementar next + +;
16         progressed ← 1;
17     end
18     if progressed == 0 then
           // Avanzar al siguiente evento
19         next_event ← min(endA, endB);
20         cur_time ← next_event;
           // Liberar tareas que terminen en ese instante
21         endA ← ∞;
22         devA ← −1;
23         running − −;
24     end
25 end
26 Retornar cur_time como tiempo total de ejecución (makespan);
```

Función principal del calendarizador estático concurrente

El Algoritmo 12 coordina la ejecución de los Algoritmos 9, 10 y 11, integrándolos para conformar el calendarizador estático concurrente. Su propósito es asignar el módulo encargado de evaluar cada punto, con el objetivo de minimizar el tiempo total de ejecución (*makespan*).

Primero, carga en memoria la tabla con los tiempos de ejecución registrados para cada módulo y cada punto de integración. Luego, analiza los tiempos de cada fila de la tabla para identificar, por punto el mejor y segundo mejor módulo de ejecución

y almacena los datos en un arreglo `array_top2` (Función `build_top2()` correspondiente al Algoritmo 9). Esta etapa reduce el espacio de búsqueda, permitiendo que la optimización se centre sólo en las dos mejores opciones por punto, en lugar de considerar todos los módulos posibles.

Posteriormente, el algoritmo usa una metaheurística de búsqueda tabú (Función `tabu_assign_top2()` correspondiente al Algoritmo 10) para decidir, entre las dos opciones de cada punto, cuál asignación global produce el menor tiempo total de ejecución (*makespan*). Con los resultados de la búsqueda tabú, el algoritmo construye el arreglo final de asignación, `array[i]`, donde cada elemento contiene el identificador del módulo que ejecutará el punto `i`. Este arreglo se guarda en memoria y luego es utilizado por el sistema de calendarización (líneas 6–9).

Algoritmo 12: Función principal del calendarizador estático concurrente

Entrada: nombre de la plataforma `PLAT`, número de dimensiones `DIM`, número de puntos de integración `n`
Salida: arreglo con los identificadores de los módulos más adecuados por punto `array[]`

1 Leer tabla con tiempos de ejecución $table = read\_table(PLAT, DIM, n)$;
2 Construir $array\_top2$ con los dos mejores módulos por punto: $array\_top2 = build\_top2(tabla)$;
3 $tenure \leftarrow 7$
4 Ejecutar búsqueda tabú $tabu\_search = tabu\_assign\_top2(table, array\_top2, tenure)$;
5 Reservar memoria para `array`;
6 **for** $i \leftarrow 0$ **to** $n - 1$ **do**
7 | Seleccionar módulo d según asignación óptima (*best* o *second*);
8 | Guardar en $array[i] \leftarrow d$;
9 **end**

Esta estrategia presenta las siguientes ventajas:

- Concurrencia: explota la concurrencia de forma sencilla, evitando la sobrecarga de un calendarizador dinámico.
- Espera activa: incluir candados por módulo garantiza que no se produzcan accesos simultáneos no controlados a un mismo recurso.

Sin embargo, presenta ciertas limitaciones:

- Si el tiempo de ejecución de los puntos asignados a un módulo es significativamente mayor que en otros, el sistema de calendarización puede experimentar desequilibrios de carga.
- Al no replanificar dinámicamente, no se corrige la asignación inicial aunque se detecten ineficiencias durante la ejecución.

- La creación de hilos genera un *overhead* en el tiempo de ejecución.
- Los procesos generados pueden crear retrasos en la ejecución al competir por el tiempo de la CPU.

Por lo anterior, el esquema estático concurrente resulta especialmente adecuado cuando: se dispone de una planificación previa de alta calidad, los tiempos de ejecución presentan una variabilidad reducida o predecible y existe certeza de que los procesos no competirán por recursos de CPU.

3.5.4 Tabla de características

La Tabla 3.3 presenta una comparación entre los tres calendarizadores desarrollados, sintetizando sus diferencias de manera estructurada. Cada uno de los calendarizadores representa una estrategia distinta de asignación y evaluación de puntos sobre los módulos de ejecución disponibles, variando principalmente en su nivel de optimización, adaptabilidad y grado de paralelismo.

Tabla 3.3: Características de los tres calendarizadores implementados.

Característica	**Estático secuencial**	**Dinámico secuencial**	**Estático concurrente**
Tipo de planificación	Previa y fija	En tiempo de ejecución	Previa optimizada por búsqueda tabú
Asignación de módulos	Se guarda en un arreglo auxiliar	Se calcula al momento de cada llamada	Se guarda en arreglo optimizado
Uso de recursos	Secuencial	Secuencial	Paralelo, dos módulos en ejecución sin conflicto
Modelo de ejecución	Punto por punto	Punto por punto	Ejecución en pares simultáneos
Adaptabilidad	Nula: depende solo de la tabla de tiempos	Alta: decide el mejor módulo en cada paso	Media: depende de planificación previa (búsqueda tabú)
Sincronización	No requerida	No requerida	Requiere candados por módulo para evitar conflictos

El calendarizador estático secuencial basa su funcionamiento en una planificación previa y fija. Las asignaciones de cada punto a un módulo se calculan una sola vez antes de la ejecución y se almacenan en un arreglo auxiliar. Debido a su naturaleza determinista y su ejecución punto por punto, este esquema no requiere sincronización ni toma de decisiones durante el proceso, aunque su aprovechamiento de recursos es limitado, ya que sólo utiliza un módulo a la vez.

Por otro lado, el calendarizador dinámico secuencial realiza la selección del módulo más adecuado en tiempo de ejecución. En cada paso, evalúa las opciones disponibles y elige el módulo que ofrece el menor tiempo de procesamiento para el punto actual. Esta característica proporciona una mayor adaptabilidad, ya que el calendarizador puede reaccionar ante variaciones en los tiempos de ejecución. Sin embargo, al igual que el

calendarizador estático secuencial, su ejecución permanece totalmente secuencial, sin explotar la posible concurrencia entre módulos de ejecución.

Finalmente, el calendarizador estático concurrente representa una evolución del enfoque estático. Aunque también se basa en una planificación previa, ésta se obtiene mediante un proceso de optimización con búsqueda tabú. Dicha técnica explora múltiples combinaciones de asignación para encontrar una configuración que minimice el tiempo total de ejecución (*makespan*), considerando la posibilidad de evaluar dos puntos de forma simultánea en distintos módulos sin generar conflictos. Este método logra un mejor aprovechamiento de los recursos al permitir concurrencia, aunque requiere mecanismos de sincronización (como candados) para garantizar que los módulos no se asignen de manera conflictiva.

Capítulo 4

Pruebas

En este capítulo se presentan las pruebas realizadas para evaluar el desempeño, la precisión y la adaptabilidad del sistema de calendarización implementado. El objetivo principal es analizar la eficiencia de los módulos de ejecución y las estrategias de calendarización propuestas bajo diferentes condiciones de *hardware* y configuraciones de prueba. Es decir, diferentes números de puntos de integración y dimensiones. Las pruebas se dividieron en dos grandes apartados: pruebas preliminares y pruebas de calendarización.

En la primera sección se describen las plataformas utilizadas para ambos tipos de pruebas, junto con las funciones a evaluar y las principales condiciones consideradas para su ejecución.

En la segunda sección, correspondiente a las pruebas preliminares, se valida el funcionamiento correcto de los módulos de ejecución y se establecen los parámetros base necesarios para las pruebas que evaluarán el rendimiento de los calendarizadores. Se analizan tres aspectos fundamentales: la complejidad computacional de las funciones del *benchmark* de integrales multidimensionales, con el fin de seleccionar la más adecuada para las pruebas posteriores; la estimación del número mínimo de puntos de integración necesarios para alcanzar un error aceptable; y la aplicación de la extrapolación de Romberg para reducir el número de puntos requeridos y, en consecuencia, el tiempo de ejecución, sin comprometer la precisión.

La última sección, correspondiente a las pruebas de calendarización, evalúa el desempeño de las tres estrategias implementadas: estático secuencial, dinámico secuencial y estático concurrente. En primer lugar se construyen las tablas de tiempo que sirven para alimentar a los calendarizadores. Posteriormente se presentan las pruebas realizadas con los calendarizadores en dos plataformas con diferentes capacidades de cómputo: una estación de trabajo CUDA y una Jetson TX2. Estas pruebas permiten comparar la eficiencia de cada estrategia en entornos heterogéneos, identificando las ventajas y limitaciones de cada estrategia, así como su capacidad para adaptarse a las restricciones de *hardware*.

En resumen, este capítulo establece los fundamentos experimentales que permiten valorar la efectividad del sistema de calendarización propuesto, proporcionando evidencia cuantitativa sobre su precisión numérica, eficiencia computacional y flexi-

bilidad frente a distintos entornos de ejecución.

4.1 Funciones, dispositivos y condiciones base

Como se mencionó anteriormente, las pruebas fueron diseñadas para validar el comportamiento, la eficiencia y la adaptabilidad del sistema de calendarización propuesto. Para ello, se utilizaron las funciones del *benchmark* de integrales multidimensionales, debido a su variabilidad en complejidad y precisión numérica. Estas funciones permiten evaluar de forma robusta el rendimiento de los distintos módulos de ejecución y calendarizadores implementados.

En el módulo de ejecución OMP las pruebas no se ejecutaron sobre todos los procesadores de manera simultánea. En cambio, se definieron dos configuraciones: (1) OMP-P, emplea únicamente los núcleos de rendimiento (*performance cores*) y (2) OMP-E, utiliza los núcleos de eficiencia (*efficient cores*). El objetivo de esta separación es permitir que cada núcleo alcance su frecuencia máxima de operación, sin verse limitado por la necesidad de sincronizarse con otros núcleos de distinta naturaleza.

Cada prueba que mide el tiempo de ejecución fue repetida diez veces consecutivas, con el objetivo de mitigar posibles fluctuaciones ocasionadas por factores externos al sistema de calendarización; como procesos en segundo plano o gestión dinámica de recursos por parte del sistema operativo. Como medida representativa de cada conjunto de ejecuciones se tomó la mediana, debido a que proporciona una estimación más robusta frente a valores atípicos que podrían sesgar el resultado si se utilizaran (e. g., el promedio).

Las pruebas se llevaron a cabo en dos plataformas heterogéneas diferentes: una estación de trabajo con sistema operativo Rocky Linux 9 y un sistema embebido Jetson TX2 con sistema operativo Ubuntu 18.04. Esta decisión responde a la necesidad de validar el sistema de calendarización bajo entornos heterogéneos. Por ello, es necesario contar con plataformas que representen extremos opuestos en cuanto a potencia de cómputo, disponibilidad de módulos de ejecución, arquitecturas de CPU y capacidades de vectorización y paralelismo.

Emplear una estación de trabajo CUDA y una Jetson TX2 proporciona un contraste claro entre un sistema de escritorio de alto rendimiento y un sistema embebido con recursos limitados. Además, ambas plataformas difieren en los módulos de ejecución que pueden utilizar.

La estación de trabajo permite emplear todos los módulos disponibles: secuencial, AVX, OpenMP y CUDA. En cambio, la Jetson TX2 incorpora una CPU ARM que no soporta instrucciones AVX, por lo que no puede ejecutar dicho módulo; adicionalmente, sus capacidades de paralelismo son diferentes a las de la estación de trabajo, lo que modifica el comportamiento del módulo OMP. Estas diferencias permiten verificar que el sistema de calendarización detecta automáticamente los módulos disponibles en cada plataforma, evita el uso de módulos incompatibles (como el módulo AVX) y redistribuye la carga de trabajo sin requerir intervención del usuario.

En resumen, la combinación seleccionada de plataformas permite validar el sistema de calendarización en condiciones de alto rendimiento, recursos limitados, heterogeneidad arquitectónica extrema, disponibilidad desigual de módulos y diferentes sistemas operativos.

Esta diversidad asegura una evaluación realista y sólida del desempeño del sistema de calendarización implementado. A continuación, se presentan las especificaciones técnicas de las plataformas:

- Estación de trabajo CUDA:
  - GPU: GA104 [GeForce RTX 3070 Lite Hash Rate] 64-bits
  - CPU: 13th Gen Intel® Core i9-13900K 64-bits
  - Memoria RAM: 31 GB 64-bits SRAM
- NVIDIA Jetson TX2:
  - GPU: Arquitectura NVIDIA Pascal con 256 NVIDIA CUDA cores
  - CPU: Dual-Core Nvidia Denver 2 64-bits CPU y Procesador Quad-Core Arm® Cortex®-A57 MPCore
  - Memoria RAM: 8 GB 128-bits LPDDR4 59.7 GB/s

4.2 Pruebas preliminares

Antes de realizar las pruebas de calendarización, fue necesario llevar a cabo un conjunto de pruebas preliminares destinadas a validar la correcta operación de los módulos de ejecución con y sin la extrapolación de Romberg, así como establecer algunas condiciones para los experimentos posteriores. Estas pruebas se enfocan en analizar tres aspectos fundamentales: la complejidad computacional de las funciones del *benchmark*, el mínimo número de puntos de integración para obtener un error numérico aceptable, y el efecto de la extrapolación de Romberg en la precisión numérica y la reducción del tiempo de ejecución.

En conjunto, estas pruebas proporcionan una base sólida para comprender el comportamiento numérico de la biblioteca de integrales multidimensionales, asegurando que las configuraciones elegidas para las etapas posteriores representen un equilibrio adecuado entre precisión y eficiencia computacional.

4.2.1 Pruebas para evaluar la complejidad de las funciones del benchmark de integrales multidimensionales

Con el objetivo de seleccionar una función adecuada para realizar las pruebas de calendarización, se evaluó la complejidad computacional de cada una de las cinco funciones contenidas en el *benchmark* de integrales multidimensionales, las cuales poseen estructuras algebraicas y grados de dificultad diferentes. Para ello, se realizaron ejecuciones utilizando exclusivamente los módulos implementados con la regla de

cuadratura de Gauss-Kronrod, sin aplicar la extrapolación de Romberg, dentro de un rango de 40 a 50 puntos de integración. Las pruebas se efectuaron únicamente en seis dimensiones, con el propósito de observar con mayor claridad la complejidad de cada función, dado que aumentar las dimensiones incrementa significativamente el costo computacional. El análisis de estos resultados permitió comparar de forma objetiva el tiempo requerido por cada función bajo condiciones controladas y homogéneas.

La Tabla 4.1 presenta las funciones del *benchmark* y el tiempo (s) que tardaron en evaluarse en cada módulo de ejecución en relación con el número de puntos. Cada columna (Función 1.1 – Función 1.5) corresponde a una función y los resultados se agrupan por módulo:

- Secuencial: implementación pura en CPU sin paralelismo ni vectorización.
- CUDA: ejecución paralela en GPU utilizando cómputo masivamente paralelo.
- OMP-E: ejecución paralela en CPU empleando únicamente los núcleos de eficiencia.
- OMP-P: ejecución paralela en CPU empleando únicamente los núcleos de rendimiento.
- AVX: implementación vectorizada por *hardware* SIMD usando instrucciones AVX.

Cada sección presenta los tiempos correspondientes a cinco puntos de integración consecutivos (46–50), lo que permite comparar la complejidad y el comportamiento de las funciones en todos los módulos de ejecución, evitando sesgos que podrían surgir si se evaluaran únicamente en un módulo.

A partir de los datos presentados en la Tabla 4.1, se determinó que:

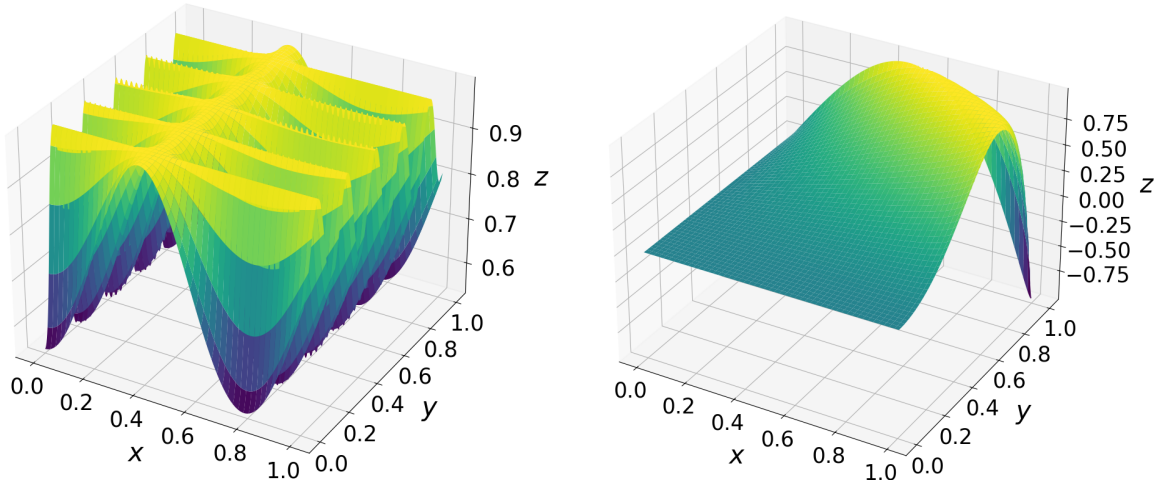
- La Función 1.1 presenta los tiempos de ejecución más bajos en todos los módulos. Su crecimiento, con respecto al número de puntos, es casi perfectamente proporcional, lo que confirma que el costo por evaluación es mínimo y estable.
- La Función 1.5 resulta ser ligeramente más costosa que la Función 1.1, pero claramente menos costosa que las demás funciones. Esto indica que el integrando de la Función 1.5 agrega cierta complejidad adicional respecto a la Función 1.1, pero no presenta términos altamente costosos. El patrón se mantiene estable en todas las arquitecturas, lo que confirma una complejidad baja a media.
- La Función 1.4 ocupa un punto intermedio entre las funciones. Su crecimiento es suave y coherente, lo cual apunta a una complejidad estable y relativamente menor en comparación con las Funciones 1.2 y 1.3.
- Los tiempos de la Función 1.3 son sistemáticamente mayores que los de las Funciones 1.1, 1.4 y 1.5, y son cercanos a los de la Función 1.2, aunque ligeramente menores. Esto indica una complejidad aritmética elevada pero no extrema.
- La Función 1.2 es consistentemente la función más costosa en la mayoría de los módulos. La marcada diferencia en tiempo respecto a otras funciones evidencia que su complejidad aritmética es significativamente mayor.

Tabla 4.1: Funciones del *benchmark* de integrales multidimensionales evaluadas en 6 dimensiones, en estación de trabajo CUDA. Tiempo de ejecución (s).

Puntos	Función 1.1	Función 1.2	Función 1.3	Función 1.4	Función 1.5
SECUENCIAL					
46	75.3996625	936.0831740	809.1952620	238.0729600	276.2268660
47	85.7801670	1072.7830750	918.7347710	272.3382770	311.3526115
48	97.2724690	1215.6837025	1041.5524930	308.9656615	354.3307145
49	109.9852875	1379.2441440	1177.5097000	349.2575320	401.5813935
50	124.1955935	1549.0077680	1329.3573790	394.9378110	447.0571705
CUDA					
46	3.3083423	43.8635459	46.9730884	11.1234946	8.3254663
47	3.8602425	47.9857967	52.0492519	12.8510994	9.1125629
48	4.1145525	51.1187600	55.4455836	13.6634206	9.6894928
49	5.0105843	66.9855101	71.7827684	17.3861715	12.7275292
50	5.3468530	71.0629356	76.2570657	18.4159402	13.5162827
OMP-E					
46	17.5718030	154.2491160	114.4462040	40.8227235	47.5153235
47	19.3367120	172.0041920	127.7702755	47.0031900	55.1497895
48	20.9470510	190.2397720	141.4722120	50.5358580	59.6946905
49	27.9068520	281.7533230	210.9048005	71.9929870	87.9230165
50	32.3194350	310.0759610	230.7351250	79.7889495	94.8209015
OMP-P					
46	8.7982325	101.4547705	87.9471200	28.6811405	32.2358530
47	9.8808690	114.4845110	98.4313410	32.4126835	36.1624040
48	11.0473450	127.7699575	109.9139960	36.2801730	40.5722820
49	14.5778090	170.1227050	146.5415880	47.6676100	53.2759140
50	16.2536900	189.5077510	163.3546610	53.1579585	58.9326380
AVX					
46	70.2210900	929.1938370	813.6420780	232.5348960	268.7045565
47	80.0087370	1062.7284845	923.7070030	264.1679865	304.7978095
48	90.2645070	1206.1304360	1044.9015930	298.0131675	343.3589055
49	102.2595550	1355.8468980	1180.5045915	337.5464170	391.4487065
50	115.5569825	1536.1999955	1334.8980040	381.4114960	438.6309785

La elevada complejidad de la Función 1.2 se debe a su comportamiento altamente oscilatorio, como se muestra en la Figura 4.1a. Esta característica la convierte, en principio, en una candidata adecuada para las pruebas de calendarización y para la estimación del error numérico: al tratarse de la función más compleja, lograr una reducción en el tiempo de ejecución y obtener un error aceptable implicaría que las demás funciones, de menor complejidad, también podrían evaluarse de manera eficiente sin perder precisión numérica.

Sin embargo, se observó que, debido a su elevada complejidad, los algoritmos de integración empleados en este trabajo no permiten reducir de forma significativa el tiempo de ejecución sin comprometer la precisión del resultado. Como consecuencia, esta función fue descartada como caso de prueba principal para los calendarizadores implementados, ya que no ofrece una mejora tangible en términos de optimización del desempeño.



(a) Función 1.2 **(b)** Función 1.3

Figura 4.1: Funciones del *benchmark* de integrales multidimensionales.

En su lugar, se eligió la Función 1.3 como representante de una función lo suficientemente compleja pero abordable. Su comportamiento presenta una oscilación más suave, como se ilustra en la Figura 4.1b, lo que permite observar reducciones significativas en el tiempo de ejecución sin comprometer la precisión del resultado. Si el sistema de calendarización propuesto es capaz de optimizar correctamente la evaluación de la Función 1.3, también podrá manejar de manera eficiente las otras tres funciones restantes del *benchmark* (Funciones 1.1, 1.4 y 1.5), las cuales presentan un menor grado de complejidad.

4.2.2 Pruebas para estimar el mínimo número de puntos para un error aceptable

El propósito de estas pruebas es estimar el número mínimo de puntos de integración necesarios para alcanzar un error numérico aceptable en el cálculo de integrales multidimensionales. En función del dominio del problema, el error puede ser $\leqslant 10^{-4}$, tomando como referencia la convergencia de los resultados obtenidos en los distintos módulos de ejecución.

Se realizaron pruebas independientes con los módulos de ejecución (CUDA, OMP y AVX) disponibles en la biblioteca de integrales multidimensionales, usando la Función 1.3, sin aplicar la extrapolación de Romberg, con el fin de determinar la precisión del resultado que ofrece la cuadratura de Gauss-Kronrod. El rango de prueba comprendió desde 125 hasta 129 puntos de integración, permitiendo observar la evolución del error conforme se incrementa el número de puntos.

La Tabla 4.2 presenta los resultados de la integral para cada número de puntos. Los datos están organizados en bloques, cada bloque corresponde a una dimensión específica: 3, 4, 5 y 6 dimensiones. Las filas de cada bloque corresponden a los números de puntos evaluados (125–129) y reportan el resultado numérico resultante al aplicar

la cuadratura de Gauss-Kronrod con ese número de puntos. Por último, cada columna representa el módulo de ejecución que evaluó la integral (CUDA, OMP y AVX).

Tabla 4.2: Error numérico de la Función 1.3, evaluada en estación de trabajo CUDA usando la cuadratura de Gauss-Kronrod.

Puntos	CUDA	OMP	AVX
3 Dimensiones			
125	0.166292299805	0.166292299805	0.166292299805
126	0.166292235618	0.166292235618	0.166292235618
127	0.166292172938	0.166292172938	0.166292172938
128	0.166292111719	0.166292111719	0.166292111719
129	0.166292051917	0.166292051917	0.166292051917
4 Dimensiones			
125	0.090418755438	0.090418755438	0.090418755437
126	0.090418698747	0.090418698746	0.090418698746
127	0.090418643388	0.090418643388	0.090418643387
128	0.090418589320	0.090418589320	0.090418589320
129	0.090418536503	0.090418536503	0.090418536503
5 Dimensiones			
125	0.048806937502	0.048806937495	0.048806937499
126	0.048806884346	0.048806884339	0.048806884318
127	0.048806832464	0.048806832457	0.048806832400
128	0.048806781811	0.048806781803	0.048806781808
129	0.048806732352	0.048806732344	0.048806732349
6 Dimensiones			
125	0.025751755724	0.025751754079	0.025751754869
126	0.025751706296	0.025751704559	0.025751705400
127	0.025751658139	0.025751656314	0.025751657183
128	0.025751611139	0.025751609216	0.025751564539
129	0.025751565567	0.025751563564	0.025751520330

En todos los casos, los resultados numéricos obtenidos por los módulos son idénticos o prácticamente idénticos (sólo hay diferencias en el último dígito debido al redondeo de doble precisión). Esto indica que:

- Todos los módulos de ejecución implementan correctamente la regla de cuadratura de Gauss-Kronrod.
- No existe pérdida de precisión atribuible a diferencias en paralelismo o vectorización.
- El cálculo de la integral es estable y depende únicamente del número de puntos, no del módulo empleado.

Al comparar los resultados, fue posible analizar la evolución del error numérico para los distintos números de puntos de integración. En particular, se observó que:

- Para 125, 126 y 127 puntos, el error disminuye, pero lo hace en el orden de 10^{-7} a 10^{-8} por incremento de un punto.

- A partir de 128 puntos, las mejoras adicionales se reducen prácticamente al orden del sexto decimal.
- Las diferencias entre CUDA, OMP y AVX se mantienen iguales hasta el sexto decimal, lo que indica que el resultado ya se encuentra en una zona de *estabilidad numérica*.

Este comportamiento sugiere que el método de integración alcanza una región en la que incrementar el número de puntos deja de generar mejoras significativas en la precisión. Por ello, se seleccionó como referencia el número mínimo de puntos a partir del cual el error deja de disminuir de manera apreciable: 128. Este número de puntos proporciona una precisión de 10^{-6}, dentro del margen aceptable, sin necesidad de incrementar el costo computacional. Además, es un punto de equilibrio óptimo entre exactitud y eficiencia.

Estimación del error numérico aplicando la extrapolación de Romberg

Una vez obtenido el error numérico usando la cuadratura de Gauss-Kronrod, se procedió a realizar una segunda serie de pruebas añadiendo el método de extrapolación de Romberg, con el objetivo de evaluar su impacto en la precisión y el tiempo de ejecución. Este conjunto de experimentos utilizó la misma función de prueba y los mismos módulos de ejecución, manteniendo las condiciones de prueba previamente establecidas para asegurar una comparación justa de los resultados.

La extrapolación de Romberg se implementó con el fin de reducir el tiempo de ejecución, manteniendo una precisión aceptable en los resultados al evaluar las integrales multidimensionales. A partir de las aproximaciones calculadas mediante la cuadratura de Gauss-Kronrod con distintos números de puntos, el método de Romberg estima un valor límite que se aproxima al resultado exacto mediante una secuencia de refinamientos sucesivos. De esta manera, se logra una corrección sistemática del error de truncamiento, permitiendo alcanzar niveles de precisión comparables a los de una cuadratura más densa, pero con un número considerablemente menor de evaluaciones de la función.

En estas pruebas, se redujo el número de puntos de integración a un rango de 36 a 40, con el propósito de evaluar si la extrapolación de Romberg permite conservar una precisión aceptable, al mismo tiempo que disminuye el tiempo de ejecución.

La Tabla 4.3 presenta los resultados al integrar la Función 1.3 mediante la cuadratura de Gauss-Kronrod combinada con la extrapolación de Romberg. La tabla sigue la misma estructura de la Tabla 4.2, se divide en cuatro bloques correspondientes a 3, 4, 5 y 6 dimensiones. Las filas, de cada bloque, corresponden a los números de puntos utilizados (36–40) y las columnas representan los módulos de ejecución que evaluaron la integral (CUDA, OMP y AVX).

Esta organización permite observar con claridad que el error numérico comienza a estabilizarse a partir de 40 puntos de integración. En ese punto, la extrapolación de Romberg produce un error aproximado de 10^{-4}, valor que, aunque ligeramente superior al obtenido sin aplicar extrapolación, se considera aceptable dentro de los

márgenes definidos en esta tesis para mantener un equilibrio entre precisión y eficiencia computacional.

Tabla 4.3: Error numérico de la Función 1.3, evaluada en estación de trabajo CUDA usando la cuadratura de Gauss-Kronrod y la extrapolación de Romberg.

Puntos	CUDA	OMP	AVX
3 Dimensiones			
36	0.166335717828	0.166335717828	0.166335717828
37	0.166333229263	0.166333229263	0.166333229263
38	0.166330931258	0.166330931258	0.166330931258
39	0.166328804849	0.166328804849	0.166328804849
40	0.166326833371	0.166326833371	0.166326833371
4 Dimensiones			
36	0.090457281921	0.090457281921	0.090457281921
37	0.090455062320	0.090455062320	0.090455062320
38	0.090453014013	0.090453014013	0.090453014013
39	0.090451119806	0.090451119806	0.090451119806
40	0.090449364567	0.090449364567	0.090449364567
5 Dimensiones			
36	0.048844879520	0.048844879520	0.048844879520
37	0.048842446717	0.048842446717	0.048842446717
38	0.048841175414	0.048841175414	0.048841175414
39	0.048838629778	0.048838629778	0.048838629778
40	0.048837194702	0.048837194702	0.048837194702
6 Dimensiones			
36	0.025778016783	0.025778016785	0.025778016785
37	0.025776259169	0.025776259164	0.025776259164
38	0.025775398725	0.025775398727	0.025775398717
39	0.025774084951	0.025774084950	0.025774084940
40	0.025773357342	0.025773357345	0.025773357345

El error obtenido (10^{-4}) confirma que el método de extrapolación de Romberg permite obtener una precisión adecuada con un número de puntos considerablemente menor, lo cual representa una reducción significativa en el costo computacional. El tiempo de ejecución registrado para 40 puntos con la extrapolación de Romberg fue notablemente inferior al correspondiente a 128 puntos sin ella, lo que sugiere una ganancia de eficiencia importante. No obstante, el análisis detallado de estos tiempos y de la influencia de las estrategias de calendarización sobre dicho rendimiento se aborda en el siguiente capítulo (véase Sección 5.1).

4.3 Pruebas de calendarización

Una vez verificado que el error numérico se encuentra dentro del margen aceptable y determinado el número de puntos necesarios para alcanzarlo (128 puntos producen un error de 10^{-6} y 40 puntos alcanzan un error de 10^{-4}), se procedió a evaluar el comportamiento de las diferentes estrategias de calendarización implementadas. El objetivo de estas pruebas es analizar la eficiencia de cada estrategia en la asignación de

puntos de integración entre los distintos módulos disponibles, considerando escenarios con diferentes dimensiones y arquitecturas de *hardware*.

Cada calendarizador fue probado tanto en la estación de trabajo CUDA como en la Jetson TX2, lo que permitió analizar su comportamiento en plataformas heterogéneas con diferentes capacidades de cómputo. Todas las pruebas se realizaron usando la misma función de prueba (Función 1.3).

4.3.1 Tablas de tiempo para alimentar los calendarizadores

Con el propósito de generar información de referencia que permita a los calendarizadores seleccionar de forma automática el módulo de ejecución más eficiente. Se construyeron tablas de tiempos de ejecución específicas para cada plataforma: estación de trabajo CUDA y Jetson TX2. Estas tablas alimentan las tres estrategias de calendarización, implementadas y descritas en el capítulo anterior, ya que permiten identificar, para cada punto de integración, qué módulo ofrece el mejor desempeño en relación a las dimensiones de la integral y a la plataforma utilizada.

Para crear dichas tablas se realizaron pruebas de tiempo en los cuatro módulos de ejecución (Secuencial, CUDA, OpenMP y AVX), realizando las mediciones desde el punto 1 hasta el punto 40. Todas las pruebas se efectuaron sin aplicar la extrapolación de Romberg, con el fin de evaluar el tiempo de ejecución de la cuadratura de Gauss-Kronrod en cada módulo sin el costo de tiempo computacional que agrega la extrapolación.

Las plataformas permitieron analizar el comportamiento del sistema bajo entornos heterogéneos, donde las capacidades de cómputo y los módulos disponibles varían significativamente. En particular, la Jetson TX2 no cuenta con soporte para instrucciones AVX, lo que representa un escenario ideal para observar cómo los calendarizadores se adaptan automáticamente ante la ausencia de un módulo específico.

Con los tiempos de ejecución obtenidos, se generaron las tablas de tiempo con estructura $N \cdot M$, donde N representa el número de puntos de integración (del 1 al 40) y M el número de módulos de ejecución disponibles. Estas tablas fueron construidas individualmente para cada plataforma (Estación de trabajo CUDA y Jetson TX2) y a su vez para cada dimensión (3D, 4D, 5D y 6D), con el fin de capturar las variaciones de desempeño asociadas tanto al crecimiento dimensional como a las diferencias arquitectónicas del *hardware*. En el caso de la Jetson TX2, las tablas diseñadas para esta plataforma no incluyen el módulo AVX.

Los calendarizadores utilizan esta información como entrada para determinar el módulo de ejecución más eficiente para cada punto. De este modo, el proceso de integración se optimiza.

A continuación, se muestra un fragmento de cada tabla construida. La Tabla 4.4 presenta las tablas de tiempo diseñadas para la estación de trabajo CUDA. Su estructura permite comparar, para un mismo número de puntos, el desempeño de los módulos de ejecución (SEC, CUDA, OMP-E, OMP-P y AVX). La primera columna indica el número de puntos y las demás columnas indican los módulos de ejecución

empleados. La tabla se divide en cuatro bloques, correspondientes a 3, 4, 5 y 6 dimensiones, cada bloque representa el fragmento de una tabla de tiempo. Dentro de cada bloque se muestran los tiempos obtenidos para cinco puntos de integración consecutivos (36–40). Esto permite identificar patrones importantes:

- El módulo CUDA es el más rápido, especialmente en altas dimensiones.
- Los módulos AVX y SEC muestran tiempos similares, ya que las operaciones vectoriales no son lo suficientemente explotadas en este caso.
- Los módulos OMP-E y OMP-P presentan una aceleración moderada respecto al módulo secuencial, pero inferior al módulo CUDA.
- La diferencia entre 3 y 6 dimensiones ilustra el crecimiento exponencial del costo computacional al aumentar la dimensionalidad.

Tabla 4.4: Fragmento de las tablas de tiempo (s) para alimentar los calendarizadores en estación de trabajo CUDA.

Puntos	SEC	CUDA	OMP-E	OMP-P	AVX
3 Dimensiones					
36	0.0081250	0.0005965	0.0011250	0.0007960	0.0027980
37	0.0058765	0.0006365	0.0011495	0.0008070	0.0027135
38	0.0039605	0.0006535	0.0011470	0.0008675	0.0030305
39	0.0102905	0.0006735	0.0011780	0.0008685	0.0031190
40	0.0129425	0.0006700	0.0012575	0.0009020	0.0033490
4 Dimensiones					
36	0.1081705	0.0063280	0.0201370	0.0144610	0.1051795
37	0.1230435	0.0083885	0.0217275	0.0157150	0.1173515
38	0.1318860	0.0086055	0.0235115	0.0168765	0.1338460
39	0.1460755	0.0092070	0.0253470	0.0182765	0.1454455
40	0.1614300	0.0094480	0.0271755	0.0196750	0.1615425
5 Dimensiones					
36	4.5701550	0.2528100	0.8296870	0.5664170	4.6040615
37	5.2374090	0.3386130	0.9451825	0.6310175	5.2247210
38	5.9664955	0.3576945	1.0418870	0.6998535	5.9626540
39	6.8116815	0.3928375	1.1676685	0.7778680	6.8369160
40	7.7493995	0.4155625	1.2766010	0.8599085	7.7826535
6 Dimensiones					
36	188.1915175	9.7744500	33.8972150	23.2697750	188.7730510
37	221.3589945	13.9541845	38.5375195	26.6600405	221.5038270
38	259.1308680	15.1307505	43.8080475	30.4321160	259.7708710
39	302.2779010	17.0622010	50.5999200	34.5655860	302.2110690
40	351.6011680	18.4122340	56.5778165	39.2335565	350.7724255

En conjunto, esta tabla cumple dos funciones fundamentales en el sistema:

1. Proveer los datos necesarios para que los calendarizadores puedan decidir qué módulo utilizar para cada punto, en la estación de trabajo CUDA.

2. Permitir analizar el comportamiento relativo de cada módulo en función del número de puntos, la dimensión del problema y los recursos de la plataforma.

La Tabla 4.5 presenta un fragmento de cada tabla de tiempo generada específicamente para la Jetson TX2. La organización de la tabla mantiene la misma estructura empleada para la Tabla 4.4, con la diferencia de que no incluye el módulo AVX, ya que la plataforma no soporta instrucciones vectoriales AVX. Esta organización facilitar la comparación entre plataformas.

- El módulo CUDA posee una aceleración con respecto al módulo SEC, pero de forma menos marcada que en la estación de trabajo CUDA, debido a la menor capacidad de cómputo de la GPU integrada en la Jetson TX2.
- Los módulos OMP-E y OMP-P muestran mejoras moderadas, expresando las limitaciones del procesador ARM en cómputo intensivo.
- El módulo SEC presenta tiempos altos en dimensiones elevadas, evidenciando el crecimiento exponencial del costo de la integral.
- La ausencia del módulo AVX modifica la distribución de tiempos respecto a la estación de trabajo, generando un escenario más restringido en términos de heterogeneidad.

Tabla 4.5: Fragmento de las tablas de tiempo (s) para alimentar los calendarizadores en Jetson TX2.

Puntos	SEC	CUDA	OMP-E	OMP-P
3 Dimensiones				
36	0.0283380	0.0822570	0.0147250	0.0287100
37	0.0308645	0.0843120	0.0164210	0.0313865
38	0.0332555	0.0908845	0.0171880	0.0335250
39	0.0361240	0.0863445	0.0191180	0.0362610
40	0.0387175	0.0837800	0.0199990	0.0390635
4 Dimensiones				
36	1.3844420	0.1363845	0.5233365	1.3850520
37	1.5470000	0.1641760	0.6049235	1.5471000
38	1.7177325	0.1738900	0.6496380	1.7177350
39	1.9148380	0.1883555	0.7182320	1.9102840
40	2.1073190	0.1852710	0.7936845	2.1065165
5 Dimensiones				
36	63.5042890	4.0377040	22.0294770	63.3678255
37	72.8950990	5.3386920	25.3700260	72.7052670
38	83.5235035	5.5025570	29.4548390	83.3140820
39	95.0809615	6.0663860	32.7650515	94.8652030
40	107.4584325	6.3487945	36.9740870	107.3620445
6 Dimensiones				
36	2798.6301325	155.2763245	940.2878875	2795.7039895
37	3286.2258590	218.5440930	1147.9061050	3278.4770255
38	3872.0503585	237.0778720	1333.7556295	3859.5540160
39	4524.1764865	269.5824400	1513.0280615	4509.6641820
40	5250.9282605	290.8073185	1761.0665985	5241.1169100

Los propósitos de esta tabla son:

1. Proveer los tiempos de ejecución necesarios para que los calendarizadores puedan decidir qué módulo utilizar para cada punto, considerando únicamente los módulos disponibles en la Jetson TX2. De este modo, el sistema se adapta automáticamente a las limitaciones de *hardware* sin intervención del usuario.

2. Permitir analizar el impacto del entorno embebido sobre el rendimiento de la biblioteca de integrales multidimensionales.

Estos fragmentos representan sólo una parte de las tablas completas almacenadas para el sistema de calendarización, las cuales contienen los tiempos de ejecución para todos los puntos de integración requeridos (1–40).

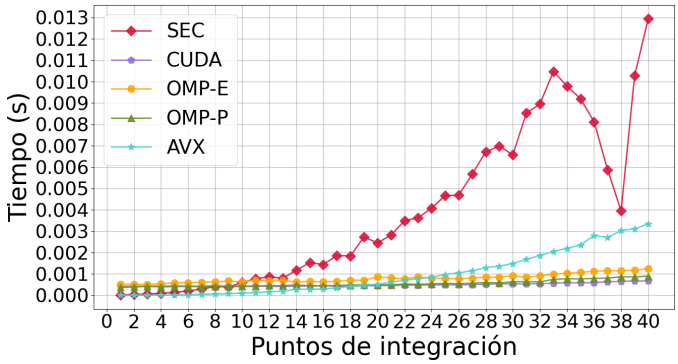
Las Figuras 4.2 presentan las curvas de tiempo de ejecución obtenidas al evaluar la Función 1.3 utilizando los módulos de ejecución disponibles en la estación de trabajo CUDA. Todas las gráficas fueron generadas a partir de las tablas de tiempos completas, por lo que reflejan el comportamiento real de cada módulo para todos los puntos de integración considerados.

Las figuras organizan los resultados por dimensión: 3D (Figura 4.2a), 4D (Figura 4.2b), 5D (Figura 4.2c) y 6D (Figura 4.2d), permitiendo observar de manera clara cómo crece el tiempo de ejecución conforme aumentan los puntos de integración y las dimensiones. A partir de las figuras, se observa que:

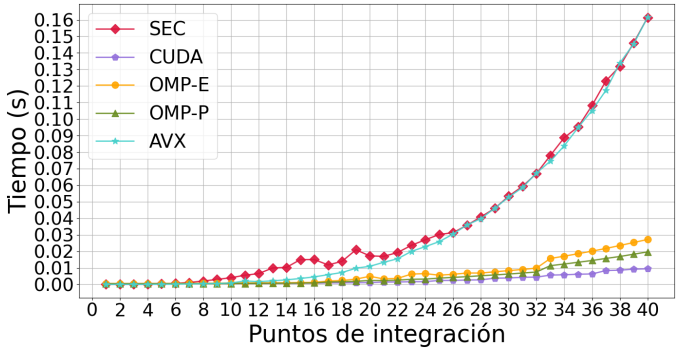
- El módulo CUDA presenta los menores tiempos conforme el número de puntos aumenta en todas las dimensiones, evidenciando la ventaja de una GPU de alto rendimiento cuando se trata de cargas altamente paralelizables.
- El módulo AVX presenta un desempeño superior al del módulo SEC en 3D (Figura 4.2a); sin embargo, a medida que aumenta la dimensionalidad, su rendimiento se aproxima al del módulo SEC y deja de ofrecer mejoras apreciables. Esto se debe a que su implementación no es completamente paralelizable.
- Los módulos OMP-E y OMP-P alcanzan mejoras moderadas respecto los módulos SEC y AVX, pero no logran superar al módulo CUDA.
- El módulo SEC exhibe el crecimiento de tiempo más pronunciado.

Por otro lado, las Figuras 4.3 muestran el comportamiento de los módulos de ejecución disponibles en la Jetson TX2. La organización es la misma: 3D (Figura 4.3a), 4D (Figura 4.3b), 5D (Figura 4.3c) y 6D (Figura 4.3d). En las figuras se observa que:

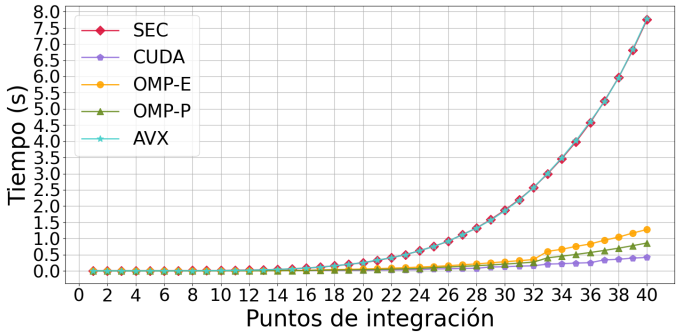
- El módulo CUDA posee los tiempos de ejecución más grandes en 3D (Figura 4.3a), pero ofrece una aceleración notable respecto al módulo SEC en las demás dimensiones, no obstante, la separación entre ambas curvas es significativamente menor que en la estación de trabajo. Esto refleja la diferencia entre una GPU integrada (Jetson TX2) y una GPU dedicada de escritorio.



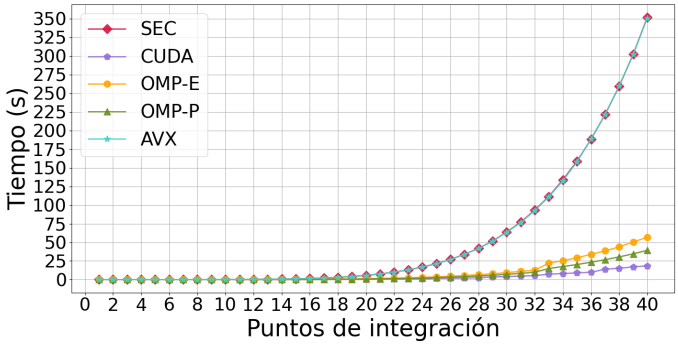
(a) 3D



(b) 4D

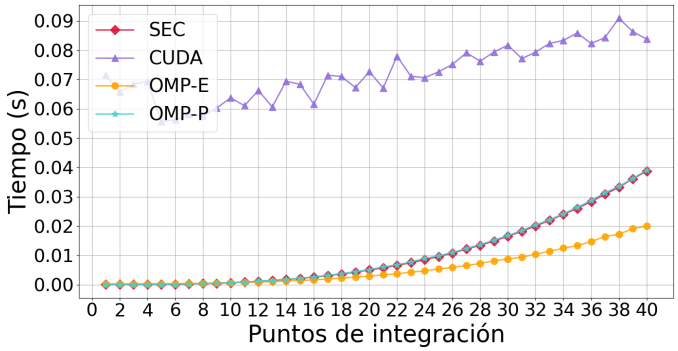


(c) 5D

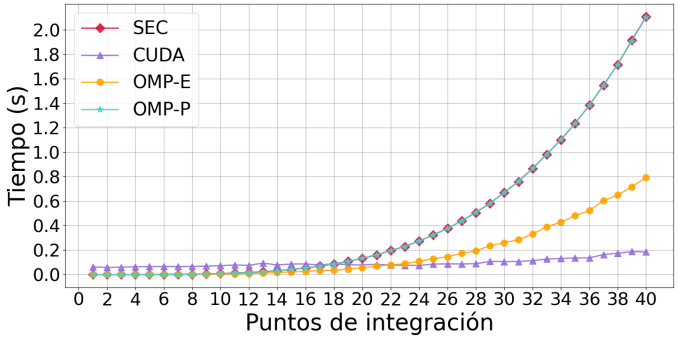


(d) 6D

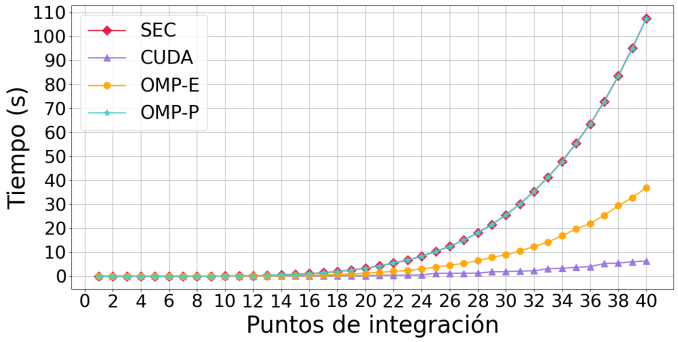
Figura 4.2: Comparación de tiempos de ejecución (s) al evaluar la Función 1.3 en estación de trabajo CUDA.



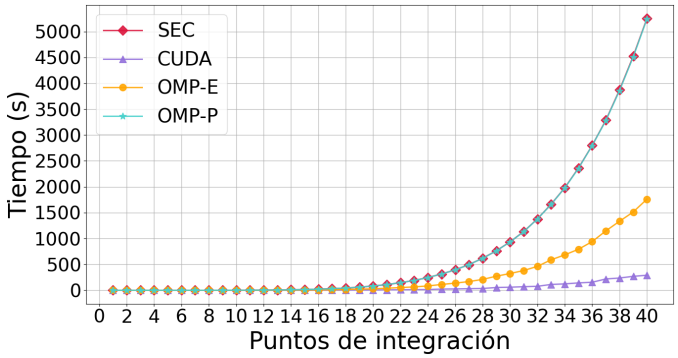
(a) 3D



(b) 4D



(c) 5D



(d) 6D

Figura 4.3: Comparación de tiempos de ejecución (s) al evaluar la Función 1.3 en Jetson TX2.

- El módulo OMP-P no proporcionan mejoras significativas, su desempeño es bastante cercano al módulo SEC, debido a las restricciones del procesador ARM en tareas de cómputo intensivo con núcleos de rendimiento.
- El módulo OMP-E presenta el mejor desempeño en 3D (Figura 4.3a) y un desempeño superior al de los módulos OMP-P y SEC, pero sin superar el módulo CUDA, en las demás dimensiones.
- El tiempo del módulo SEC crece rápidamente con el número de dimensiones, volviéndose demasiado grande en 5D (Figura 4.3c) y 6D (Figura 4.3d).

En todas las gráficas, tanto en la estación de trabajo como en la Jetson TX2, se observa un patrón común: el tiempo de ejecución crece de forma acelerada a medida que aumentan los puntos de integración y las dimensiones. Las curvas permiten identificar visualmente las diferencias entre plataformas y muestran cómo la disponibilidad o ausencia de módulos especializados (como CUDA o AVX) condiciona el rendimiento general del sistema de calendarización.

4.3.2 Pruebas de tiempo de ejecución usando el calendarizador estático secuencial

En esta sección se presentan los resultados obtenidos al realizar las pruebas con el calendarizador estático secuencial (SCHED SS). Las mediciones se realizaron en ambas plataformas, utilizando la Función 1.3 y el rango de 1 a 40 puntos de integración, aplicando la extrapolación de Romberg.

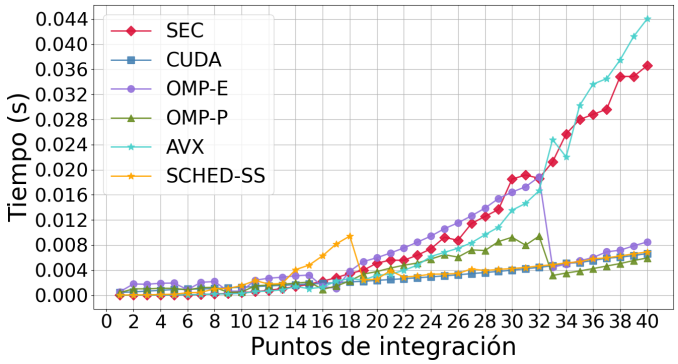
El objetivo de estas pruebas es cuantificar el tiempo total de ejecución y analizar el comportamiento del sistema de calendarización, contrastando el desempeño individual de cada módulo de ejecución con el obtenido por el calendarizador estático secuencial.

Las Figuras 4.4 y 4.5 muestran los tiempos de ejecución al evaluar la Función 1.3 en la estación de trabajo CUDA y en la Jetson TX2.

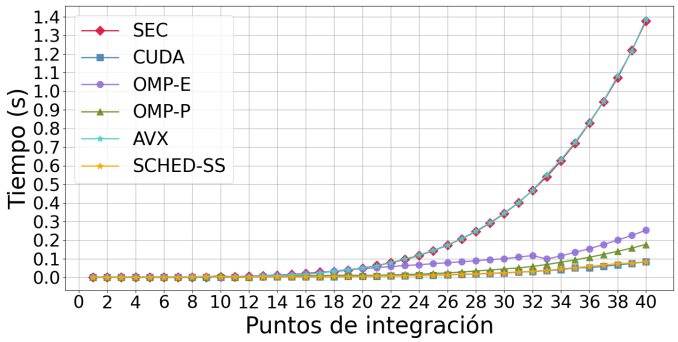
Las Figuras 4.4, correspondientes a los tiempos de ejecución en la estación de trabajo CUDA, organizan las gráficas por dimensión: 3D (Figura 4.4a), 4D (Figura 4.4b), 5D (Figura 4.4c) y 6D (Figura 4.4d), lo que facilita analizar cómo se comporta el calendarizador en distintas dimensiones. A partir de esto, se observa que:

- El módulo CUDA continúa mostrando el mejor rendimiento individual, siendo la curva más baja en todas las dimensiones.
- El calendarizador SCHED-SS sigue de cerca el desempeño del módulo CUDA cuando la carga computacional crece considerablemente, lo cual ocurre principalmente en 5D (Figura 4.4c) y 6D (Figura 4.4d). Además, su desempeño es mejor comparado con los módulos SEC, AVX, OMP-P y OMP-E.

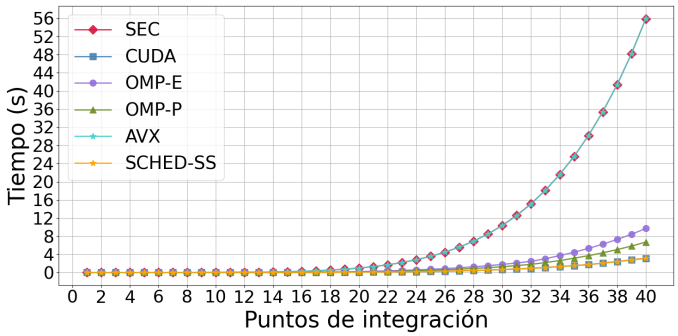
Las Figuras 4.5, organizadas por dimensión: 3D (Figura 4.5a), 4D (Figura 4.5b), 5D (Figura 4.5c) y 6D (Figura 4.5d), muestran los tiempos de ejecución en la Jetson TX2, donde el comportamiento cambia significativamente debido a las restricciones de la plataforma:



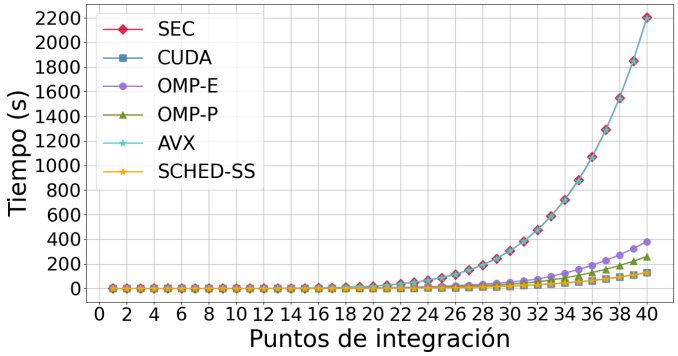
(a) 3D



(b) 4D

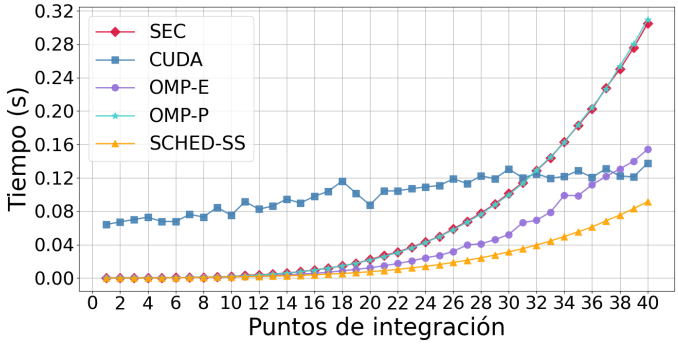


(c) 5D

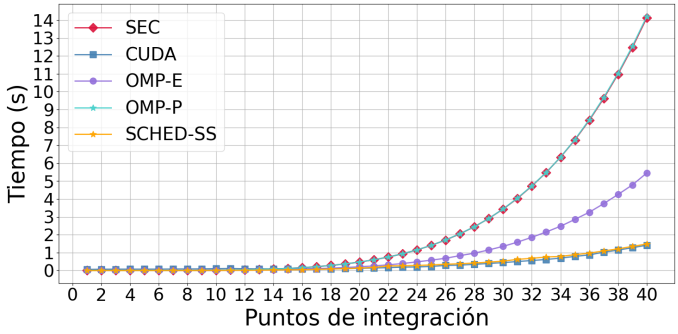


(d) 6D

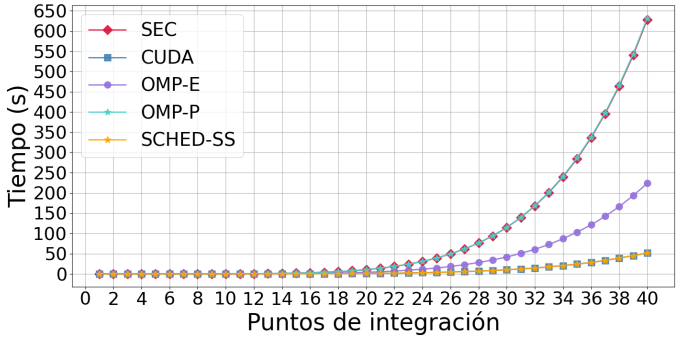
Figura 4.4: Comparación de tiempos de ejecución (s) al evaluar la Función 1.3 usando la extrapolación de Romberg: módulos VS calendarizador estático secuencial, en estación de trabajo CUDA.



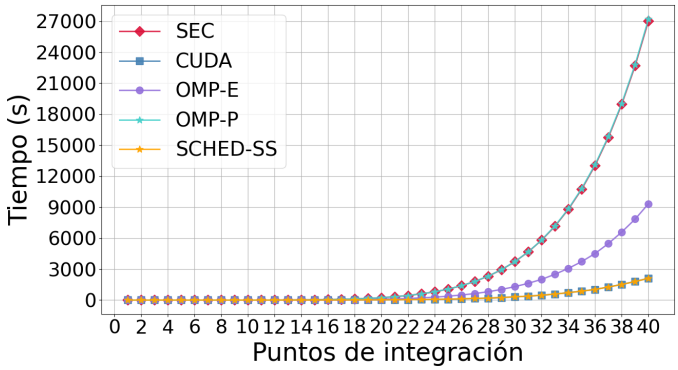
(a) 3D



(b) 4D



(c) 5D



(d) 6D

Figura 4.5: Comparación de tiempos de ejecución (s) al evaluar la Función 1.3 usando la extrapolación de Romberg: módulos VS calendarizador estático secuencial, en Jetson TX2.

- El módulo CUDA continúa siendo más eficiente que los módulos que usan la CPU en 4D(Figura 4.5b), 5D (Figura 4.5c) y 6D (Figura 4.5d), sin embargo continua teniendo tiempos de ejecución elevados en 3D (Figura 4.5a).
- El calendarizador SCHED-SS se ajusta al rendimiento de la Jetson, combinando los módulos disponibles. No obstante, su desempeño queda limitado por la menor capacidad de procesamiento de la CPU ARM y de la GPU integrada.
- En 3D (Figura 4.5a) el calendarizador SCHED-SS posee el menor tiempo de ejecución, cumpliendo el objetivo de reducir el tiempo a través de una estrategia de calendarización.
- En 5D (Figura 4.5c) y 6D (Figura 4.5d), el calendarizador SCHED-SS mantiene un mejor desempeño en comparación con el módulo SEC, pero sigue de cerca el rendimiento del módulo CUDA.

Estas figuras resultan fundamentales para validar el funcionamiento del calendarizador estático secuencial, ya que permiten comparar la estrategia de asignación con los tiempos de cada módulo individual y muestran cómo el rendimiento del sistema de calendarización surge de la adecuada combinación de los recursos disponibles.

La Tabla 4.6 presenta los tiempos de ejecución obtenidos con el calendarizador estático secuencial en un rango de 36 a 40 puntos. Estos valores corresponden únicamente a un subconjunto de puntos de integración y a las cuatro dimensiones evaluadas (3D–6D), seleccionados con el propósito de ilustrar de manera compacta el comportamiento del calendarizador en ambas plataformas: la estación de trabajo CUDA y la Jetson TX2.

Todos los datos mostrados en esta tabla provienen directamente de las mismas mediciones utilizadas para construir las gráficas de las Figuras 4.4 y 4.5. En las figuras, los valores de tiempo se representan de forma visual para todos los puntos evaluados (1–40), mientras que la tabla ofrece una vista concentrada que permite observar de forma más clara la tendencia del calendarizador para un intervalo específico de puntos de integración (36–40).

La tabla sirve como referencia numérica directa del comportamiento que aparece representado en las curvas del calendarizador SCHED-SS dentro de las gráficas, mostrando cómo el tiempo de ejecución crece conforme aumenta el número de puntos de integración y la dimensionalidad del dominio.

En general, las pruebas confirman que esta estrategia es adecuada para entornos donde las condiciones de ejecución son estables y el costo de calendarización previa se compensa con la reducción del tiempo total de cómputo.

4.3.3 Pruebas de tiempo de ejecución usando el calendarizador dinámico secuencial

Los resultados obtenidos usando el calendarizador dinámico secuencial (SCHED DS) muestran que este enfoque logra un desempeño muy similar al del calendarizador estático secuencial en condiciones estables, con una ligera penalización en tiempo

Tabla 4.6: Pruebas realizadas en calendarizador estático secuencial usando la Función 1.3 y la extrapolación de Romberg. Tiempo de ejecución (s)

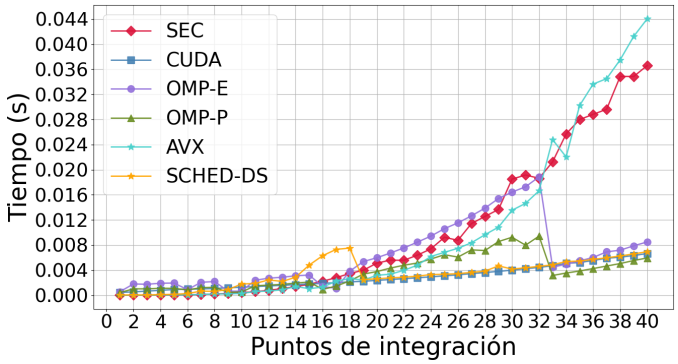
Puntos	Estación de trabajo CUDA	Jetson TX2
3 Dimensiones		
36	0.0058130	0.0611985
37	0.0059520	0.0684210
38	0.0062590	0.0754485
39	0.0065900	0.0831530
40	0.0069075	0.0913880
4 Dimensiones		
36	0.0583350	0.9736850
37	0.0661450	1.0948180
38	0.0746965	1.2114875
39	0.0772280	1.3468675
40	0.0838085	1.4856400
5 Dimensiones		
36	1.7585130	28.1786615
37	2.0808835	33.4708585
38	2.4203720	39.0795090
39	2.7959825	45.2268855
40	3.1883440	51.7129405
6 Dimensiones		
36	64.2720825	1029.1366955
37	78.3937330	1261.2886605
38	93.7072915	1512.0529185
39	111.0044350	1796.3995345
40	129.6988870	2104.5626530

debido al proceso de comparación que se realiza en cada iteración. Sin embargo, dicha sobrecarga es mínima y se mantiene constante a lo largo de toda la ejecución.

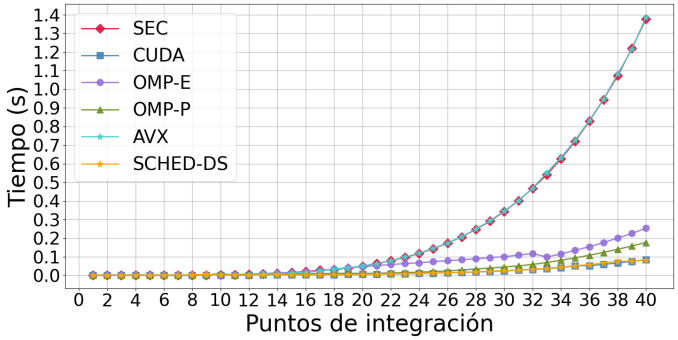
Las Figuras 4.6 y 4.7 muestran la comparación de los tiempos de ejecución en la estación de trabajo CUDA y en la Jetson TX2, contrastando los módulos individuales frente al calendarizador dinámico secuencial.

En la estación de trabajo CUDA (Figuras 4.6), donde las gráficas están ordenadas por dimensión: 3D (Figura 4.6a), 4D (Figura 4.6b), 5D (Figura 4.6c) y 6D (Figura 4.6d), el calendarizador SCHED-DS tiende a comportarse de manera similar al módulo CUDA, el cual tiene los menores tiempos de ejecución en cada dimensión. No obstante, en 3D (Figura 4.6a) presenta ligeras irregularidades, debido a la variabilidad introducida por la asignación dinámica. A partir de 4D (Figura 4.6b) y en adelante, la curva del calendarizador SCHED-DS converge hacia una tendencia suave, dominada por el costo acumulado de las evaluaciones.

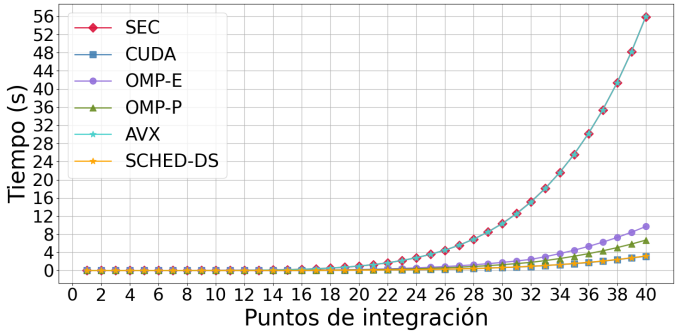
En la Jetson TX2 (Figuras 4.7), donde las gráficas están ordenadas por dimensión: 3D (Figura 4.7a), 4D (Figura 4.7b), 5D (Figura 4.7c) y 6D (Figura 4.7d), el comportamiento del calendarizador SCHED-DS en 3D (Figura 4.7a) es mejor en comparación con los módulos individuales, cumpliendo el objetivo de reducir el tiempo.



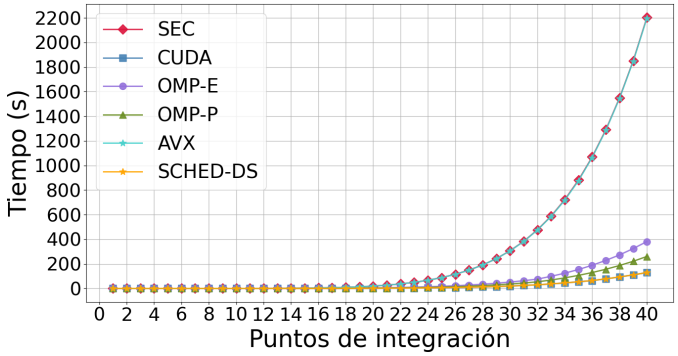
(a) 3D



(b) 4D

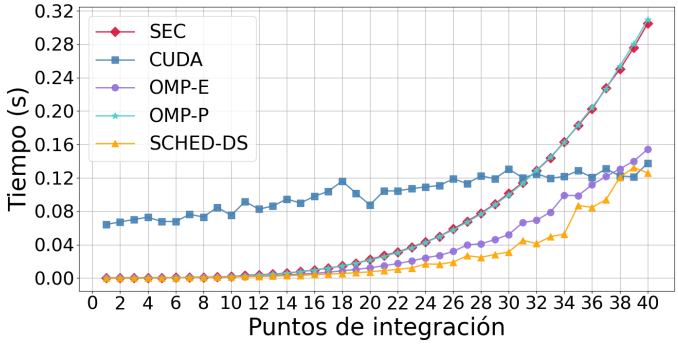


(c) 5D

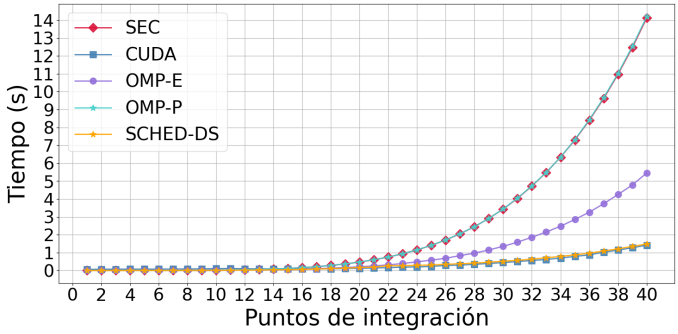


(d) 6D

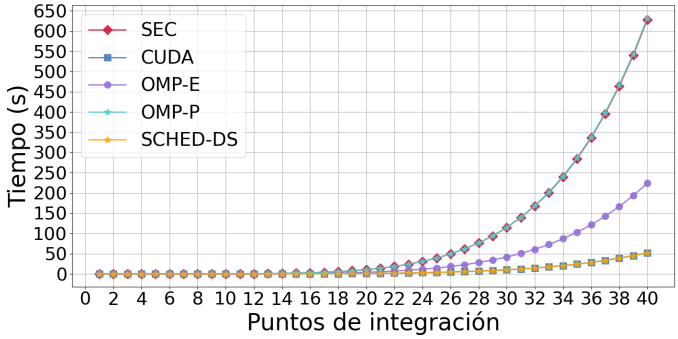
Figura 4.6: Comparación de tiempos de ejecución (s) al evaluar la Función 1.3 usando la extrapolación de Romberg: módulos VS calendarizador dinámico secuencial, en estación de trabajo CUDA.



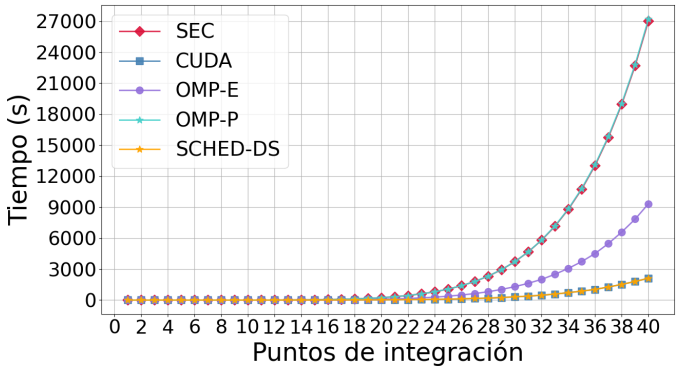
(a) 3D



(b) 4D



(c) 5D



(d) 6D

Figura 4.7: Comparación de tiempos de ejecución (s) al evaluar la Función 1.3 usando la extrapolación de Romberg: módulos VS calendarizador dinámico secuencial, en Jetson TX2.

Sin embargo, en dimensiones altas (5D (Figura 4.7c) y 6D (Figura 4.7d)), la curva del calendarizador SCHED-DS muestra un desempeño similar al módulo CUDA, sin agregar mejoras.

La Tabla 4.7 presenta un fragmento de los tiempos de ejecución obtenidos con el calendarizador dinámico secuencial. Estos valores corresponden a los mismos datos que se utilizaron para generar las gráficas mostradas en las Figuras 4.6 y 4.7; sin embargo, aquí se resume únicamente el intervalo de puntos de integración de 36 a 40, con el fin de observar claramente el comportamiento del calendarizador en ambas plataformas: la estación de trabajo CUDA y la Jetson TX2.

La estructura de la tabla se organiza por bloques, uno por cada dimensión del problema (3D, 4D, 5D y 6D), permitiendo comparar directamente el crecimiento del tiempo de ejecución conforme aumenta el número de puntos de integración y la dimensionalidad del dominio.

En conjunto, esta tabla ofrece una vista sintetizada pero representativa del desempeño del calendarizador dinámico, permitiendo relacionar cuantitativamente los valores que se compararon visualmente en las figuras.

Tabla 4.7: Pruebas realizadas en calendarizador dinámico secuencial usando la Función 1.3 y la extrapolación de Romberg. Tiempo de ejecución (s)

Puntos	Estación de trabajo CUDA	Jetson TX2
3 Dimensiones		
36	0.0057150	0.0846915
37	0.0059625	0.0937825
38	0.0062490	0.1205385
39	0.0065820	0.1324245
40	0.0069380	0.1258805
4 Dimensiones		
36	0.0577000	0.9707205
37	0.0672020	1.0892880
38	0.0754635	1.2144655
39	0.0771860	1.3475005
40	0.0824145	1.4838455
5 Dimensiones		
36	1.8011205	28.1653055
37	2.0921675	33.4641705
38	2.4201670	39.0657690
39	2.7949995	45.2000950
40	3.1875860	51.7072995
6 Dimensiones		
36	64.2837270	1033.4060260
37	78.3934990	1265.6574855
38	93.7225400	1517.8290865
39	111.0416065	1804.1610285
40	129.7061360	2113.3271530

En términos generales, las pruebas confirman que el calendarizador dinámico secuencial constituye una estrategia flexible y confiable, capaz de mantener una distri-

bución eficiente de las tareas con un costo computacional adicional despreciable, lo que lo convierte en una alternativa viable cuando se busca adaptabilidad sin comprometer el rendimiento global.

4.3.4 Pruebas de tiempo de ejecución usando el calendarizador estático concurrente

El calendarizador estático concurrente (SCHED SC) fue diseñado con el propósito de aprovechar la concurrencia a nivel de hilos, evaluando dos puntos consecutivos de la integral multidimensional en distintos módulos con espera activa, es decir, siempre que los módulos correspondientes estuvieran disponibles. Sin embargo, durante las pruebas se observó que este enfoque no logró mejorar los tiempos de ejecución con respecto a los calendarizadores secuenciales. Por el contrario, en la mayoría de los casos el rendimiento fue inferior, especialmente en las configuraciones de mayor dimensión.

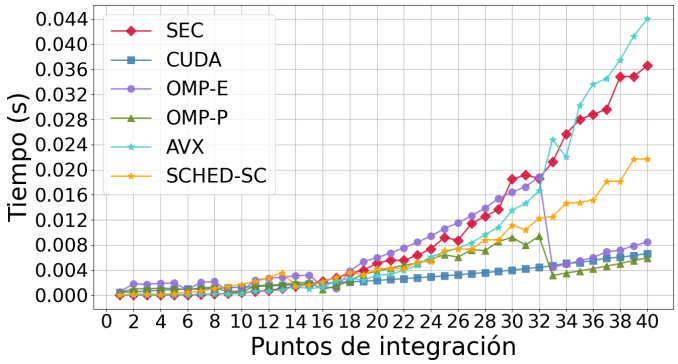
Las Figuras 4.8 y 4.9 presentan los tiempos de ejecución obtenidos al usar el calendarizador estático concurrente, tanto en la estación de trabajo CUDA como en la Jetson TX2. Estas gráficas permiten analizar el desempeño de dicho calendarizador frente a los módulos individuales.

El desempeño en la estación de trabajo CUDA (Figuras 4.8), donde las gráficas están ordenadas por dimensión: 3D (Figura 4.8a), 4D (Figura 4.8b), 5D (Figura 4.8c) y 6D (Figura 4.8d), muestra que:

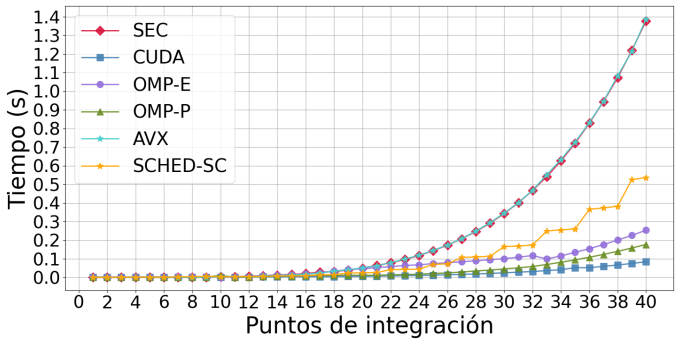
- El calendarizador SCHED-SC supera en rendimiento a los módulos SEC y AVX. Esto se debe a que la concurrencia entre dos módulos de ejecución permite repartir la carga de forma más equilibrada.
- Los módulos CUDA, OMP-P y OMP-E tienen mejores tiempos de ejecución a diferencia del calendarizador SCHED-SC, debido al overhead inherente a la sincronización entre módulos concurrentes y a las restricciones de exclusión mutua del calendarizador, lo que incrementa ligeramente su tiempo respecto a los módulos paralelos individuales.

En la Jetson TX2 (Figuras 4.9), donde las gráficas están ordenadas por dimensión: 3D (Figura 4.9a), 4D (Figura 4.9b), 5D (Figura 4.9c) y 6D (Figura 4.9d), los resultados muestran que:

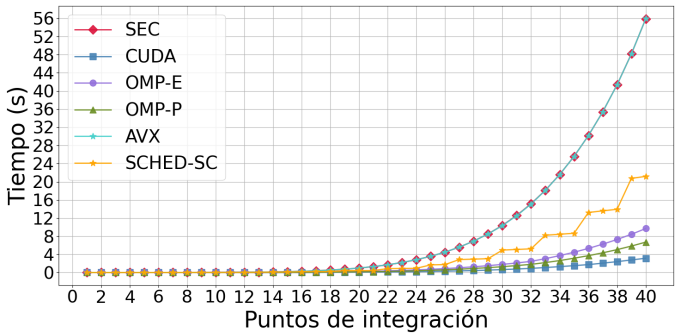
- En 3D (Figura 4.9a) el calendarizador SCHED-SC muestra un bajo desempeño, siguiendo de cerca el comportamiento de los módulos SEC y OMP-P.
- En 4D (Figura 4.9b), 5D (Figura 4.9c) y 6D (Figura 4.9d), donde el costo computacional se incrementa considerablemente, el calendarizador SCHED-SC supera a los módulos SEC y OMP-P y en algunos casos, al módulo OMP-E.
- El módulo CUDA continua siendo el módulo con los mejores tiempos de ejecución, el calendarizador SCHED-SC no logra superarlo, la separación entre ambas curvas es significativamente grande.



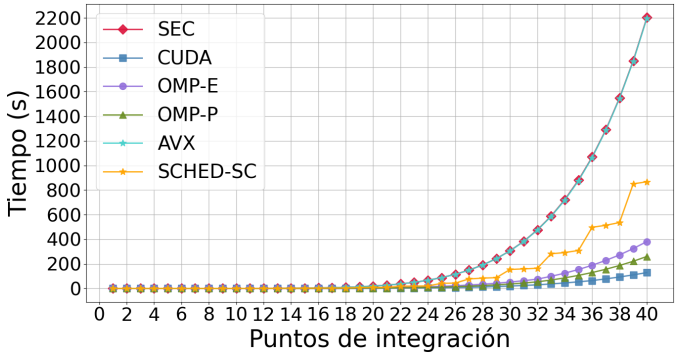
(a) 3D



(b) 4D

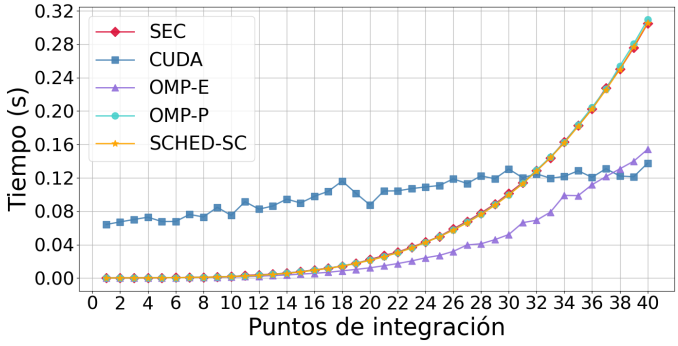


(c) 5D

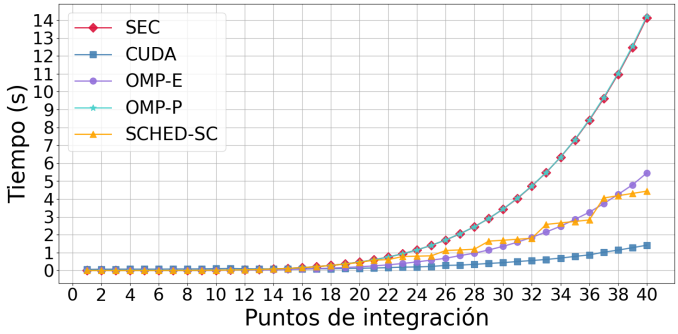


(d) 6D

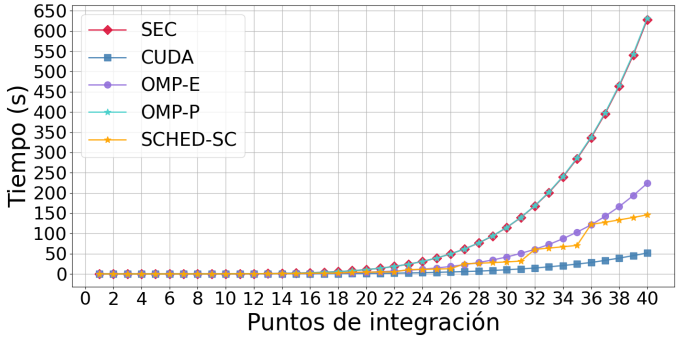
Figura 4.8: Comparación de tiempos de ejecución (s) al evaluar la Función 1.3 usando la extrapolación de Romberg: módulos VS calendarizador estático concurrente, en estación de trabajo CUDA.



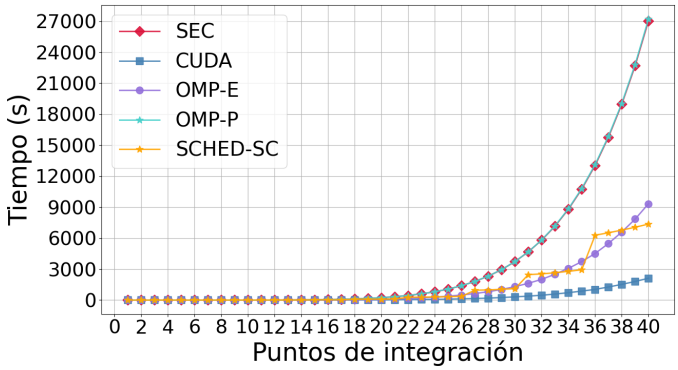
(a) 3D



(b) 4D



(c) 5D



(d) 6D

Figura 4.9: Comparación de tiempos de ejecución (s) al evaluar la Función 1.3 usando la extrapolación de Romberg: módulos VS calendarizador estático concurrente, en Jetson TX2.

La Tabla 4.8 presenta un fragmento representativo de los tiempos de ejecución obtenidos con el calendarizador estático concurrente, tanto en la estación de trabajo CUDA como en la Jetson TX2. Estos valores corresponden a los mismos datos que se utilizaron para generar las gráficas mostradas en las Figuras 4.8 y 4.9.

El objetivo de esta tabla es mostrar de manera estructurada cómo se comporta el calendarizador cuando combina dos módulos de ejecución de forma concurrente para resolver las integrales multidimensionales. Los resultados se presentan para cinco puntos de integración consecutivos (36–40), permitiendo observar la tendencia del tiempo de cómputo conforme aumenta la carga y crece la complejidad del problema en 3D, 4D, 5D y 6D.

Tabla 4.8: Pruebas realizadas en calendarizador estático concurrente usando la Función 1.3 y la extrapolación de Romberg. Tiempo de ejecución (s)

Puntos	Estación de trabajo CUDA	Jetson TX2
3 Dimensiones		
36	0.0151475	0.2023605
37	0.0181585	0.2254015
38	0.0181555	0.2499150
39	0.0216875	0.2771290
40	0.0216970	0.3054160
4 Dimensiones		
36	0.3653950	2.8283780
37	0.3733455	4.0554185
38	0.3819945	4.1832000
39	0.5252940	4.3109860
40	0.5363515	4.4378590
5 Dimensiones		
36	13.2290310	122.3459580
37	13.5906915	127.6760570
38	13.9398545	133.4473490
39	20.7490085	139.5041090
40	21.1865920	145.8986720
6 Dimensiones		
36	497.7480595	6279.4874210
37	513.7554920	6513.4717390
38	536.2696760	6775.9281455
39	850.6828590	7049.9691395
40	867.9491250	7359.7395700

Capítulo 5

Análisis

En este capítulo se presenta el análisis detallado de los resultados obtenidos a partir de las pruebas descritas en el capítulo anterior. El objetivo es evaluar el desempeño de los distintos módulos de ejecución, así como la efectividad de las estrategias de calendarización implementadas para la integración multidimensional. El análisis se centra en comparar la eficiencia, escalabilidad y adaptabilidad del sistema de calendarización en las plataformas heterogéneas utilizadas: una estación de trabajo CUDA y una Jetson TX2.

La primera sección compara el cálculo de las integrales multidimensionales utilizando los módulos de ejecución con 128 puntos sin extrapolación frente al método de extrapolación de Romberg aplicado con sólo 40 puntos. En esta parte se analiza la reducción del tiempo de ejecución y el impacto en la precisión del resultado, destacando cómo la extrapolación permite mantener errores dentro de márgenes aceptables con un costo computacional significativamente menor.

La segunda sección aborda el desempeño de los calendarizadores propuestos. Se realiza una comparación general entre las tres estrategias: estático secuencial, dinámico secuencial y estático concurrente, evaluando sus tiempos de ejecución en ambas plataformas y resaltando sus ventajas y limitaciones. Finalmente, se analiza el impacto de las estrategias de calendarización frente a la ejecución individual de los módulos, considerando las aceleraciones obtenidas y su comportamiento frente a la heterogeneidad de las plataformas.

En conjunto, este capítulo permite comprender la relación entre la complejidad del método, la carga computacional y la arquitectura de ejecución, ofreciendo una visión integral del desempeño del sistema de calendarización implementado.

5.1 Comparación: 128 puntos vs extrapolación de Romberg

Las pruebas preliminares tuvieron como propósito evaluar la precisión y el comportamiento de la biblioteca de integrales multidimensionales antes de aplicar las estrategias de calendarización. Para este fin, se compararon dos enfoques distintos para el

cálculo de las integrales: el primero utilizando 128 puntos de integración sin aplicar la extrapolación de Romberg, y el segundo reduciendo el número de puntos a 40, incorporando dicho método.

El enfoque con 128 puntos de integración utiliza únicamente la cuadratura de Gauss-Kronrod, la cual garantiza una mayor precisión numérica, a diferencia de otras cuadraturas. Sin embargo, este enfoque presenta un mayor tiempo de ejecución, debido al elevado número de puntos. En contraste, el método con 40 puntos emplea, además de la cuadratura de Gauss-Kronrod, la extrapolación de Romberg, una técnica que permite disminuir el número de puntos. Esto mejora los tiempos de ejecución y mantiene un error aceptable dentro de los margenes establecidos en esta tesis.

Como se muestra en las Tablas 4.2 y 4.3 (Sección 4.2.2), al reducir el número de puntos e incorporar la extrapolación de Romberg se obtiene un error ligeramente superior, 10^{-4}, en comparación con el error de 10^{-6} obtenido al integrar con 128 puntos. No obstante, esta diferencia se considera aceptable, debido a que la disminución en el número de puntos produce una reducción notable en el tiempo de ejecución. En particular, el tiempo requerido para la integración usando la extrapolación de Romberg corresponde sólo a una fracción del necesario para la versión de 128 puntos, lo que representa una mejora significativa en eficiencia computacional sin comprometer en exceso la exactitud del resultado.

Para evaluar el impacto de la extrapolación de Romberg en el desempeño de los módulos de ejecución, se calcularon las aceleraciones relativas (Ecuación 5.1) de cada módulo respecto al módulo CUDA evaluando 128 puntos, ya que éste representa la referencia más rápida dentro de los módulos con 128 puntos.

$$A = \frac{T_{CUDA,\,128}}{T_{MOD,\,R40}}, \tag{5.1}$$

donde $T_{CUDA,,128}$ corresponde al tiempo de ejecución del módulo CUDA al evaluar 128 puntos sin aplicar la extrapolación de Romberg, mientras que $T_{MOD,,R40}$ representa el tiempo de ejecución de los distintos módulos al evaluar 40 puntos con la extrapolación de Romberg. El término MOD hace referencia a los módulos de la biblioteca de integrales multidimensionales (Secuencial, CUDA, OpenMP y AVX).

El valor del parámetro A permite cuantificar el cambio en el rendimiento. Si $A > 1$, el módulo presenta aceleración respecto al caso base; si $A = 1$, no se observa variación en el tiempo de ejecución; y si $A < 1$, se produce una desaceleración, lo que indica que el nuevo método es menos eficiente.

La Tabla 5.1 presenta las aceleraciones obtenidas para las distintas dimensiones. La tabla se divide en dos secciones principales, correspondientes a las dos plataformas de prueba: estación de trabajo CUDA y Jetson TX2. Cada fila de la tabla corresponde a un módulo de ejecución, mientras que cada columna representa una dimensión del problema (3D, 4D, 5D y 6D). De esta forma, las celdas contienen los valores numéricos de aceleración alcanzados en cada caso.

Tabla 5.1: Aceleraciones de los módulos de ejecución usando la extrapolación de Romberg tomando como referencia el módulo de ejecución CUDA con 128 puntos.

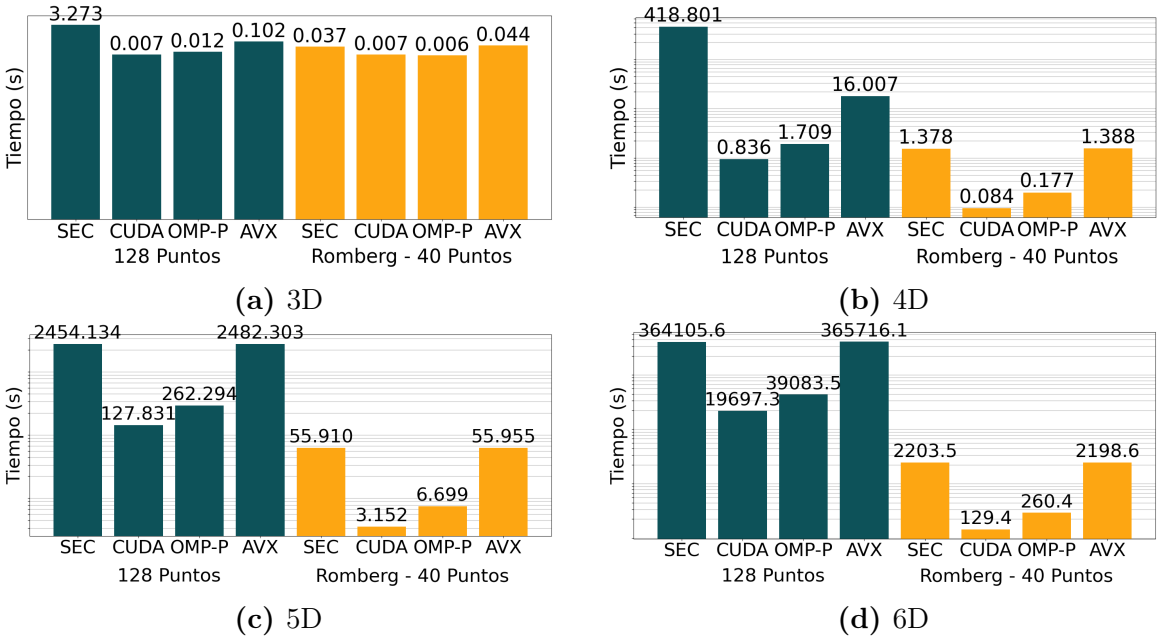
Módulos	3D	4D	5D	6D
Estación de trabajo CUDA				
SEC	0.187	0.606	2.286	8.938
CUDA	1.034	9.924	40.558	152.259
OMP-P	0.809	3.296	13.181	51.595
OMP-E	1.156	4.718	19.083	75.638
AVX	0.156	0.602	2.284	8.958
Jetson TX2				
SEC	0.524	0.928	3.473	13.447
CUDA	1.163	9.259	42.207	172.011
OMP-P	1.037	2.404	9.746	38.980
OMP-E	0.516	0.923	3.453	13.364

De acuerdo con los datos presentados en la Tabla 5.1, en la estación de trabajo CUDA se observa que el módulo de ejecución CUDA alcanza una aceleración superior a 150× en seis dimensiones. Asimismo, los módulos OMP-P y OMP-E presentan incrementos de rendimiento significativos, particularmente a partir de las cuatro dimensiones. Sin embargo, el módulo AVX, no alcanzó aceleraciones significativas y, en algunos casos, incluso presentó tiempos de ejecución mayores que el módulo SEC. Esto se debe a que las instrucciones AVX permiten paralelizar únicamente operaciones vectoriales a nivel de registro, es decir, procesar varios elementos en un mismo ciclo de reloj, pero no aprovechan múltiples núcleos, ni dispositivos de cómputo. Además, la sobrecarga asociada a la preparación de los datos y a la alineación de memoria puede anular los beneficios del procesamiento vectorial cuando el número de puntos es grande o las operaciones no son completamente vectorizables, como es el caso de la función a integrar (véase Sección 22). El módulo AVX resulta ventajoso sólo para integraciones de baja dimensionalidad o con un número reducido de puntos, donde el acceso a memoria y el tamaño del vector no penalizan el rendimiento del módulo.

Por otro lado, en la Jetson TX2 se observan tendencias similares: el módulo de ejecución CUDA presenta las mayores aceleraciones. No obstante, se identificó que el módulo OMP-E, que emplea exclusivamente los núcleos de eficiencia, obtuvo tiempos de ejecución inferiores a los del módulo OMP-P, el cual utiliza los núcleos de rendimiento. Esta tendencia se explica por las características particulares de la arquitectura heterogénea de la plataforma, donde los núcleos de eficiencia presentan un consumo energético y una frecuencia más estables bajo cargas prolongadas. En contraste, los núcleos de rendimiento tienden a operar a frecuencias más altas durante intervalos cortos, pero experimentan una rápida reducción de velocidad debido a la gestión térmica y las limitaciones de energía del sistema integrado. Como resultado, las ejecuciones paralelas en los núcleos de eficiencia mantienen un desempeño más constante y sostenido, mientras que el uso intensivo de los núcleos de rendimiento introduce variaciones y penalizaciones que aumentan el tiempo total de cómputo.

Las Figuras 5.1 y 5.2 muestran la comparación de los tiempos de ejecución obtenidos al evaluar la Función 1.3 en las dos plataformas de prueba: estación de trabajo CUDA y Jetson TX2. En ambos casos, se contrastan los resultados de los módulos de

ejecución al usar 128 puntos sin extrapolación frente a la extrapolación de Romberg con 40 puntos.



(a) 3D **(b)** 4D

(c) 5D **(d)** 6D

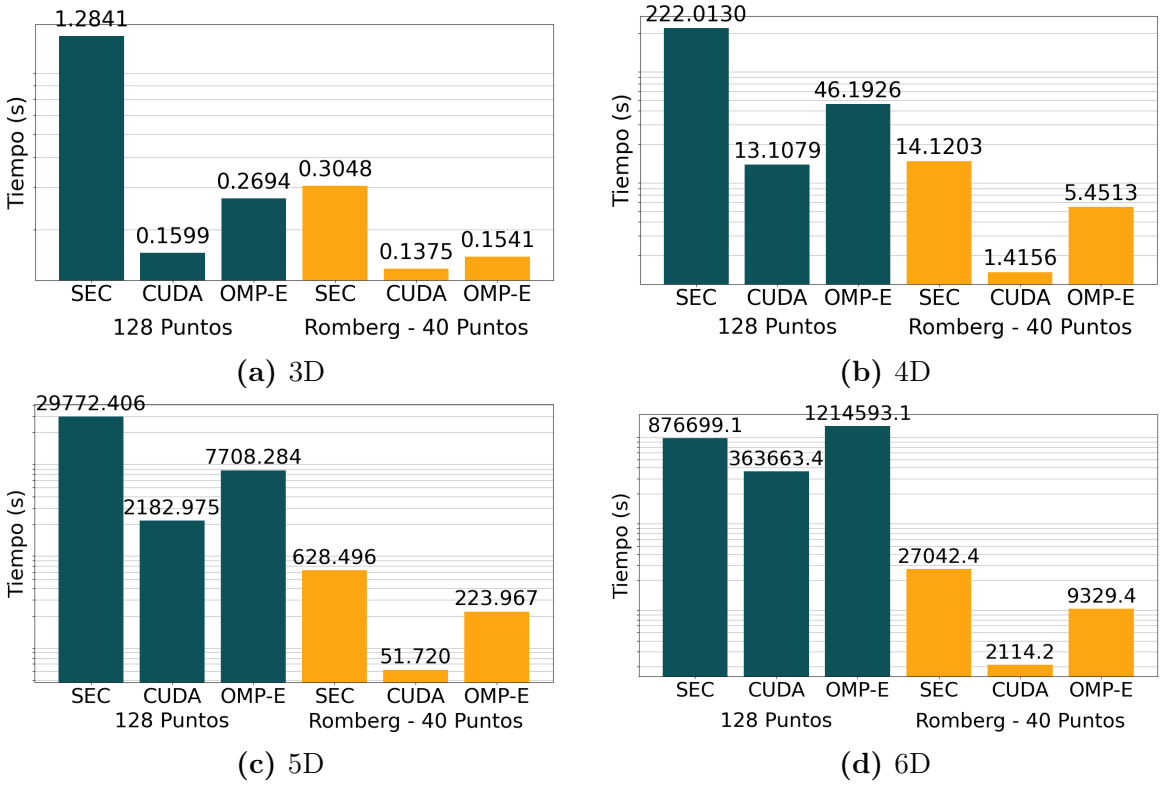
Figura 5.1: Comparación de tiempos de ejecución (s) al evaluar la Función 1.3: 128 puntos VS extrapolación de Romberg en estación de trabajo CUDA.

En la estación de trabajo CUDA (Figuras 5.1), las gráficas ordenadas por dimensión: 3D (Figura 5.1a), 4D (Figura 5.1b), 5D (Figura 5.1c) y 6D (Figura 5.1d), indican que la extrapolación de Romberg reduce de manera notable los tiempos de ejecución para la mayoría de los módulos, en comparación con la integración realizada mediante los módulos con 128 puntos.

- En 3D (Figura 5.1a), los módulos SEC, OMP-P y AVX reducen su tiempo de ejecución al emplear extrapolación de Romberg, mientras que el módulo CUDA no presenta una disminución a considerar.
- En 4D (Figura 5.1b), el beneficio es aún más evidente: los módulos paralelos reducen su tiempo en un rango aproximado del 80 % al 90 %, destacando CUDA y OMP-P, que logran los menores tiempos absolutos.
- En 5D (Figura 5.1c) y 6D (Figura 5.1d), donde la carga computacional crece exponencialmente con la dimensionalidad, la extrapolación de Romberg permite mantener tiempos de ejecución controlados. Los módulos CUDA y OMP-P muestran una reducción superior al 90 %, lo que confirma la eficacia del método para disminuir el número de evaluaciones requeridas sin comprometer la precisión del resultado.

En general, la extrapolación de Romberg demostró ser una estrategia efectiva para acelerar los cálculos en entornos de alta capacidad computacional como la esta-

ción de trabajo CUDA, logrando reducciones notables en el tiempo de ejecución en comparación con el uso directo de 128 puntos.



(a) 3D **(b)** 4D

(c) 5D **(d)** 6D

Figura 5.2: Comparación de tiempos de ejecución (s) al evaluar la Función 1.3: 128 puntos VS extrapolación de Romberg en Jetson TX2.

En la Jetson TX2 (Figuras 5.2), las gráficas ordenadas por dimensión: 3D (Figura 5.2a), 4D (Figura 5.2b), 5D (Figura 5.2c) y 6D (Figura 5.2d), muestran que los tiempos de ejecución son mayores debido a las limitaciones del *hardware* embebido, pero la tendencia de mejora se mantiene en comparación con la integración realizada mediante los módulos con 128 puntos.

- En 3D (Figura 5.2a) y 4D (Figura 5.2b), los módulos CUDA y OMP-E reducen sus tiempos de ejecución entre un 70 % y 85 %, evidenciando que la extrapolación de Romberg también es beneficiosa en plataformas con menor número de núcleos o menor frecuencia de reloj.
- En 5D (Figura 5.2c) y 6D (Figura 5.2d), la diferencia entre ambas configuraciones se amplía considerablemente: el tiempo de ejecución del módulo CUDA, con 40 puntos y extrapolación de Romberg, es entre 10 y 20 veces menor que el módulo con 128 puntos.
- El módulo secuencial SEC también se ve beneficiado, aunque en menor proporción, ya que no aprovecha el paralelismo para distribuir las evaluaciones del integrando.

El análisis comparativo muestra que la extrapolación de Romberg reduce de manera significativa el tiempo de ejecución en todos los módulos de prueba, siendo especialmente eficiente en los módulos paralelos CUDA, OMP-P y OMP-E. Esta mejora es consistente tanto en la estación de trabajo como en la Jetson TX2, lo que demuestra que la estrategia es portable y efectiva en diferentes arquitecturas.

Caída en los tiempos de ejecución de los módulos OMP-E y OMP-P

Los tiempos de ejecución de los módulos al evaluar la integral en un rango de 1 a 40 puntos de integración, utilizando la extrapolación de Romberg para las dimensiones 3D (Figura 5.3a), 4D (Figura 5.3b), 5D (Figura 5.3c) y 6D (Figura 5.3d), mostraron en la estación de trabajo CUDA una caída en los tiempos de ejecución de los módulos OMP-E y OMP-P alrededor del punto 33 para las dimensiones 3D y 4D. En las Figuras 5.4a y 5.4b se puede observar mejor esta caída.

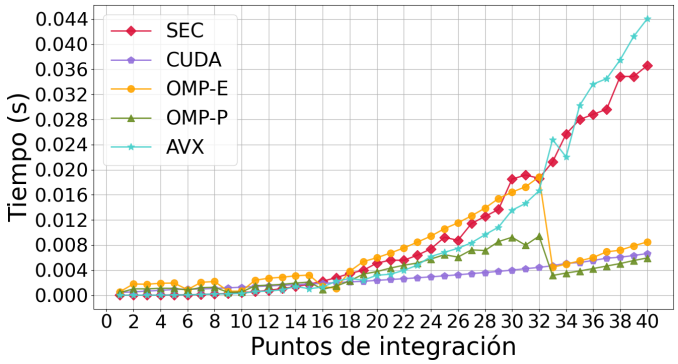
Este comportamiento no se manifestó en la Jetson TX2, lo que sugiere que está asociado a características propias de la arquitectura x86 de la estación de trabajo CUDA y del calendarizador del sistema operativo. Entre las posibles causas se encuentra:

- La arquitectura híbrida del procesador Intel, que combina núcleos de rendimiento (P-cores) y de eficiencia (E-cores);
- El *runtime* de OpenMP puede redistribuir dinámicamente los hilos entre estos núcleos en función de la carga y del número de iteraciones, generando cambios abruptos en el rendimiento.
- El compilador puede aplicar diferentes estrategias de vectorización y desenrollado de bucles, según el tamaño del conjunto de datos, activando en ciertos puntos versiones del código más eficientes.
- El mecanismo de Turbo Boost de Intel, que ajusta dinámicamente la frecuencia de reloj en función del consumo y la carga térmica, podría contribuir a esta variación: en cargas cortas o con menor número de hilos activos, algunos núcleos pueden alcanzar frecuencias más altas, reduciendo temporalmente el tiempo de ejecución.

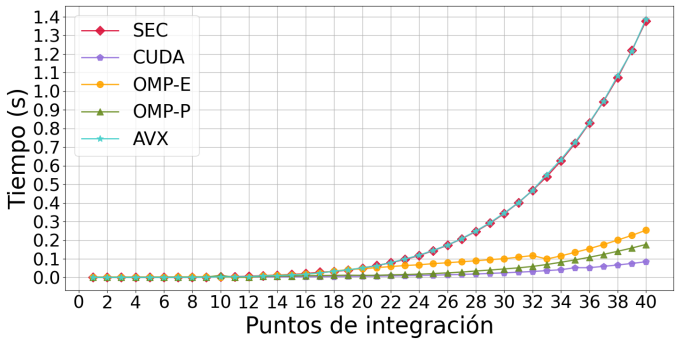
En resumen, estos factores explican por qué la caída de tiempo se presenta únicamente en configuraciones específicas y no en plataformas ARM como la Jetson TX2, donde la gestión de energía y la arquitectura de los núcleos es diferente.

5.2 Desempeño de los calendarizadores

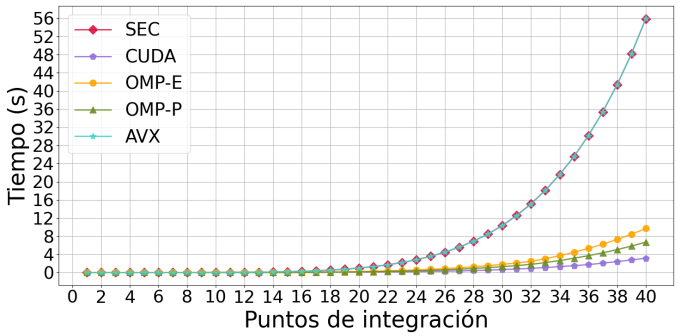
En esta sección se analizan los tiempos de ejecución obtenidos al aplicar las tres estrategias de calendarización propuestas: estático secuencial (SCHED SS), dinámico secuencial (SCHED DS) y estático concurrente (SCHED SC).



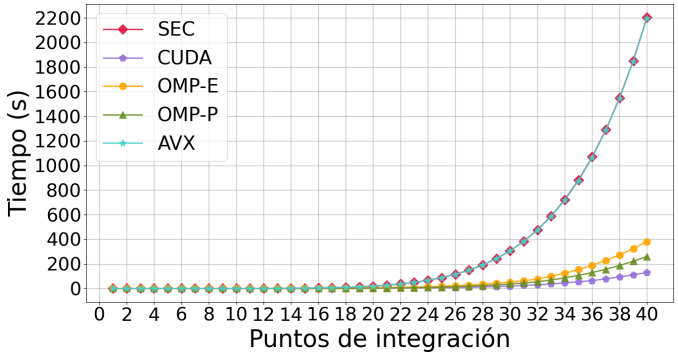
(a) 3D



(b) 4D

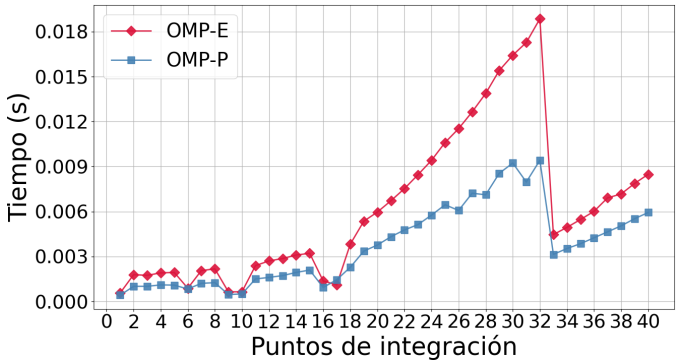


(c) 5D

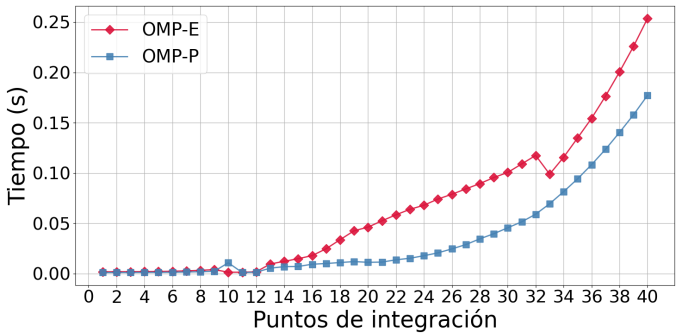


(d) 6D

Figura 5.3: Comparación de tiempos de ejecución (s) al evaluar la Función 1.3 usando la extrapolación de Romberg, en estación de trabajo CUDA.



(a) 3D



(b) 4D

Figura 5.4: Caída atípica en el tiempo de ejecución del módulo OMP alrededor del punto de integración 33 al evaluar la Función 1.3 en la estación de trabajo CUDA.

El objetivo principal de este análisis es comparar la eficiencia de cada estrategia en la asignación de módulos de ejecución, considerando su comportamiento tanto en la estación de trabajo CUDA como en la plataforma Jetson TX2.

5.2.1 Comparación general

La implementación de los tres calendarizadores presenta diferentes niveles de complejidad, tanto en su construcción como en su comportamiento computacional.

El calendarizador estático secuencial es el más sencillo de implementar, ya que sigue una estructura determinista donde las tareas se asignan de forma fija y lineal. Cada punto de integración se ejecuta de manera consecutiva sin depender de la sincronización entre entre hilos o procesos. La simplicidad del diseño permite minimizar el uso de estructuras auxiliares, empleando únicamente un arreglo para almacenar los identificadores de los módulos de ejecución. Esto resulta en una implementación más limpia y de bajo costo computacional.

La implementación del calendarizador dinámico secuencial presenta una complejidad ligeramente superior a la del calendarizador estático secuencial; sin embargo, sigue siendo manejable debido a que conserva una ejecución completamente secuencial. No obstante, requiere un mayor consumo de memoria, ya que necesita almacenar

estructuras auxiliares, como la tabla de tiempos de ejecución, para realizar comparaciones cada vez que se efectúa un cálculo.

El calendarizador estático concurrente es el más complejo de implementar debido a la inclusión de múltiples funciones auxiliares necesarias para manejar la ejecución paralela. Entre ellas destaca la búsqueda tabú, empleada para optimizar la asignación de tareas y evitar repeticiones en la distribución de subregiones. Además, es necesario incorporar un control de sincronización, candados y comunicación entre hilos concurrentes, lo cual incrementa notablemente la dificultad de programación y el consumo de recursos.

En términos de implementación, el calendarizador estático concurrente presenta la mayor complejidad debido a la cantidad de funciones de control y mecanismos de optimización requeridos. Por otro lado, los calendarizadores secuenciales resultan más simples, con un menor costo de mantenimiento.

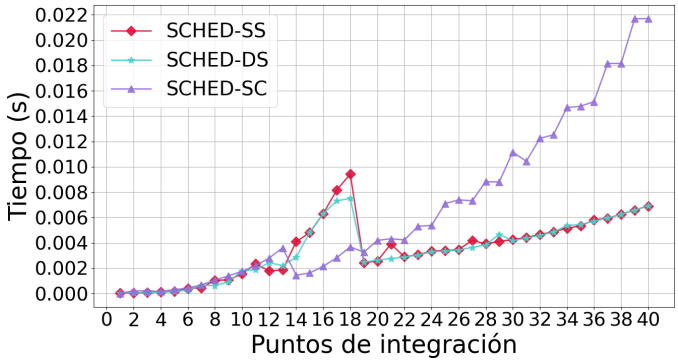
Análisis de tiempos de ejecución de los calendarizadores

En las Figuras 5.5 y 5.6 se presentan los tiempos de ejecución correspondientes a los calendarizadores estático secuencial (SCHED SS), dinámico secuencial (SCHED DS) y estático concurrente (SCHED SC). Estos resultados provienen de las pruebas descritas en el capítulo anterior (Secciones 4.3.2, 4.3.3 y 4.3.4).

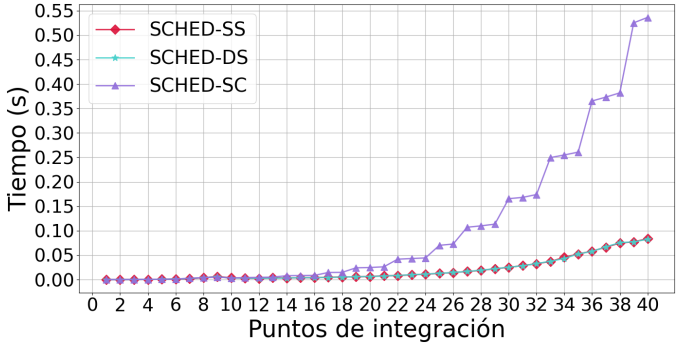
Las gráficas se encuentran organizadas por dimensión: 3D (Figuras 5.5a y 5.6a), 4D (Figuras 5.5b y 5.6b), 5D (Figuras 5.5c y 5.6c) y 6D (Figuras 5.5d y 5.6d). En general, los resultados indican que los calendarizadores secuenciales, tanto el estático como el dinámico, mantienen un desempeño estable y predecible, obteniendo tiempos de ejecución muy similares entre sí.

El calendarizador estático secuencial presenta la ventaja de una selección previa de módulos, basada en las tablas de tiempos, lo que minimiza la sobrecarga computacional en tiempo de ejecución. Aunque no emplea paralelismo, su desempeño en las pruebas resulta notablemente eficiente, registrando tiempos de ejecución inferiores a los del calendarizador estático concurrente. Esto se debe a que evita el costo asociado a la creación y sincronización de hilos, permitiendo una ejecución continua, ligera y estable.

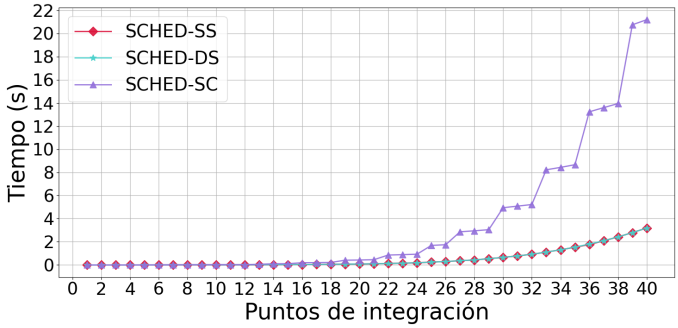
El calendarizador dinámico realiza la selección de módulos de manera directa durante la ejecución, lo que introduce una ligera sobrecarga computacional. Sin embargo, esta estrategia ofrece una mayor adaptabilidad ante posibles variaciones en el rendimiento de los módulos. A pesar del costo adicional, el calendarizador dinámico secuencial superó al calendarizador estático secuencial en las pruebas de 4D (Figuras 5.5b y 5.6b) y 5D (Figuras 5.5c y 5.6c) en ambas plataformas. Aunque las diferencias fueron de sólo una fracción de segundo, este comportamiento resultó inesperado. Una posible explicación es la naturaleza adaptable de la estrategia, que permite seleccionar en cada iteración el módulo más adecuado con base en los valores actualizados de la tabla de tiempos. Esto reduce los sesgos asociados a la asignación fija del método estático y puede favorecer ligeras mejoras en el tiempo total de ejecución.



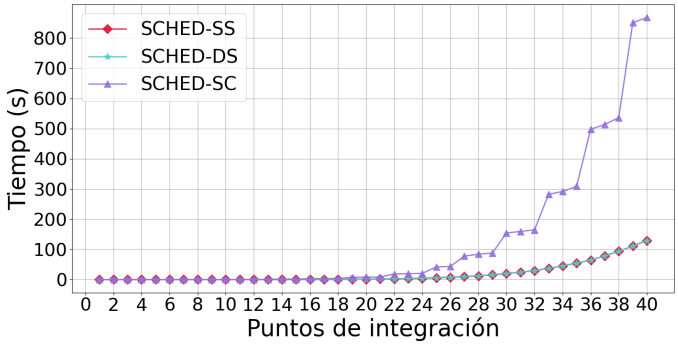
(a) 3D



(b) 4D

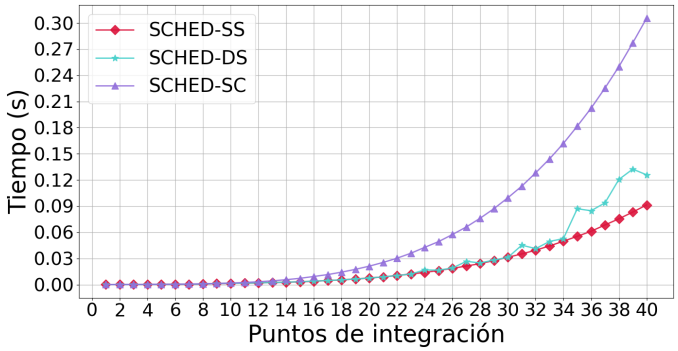


(c) 5D

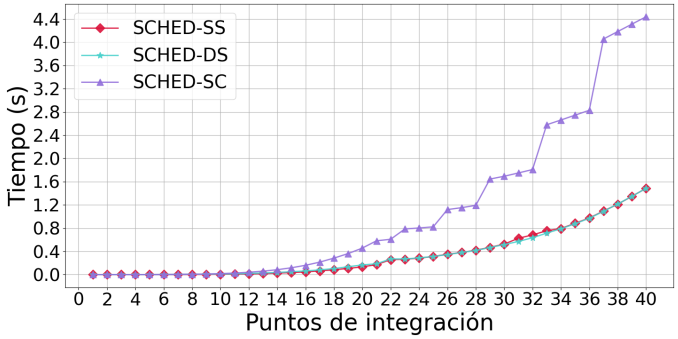


(d) 6D

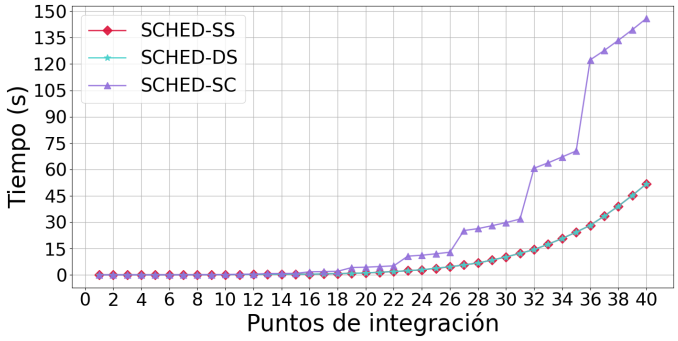
Figura 5.5: Comparación de tiempos de ejecución (s) de calendarizadores al evaluar la Función 1.3 en estación de trabajo CUDA.



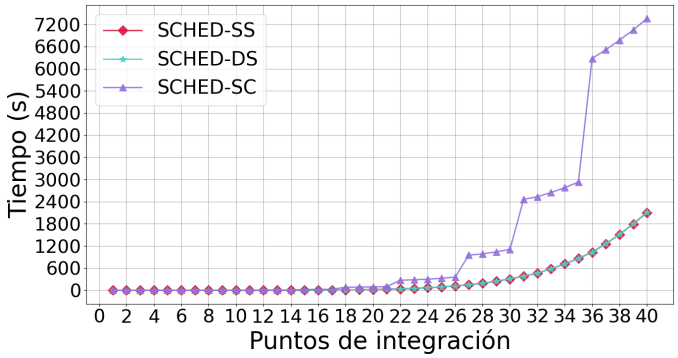
(a) 3D



(b) 4D



(c) 5D



(d) 6D

Figura 5.6: Comparación de tiempos de ejecución (s) de calendarizadores al evaluar la Función 1.3 en Jetson TX2.

En contraste, el calendarizador estático concurrente no logró superar el desempeño de las estrategias secuenciales. A pesar de estar diseñado para ejecutar dos módulos en paralelo mediante hilos independientes, la sobrecarga generada por la creación y sincronización de los hilos, así como la competencia de los hilos por el uso del procesador, redujo significativamente su eficiencia. En consecuencia, los tiempos de ejecución resultaron mayores que los obtenidos con los esquemas secuenciales, lo que sugiere que su implementación sólo sería ventajosa en sistemas con mayor cantidad de núcleos físicos y soporte de *hardware* para concurrencia intensiva.

En la estación de trabajo CUDA (Figuras 5.5), tanto el calendarizador estático secuencial como el dinámico secuencial mostraron un rendimiento muy similar, mientras que el calendarizador estático concurrente presenta una penalización significativa en el tiempo de ejecución a partir de los 20 puntos de integración, especialmente en 5D (Figura 5.5c) y 6D (Figura 5.5d).

En la Jetson TX2 (Figuras 5.6), la tendencia general es similar; sin embargo, la diferencia entre los calendarizadores secuenciales se vuelve más evidente en 3D (Figura 5.6a). En este caso, el calendarizador estático secuencial obtiene el menor tiempo de ejecución, seguido por el calendarizador dinámico secuencial y, finalmente, por el calendarizador estático concurrente. Por lo tanto, para un número reducido de dimensiones, el calendarizador estático secuencial aprovecha de manera más eficiente los recursos del sistema de calendarización en esta plataforma.

En resumen, el desempeño general de los calendarizadores demuestra que las estrategias secuenciales son más confiables y eficientes bajo las condiciones de prueba empleadas. La estrategia dinámica, aunque más compleja, demostró una ventaja concreta en escenarios de cuatro y cinco dimensiones, lo que sugiere que su aplicación resulta especialmente efectiva cuando las diferencias de rendimiento entre módulos varían con la dimensionalidad del problema. La estrategia concurrente, por otro lado, requiere una revisión más profunda o un entorno de *hardware* con mayor grado de paralelismo físico para alcanzar un beneficio real.

5.2.2 Impacto de la estrategia de calendarización

Para evaluar el beneficio real de las estrategias de calendarización, se comparó el desempeño de los calendarizadores: estático secuencial, dinámico secuencial y estático concurrente, frente al desempeño obtenido integrando 128 puntos sin extrapolación utilizando únicamente el módulo CUDA. Este último se tomó como referencia base, dado que mostró el mejor rendimiento entre todos los módulos individuales.

La Tabla 5.2 presenta las aceleraciones obtenidas para las dos plataformas utilizadas: estación de trabajo CUDA y Jetson TX2. En general, se observa que los calendarizadores estático secuencial y dinámico secuencial alcanzan las mayores aceleraciones en ambas plataformas, especialmente conforme aumenta la dimensionalidad del problema.

En la estación de trabajo CUDA, ambos calendarizadores presentan comportamientos casi idénticos, con aceleraciones que van desde aproximadamente 10× en 4D

Tabla 5.2: Aceleraciones de los calendarizadores usando la extrapolación de Romberg tomando como referencia el módulo de ejecución CUDA con 128 puntos.

Calendarizador	3D	4D	5D	6D
Estación de trabajo CUDA				
Estático secuencial	0.994	9.972	40.093	151.869
Dinámico secuencial	0.990	10.141	40.102	151.861
Estático concurrente	0.316	0.536	6.033	22.694
Jetson TX2				
Estático secuencial	1.749	8.823	42.213	172.797
Dinámico secuencial	1.270	8.833	42.217	172.080
Estático concurrente	0.523	2.953	14.962	49.412

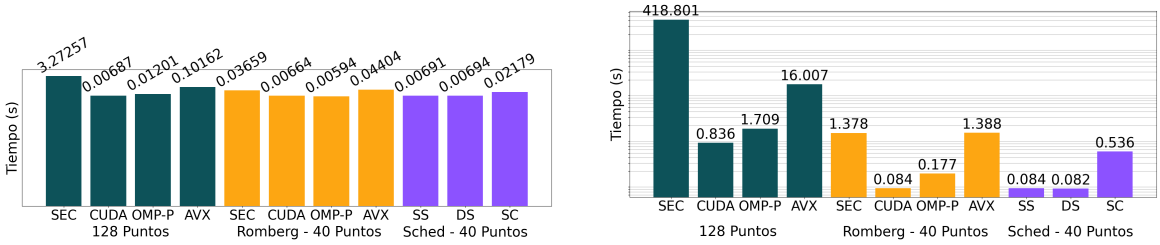
hasta más de 150× en 6D. Esto indica que la extrapolación de Romberg permitió reducir significativamente los tiempos de ejecución respecto al módulo CUDA con 128 puntos, manteniendo una eficiencia muy similar entre las variantes estática y dinámica. Por otro lado, el calendarizador estático concurrente muestra aceleraciones considerablemente menores. Apenas alcanza una aceleración de 6.03× en 5D y 22.69× en 6D, valores muy por debajo de los obtenidos por los calendarizadores secuenciales.

En la plataforma Jetson TX2, los resultados mantienen la misma tendencia: los calendarizadores secuenciales son los más eficientes, alcanzando aceleraciones de hasta 172× en 6D. El calendarizador dinámico secuencial tiene un desempeño muy similar al calendarizador estático secuencial, mostrando que el manejo dinámico de tareas no introduce penalizaciones relevantes en esta arquitectura. En contraste, el calendarizador estático concurrente vuelve a mostrar un desempeño inferior, con aceleraciones moderadas entre 0.52× en 3D y 49.4× en 6D, evidenciando que su complejidad no se ve compensada por una mejora en la velocidad.

Al analizar los tiempos de ejecución de los calendarizadores en comparación con los módulos individuales, evaluando un rango de 1 a 40 puntos con extrapolación de Romberg en la estación de trabajo CUDA, para las dimensiones 3D (Figura 5.7a), 4D (Figura 5.7b), 5D (Figura 5.7c) y 6D (Figura 5.7d), se observa que tanto el calendarizador estático secuencial como el dinámico secuencial presentan un desempeño muy cercano al del módulo individual CUDA con extrapolación de Romberg. Esto confirma que ambas estrategias administran de manera eficiente la asignación de módulos. Las diferencias de tiempo son mínimas, de apenas una fracción de segundo, lo que indica que el costo adicional asociado al calendarizador dinámico, derivado de realizar comparaciones en tiempo de ejecución, no afecta de manera significativa su rendimiento.

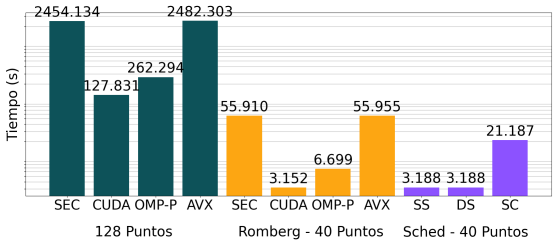
Por el contrario, el calendarizador estático concurrente muestra un aumento visible en los tiempos de ejecución, especialmente en las dimensiones más altas: 5D (Figura 5.7c) y 6D (Figura 5.7d), debido a que el costo asociado con la creación y sincronización de hilos, así como el uso compartido del procesador incrementan la complejidad sin aportar una ganancia de rendimiento.

En la Jetson TX2, para las dimensiones: 3D (Figura 5.8a), 4D (Figura 5.8b), 5D (Figura 5.8c) y 6D (Figura 5.8d), se repite la misma tendencia general. Sin embargo,

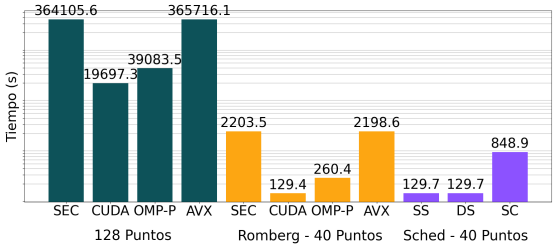


(a) 3D

(b) 4D

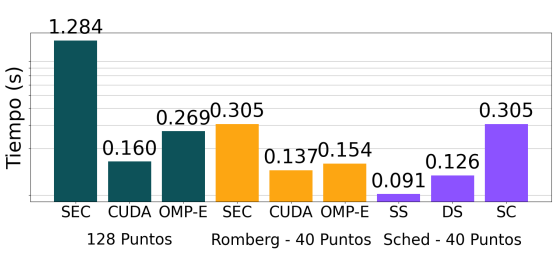


(c) 5D

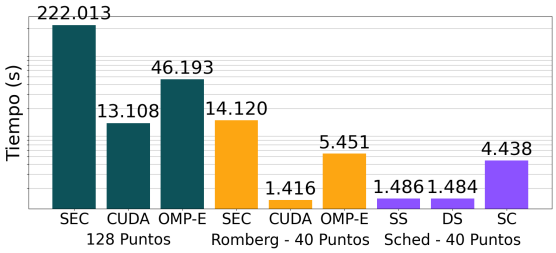


(d) 6D

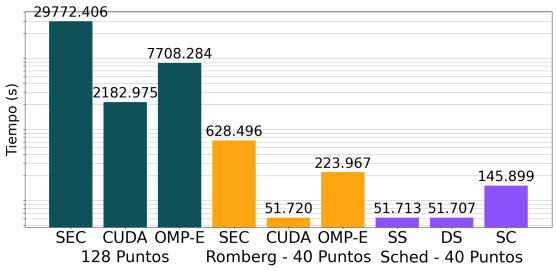
Figura 5.7: Comparación de tiempos de ejecución (s) al evaluar la Función 1.3 usando diferentes técnicas en estación de trabajo CUDA.



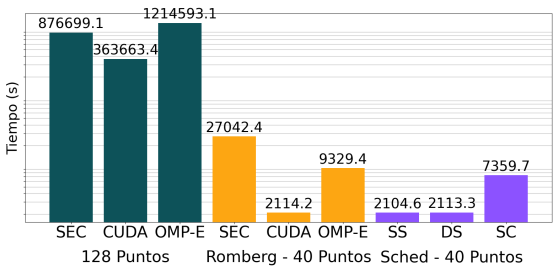
(a) 3D



(b) 4D



(c) 5D



(d) 6D

Figura 5.8: Comparación de tiempos de ejecución (s) al evaluar la Función 1.3 usando diferentes técnicas en Jetson TX2.

es importante destacar los resultados obtenidos en 3D (Figura 5.8a), donde los calendarizadores secuenciales alcanzan tiempos de ejecución inferiores a los del módulo CUDA con extrapolación de Romberg y logran aceleraciones de hasta 1.7× respecto al módulo CUDA con 128 puntos. Esto demuestra que la estrategia de calendarización

cumple con el objetivo de reducir el tiempo de ejecución.

Este comportamiento se debe principalmente a la heterogeneidad del *hardware*, que permite al calendarizador seleccionar los módulos más eficientes de acuerdo a las capacidades reales de la plataforma. Además, en esta dimensión todas las operaciones se ejecutan directamente en el CPU, dado que los tiempos de ejecución en el módulo CUDA son demasiado elevados (Figura 4.3a), lo que lo convierte en una opción no viable. Esto elimina por completo la transferencia de datos entre CPU y GPU, lo que contribuye a una reducción significativa en el tiempo de ejecución. A partir de cuatro dimensiones, aunque las aceleraciones siguen siendo elevadas, los resultados se asemejan a los obtenidos por los módulos individuales con extrapolación de Romberg, lo que indica que el beneficio principal de la calendarización radica en su capacidad de adaptación automática más que en una ganancia directa de velocidad.

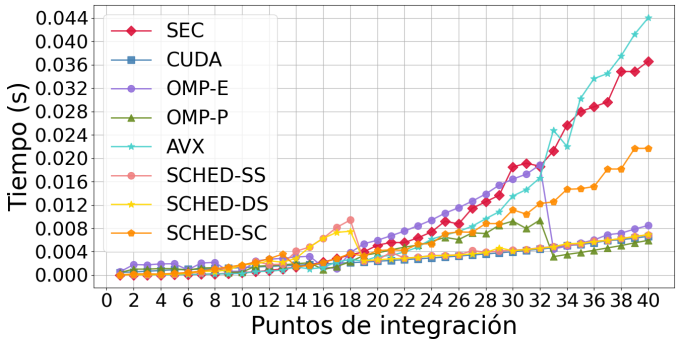
Las Figuras 5.9 y 5.10 complementan la comparación de las gráficas de barras al presentar los tiempos de ejecución de los calendarizadores comparados con los tiempos de los módulos individuales con la extrapolación de Romberg en el rango de 1 a 40 puntos de integración.

En la estación de trabajo CUDA (Figuras 5.9), se presentan las gráficas organizadas por dimensiones: 3D (Figura 5.9a), 4D (Figura 5.9b), 5D (Figura 5.9c) y 6D (Figura 5.9d). Los módulos con extrapolación de Romberg mantienen un crecimiento progresivo del tiempo de ejecución conforme aumenta el número de puntos de integración. Entre ellos, los módulos CUDA, OMP-E y OMP-P siguen destacando como los más eficientes, mientras que los módulos SEC y AVX presentan mayores tiempos.

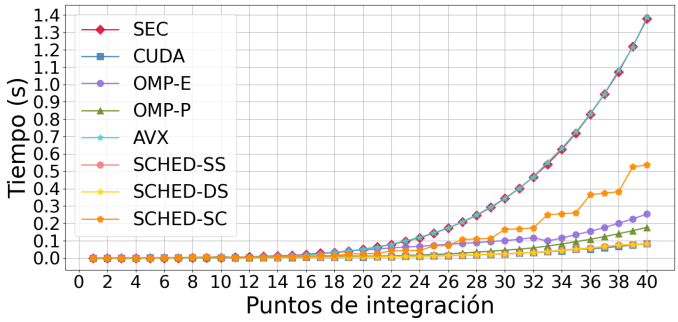
Al comparar los resultados de los calendarizadores, se aprecia que las estrategias secuenciales (SCHED-SS y SCHED-DS) logran tiempos de ejecución equivalentes al módulo individual CUDA. En cambio, el calendarizador estático concurrente (SCHED-SC) presenta un crecimiento más pronunciado, especialmente en las dimensiones 5D (Figura 5.9c) y 6D (Figura 5.9d), debido a los costos asociados a su diseño concurrente.

En la Jetson TX2 (Figuras 5.10), se presentan las gráficas organizadas por dimensiones: 3D (Figura 5.10a), 4D (Figura 5.10b), 5D (Figura 5.10c) y 6D (Figura 5.10d). Los módulos individuales CUDA y OMP-E conservan el mejor desempeño, mientras que los calendarizadores secuenciales (SCHED-SS y SCHED-DS) continúan mostrando un comportamiento estable y eficiente. Nuevamente, el calendarizador concurrente (SCHED-SC) exhibe una degradación notable en las dimensiones superiores, reflejando que su estructura paralela no se adapta eficientemente a dispositivos con menor capacidad de procesamiento paralelo y memoria compartida limitada.

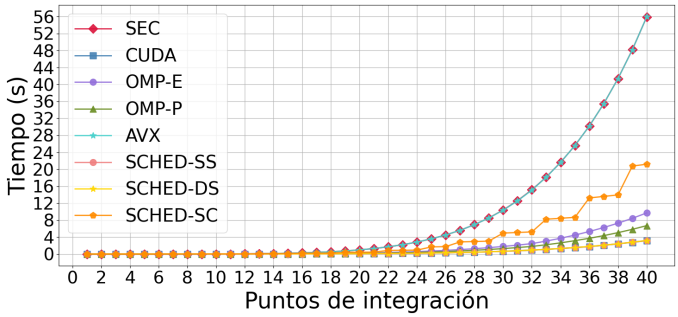
En particular, se esperaba que el calendarizador estático concurrente mostrara un rendimiento competitivo, especialmente por el manejo de concurrencia y la incorporación de la búsqueda tabú como estrategia de optimización. En (Morales y Puga, 2022), se propuso un calendarizador para este problema, basado en la búsqueda tabú. Dicho enfoque fue evaluado en un entorno de simulación, donde los resultados mostraron una mejora significativa en la asignación de módulos y en el equilibrio de carga entre los módulos disponibles.



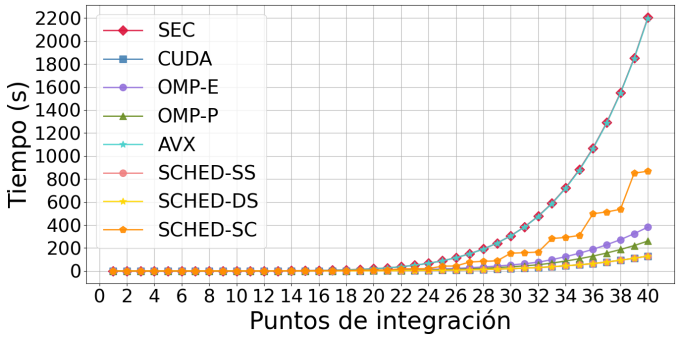
(a) 3D



(b) 4D

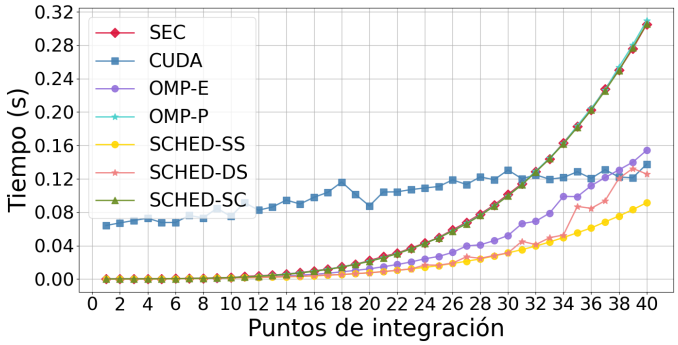


(c) 5D

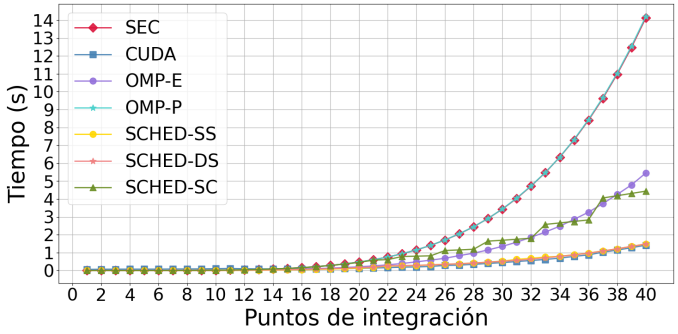


(d) 6D

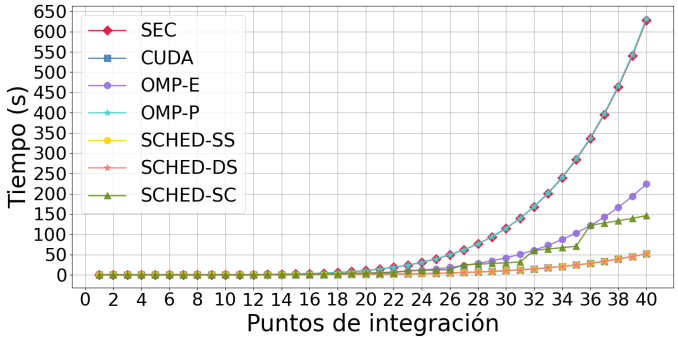
Figura 5.9: Comparación de tiempos de ejecución (s) al evaluar la Función 1.3 usando la extrapolación de Romberg: módulos VS calendarizadores, en estación de trabajo CUDA.



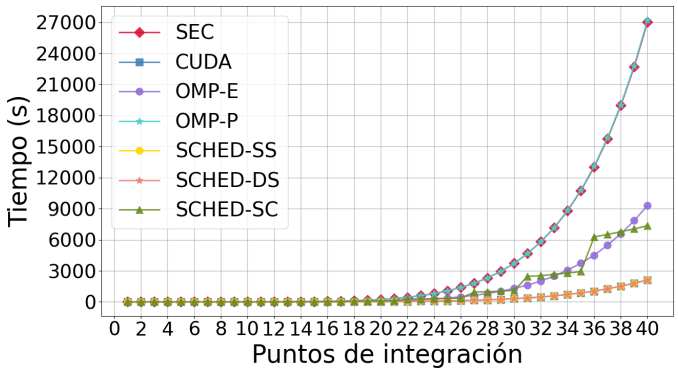
(a) 3D



(b) 4D



(c) 5D



(d) 6D

Figura 5.10: Comparación de tiempos de ejecución (s) al evaluar la Función 1.3 usando la extrapolación de Romberg: módulos VS calendarizadores, en Jetson TX2.

El algoritmo de búsqueda tabú demostró, en ese contexto, una capacidad eficiente para evitar soluciones subóptimas y reducir el tiempo total de ejecución simulado. Sin embargo, al trasladar este enfoque a entornos reales en el presente trabajo, los resultados difirieron considerablemente.

El calendarizador estático concurrente, que integra la búsqueda tabú para decidir las asignaciones, presentó un desempeño inferior al esperado. Esto se debe a que, en las simulaciones del trabajo previo, no se consideraron factores inherentes al *hardware* real, como la competencia entre procesos por el tiempo de CPU, la sobrecarga asociada a la creación y sincronización de hilos y la latencia por acceso concurrente a memoria compartida. En consecuencia, el comportamiento observado confirma que, aunque la búsqueda tabú es prometedora en entornos teóricos o simulados, su implementación directa en sistemas reales puede generar una sobrecarga que anula sus beneficios de optimización.

En general, los calendarizadores secuenciales alcanzan niveles de desempeño a la par de los módulos más rápidos ejecutados individualmente, lo que demuestra que la estrategia de selección automática no introduce penalizaciones significativas y mantiene un rendimiento competitivo. El calendarizador estático concurrente, a pesar de su diseño más complejo, no logran superar el rendimiento de las variantes secuenciales ni de los módulos optimizados en GPU.

Conclusiones

El trabajo desarrollado en esta tesis abordó el problema del cálculo eficiente de integrales multidimensionales, una tarea de gran relevancia científica y técnica debido a su presencia en aplicaciones de simulación física, análisis estadístico y modelado computacional. Sin embargo, este tipo de integrales presenta una alta complejidad conforme aumentan las dimensiones, lo que exige estrategias de optimización tanto en el nivel algorítmico como en el uso eficiente de los recursos de *hardware* disponibles.

En este trabajo se implementó una estrategia de calendarización para aprovechar de mejor manera dichos recursos. Primero, se realizó una revisión exhaustiva de los métodos de calendarización utilizados en entornos heterogéneos, identificando que la mayoría de los trabajos existentes se centran en la distribución de tareas generales, sin considerar las tareas de propósito especifico, como las integrales multidimensionales, ni las limitaciones físicas del *hardware*. El estudio permitió seleccionar tres estrategias diferentes: calendarizador estático secuencial, calendarizador dinámico secuencial y calendarizador estático concurrente, que ofrecen distintos grados de adaptabilidad y complejidad. Asimismo, se analizó la aplicación de heurísticas como la búsqueda tabú y los árboles de decisiones.

Posteriormente, se amplió la biblioteca de integrales multidimensionales incorporando tres componentes: (1) un módulo basado en instrucciones vectoriales AVX, (2) un *codelet* encargado de la gestión de los módulos y (3) las estrategias de calendarización diseñadas para optimizar la distribución de carga entre los diferentes módulos de ejecución.

El módulo de ejecución AVX se implementó con el propósito de complementar los módulos existentes (secuencial, CUDA y OMP), lo que permitió explotar el paralelismo a nivel de datos. Su incorporación aumentó la flexibilidad de la biblioteca y la volvió más adecuada para arquitecturas modernas. El *codelet* actuó como intermediario entre los calendarizadores y la biblioteca de integrales multidimensionales, además se encargó de abstraer y referenciar la ejecución de los distintos módulos. Esta estrategia permitió una administración más limpia y eficiente del código, al facilitar la selección automática del método de ejecución según las necesidades de cada punto de integración.

Como tercer componente y punto central, las tres estrategias de calendarización implementadas para mejorar el rendimiento de los módulos fueron:

- Estático secuencial: asigna previamente el mejor módulo a cada punto con base en una tabla de tiempos.

- Dinámico secuencial: permite una asignación, en tiempo de ejecución, aprovechando los recursos disponibles.
- Estático concurrente: realiza ejecuciones en pares, optimizadas por la búsqueda tabú, con control de dispositivos mediante candados, permitiendo el paralelismo sin conflictos de acceso.

A través de las pruebas preliminares, se comprobó la estabilidad numérica de los módulos de ejecución, se determinó el número mínimo de puntos necesario para alcanzar un error aceptable (igual o menor a 10^{-4}) y se verificó la efectividad de la extrapolación de Romberg para reducir la cantidad de puntos sin comprometer la precisión del resultado.

Después, se realizaron las pruebas correspondientes a las distintas estrategias de calendarización, utilizando como base las tablas de tiempo generadas para ambas plataformas de prueba: la estación de trabajo CUDA y la Jetson TX2. Estas pruebas permitieron verificar la correcta interacción entre los calendarizadores y el *codelet*, así como la capacidad del sistema de calendarización para adaptarse a entornos con diferentes características de *hardware*.

El análisis de resultados permitió evaluar de forma integral el comportamiento de los distintos módulos de ejecución y los calendarizadores propuestos. El desempeño del módulo AVX resultó limitado para configuraciones con pocos puntos, debido a su naturaleza parcialmente paralelizable y a la sobrecarga en la administración de registros vectoriales, lo que impidió que superara al módulo secuencial en la mayoría de los casos. Por otra parte, en la Jetson TX2 se observó un comportamiento interesante: el módulo OMP-E, que emplea únicamente núcleos de eficiencia, superó al módulo OMP-P, que usa núcleos de rendimiento, este comportamiento se atribuye al menor consumo energético, menor frecuencia térmica y a la mejor coherencia de caché de los núcleos de eficiencia bajo cargas de trabajo prolongadas.

Respecto a los calendarizadores, los resultados demostraron que las estrategias secuenciales, tanto estática como dinámica, ofrecen un desempeño similar y estable, con ligeras ventajas del enfoque dinámico en cuatro y cinco dimensiones. En contraste, el calendarizador estático concurrente, que integraba un esquema de búsqueda tabú para la asignación de módulos, presentó un rendimiento inferior al esperado. Debido a que la creación y sincronización de hilos, así como la competencia por los recursos del procesador que introduce retardos significativos que anulan los beneficios teóricos del algoritmo.

Finalmente, se identificaron particularidades específicas del *hardware* en el comportamiento de los módulos, como la caída puntual de tiempo observada en los módulos OMP en la estación de trabajo Intel, atribuida a la interacción entre el *Turbo Boost*, la asignación híbrida de núcleos y la vectorización automática.

En conjunto, los resultados obtenidos permiten concluir que el sistema de calendarización propuesto es funcional y adaptable a distintas plataformas, integrando de manera eficaz las técnicas de extrapolación y calendarización. Sin embargo, aunque logró reducir el tiempo de ejecución en comparación con el enfoque de 128 puntos, su desempeño se mantuvo comparable al del módulo individual de CUDA utilizando la

extrapolación de Romberg con 40 puntos, sin alcanzar la mejora esperada en los tiempos de ejecución. No obstante, las estrategias secuenciales implementadas constituyen una base sólida sobre la cual pueden desarrollarse optimizaciones futuras más especializadas, tales como la incorporación de técnicas de aprendizaje automático para la selección dinámica de módulos, el balanceo predictivo de carga entre CPU y GPU, y mecanismos de autoajuste que adapten la ejecución a las características particulares de cada plataforma y problema numérico.

Trabajo a futuro

Como trabajo a futuro, se propone la optimización del calendarizador estático concurrente. Aunque este enfoque permitió aprovechar múltiples módulos para la ejecución paralela, su desempeño aún puede mejorarse mediante estrategias más refinadas de asignación y balanceo de carga. Se buscará reducir la sobrecarga asociada a la coordinación entre hilos y dispositivos, así como explorar técnicas heurísticas más eficientes que puedan reemplazar o complementar mecanismos como la búsqueda tabú. El objetivo es obtener un comportamiento más estable y competitivo, especialmente en integrales de mayor complejidad dimensional.

No obstante, el eje central del trabajo a futuro consistirá en el diseño e implementación de un calendarizador dinámico concurrente. Esta estrategia de calendarización combinará la asignación dinámica de tareas con la ejecución paralela, permitiendo ajustar la calendarización en tiempo de ejecución conforme cambien las características del cálculo o del entorno de la plataforma. La idea es desarrollar un sistema capaz de redistribuir grupos de puntos de la integral entre los distintos módulos disponibles, reaccionando a variaciones en el rendimiento, latencia o carga del sistema. Esto ofrecería una mayor adaptabilidad frente a escenarios heterogéneos, donde las diferencias entre componentes de cómputo (como la CPU o la GPU) requieren decisiones más flexibles y eficientes que las que puede ofrecer un enfoque estático.

Finalmente, se propone ampliar el trabajo de tesis hacia nuevas plataformas de ejecución y entornos distribuidos. Probar los calendarizadores desarrollados en diferentes arquitecturas de `hardware` permitirá analizar su robustez, portabilidad y escalabilidad. Asimismo, la adaptación del sistema a un ambiente distribuido abriría la posibilidad de coordinar múltiples nodos de cómputo, lo cual resultaría especialmente relevante para problemas de integración multidimensional de gran escala.

En conjunto, se busca consolidar un sistema de calendarización más flexible, eficiente y adaptable, capaz de responder a las demandas crecientes de aplicaciones científicas que requieren cálculos de alto rendimiento en entornos cada vez más complejos y diversificados.

Glosario

Codelet

Unidad encargada de referenciar múltiples funciones codificadas, permitiendo su gestión y ejecución de manera independiente.

Componentes de ejecución

Elementos encargados de realizar las operaciones de procesamiento, específicamente la CPU y GPU.

Dimensión

Variable independiente dentro del dominio de integración. Cada dimensión añade un eje adicional al espacio sobre el cual se calcula la integral, de modo que una integral de n dimensiones corresponde al cálculo del volumen hiperdimensional bajo una función $f(x_1, x_2, ..., x_n)$ dentro de un dominio definido.

Integrando

Función matemática que a integrar. En el contexto de una integral multidimensional, es una función de varias variables: $\int f(x_1, x_2, \ldots, x_D)\, dx_1\, dx_2 \cdots dx_D$

Módulos de ejecución

Funciones responsables de ejecutar la extrapolación de Romberg, implementadas de manera secuencial y paralela (CUDA, OpenMP y AVX).

Plataformas

Equipos de cómputo y tarjetas programables utilizadas para la ejecución de los procesos.

Puntos de integración

Cantidad de evaluaciones que realiza el método de integración en cada dimensión para aproximar el valor de la integral. Un mayor número de puntos generalmente mejora la precisión, pero incrementa el tiempo de ejecución.

Acrónimos

AVX	Advanced Vector Extensions
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
DAG	Directed Acyclic Graph
E-Cores	Efficient Cores
FPGA	Field-Programmable Gate Arrays
GPU	Graphics Processing Unit
HPC	High-Performance Computing
IA	Inteligencia Artificial
OMP	Open Multi-Processing
P-Cores	Performance Cores
SCHED-DS	Schedule Dynamic Sequential
SCHED-SC	Schedule Static Concurrent
SCHED-SS	Schedule Static Sequential
SIMD	Single Instruction, Multiple Data

Referencias

Abramowitz, M. (1974). *Handbook of mathematical functions, with formulas, graphs, and mathematical tables* (1.ª ed.). New York: Dover Publications Inc.

AlEbrahim, S., y Ahmad, I. (2017). Task scheduling for heterogeneous computing systems. *Supercomput*, *73*, 2313–2338. doi: 10.1007/s11227-016-1917-2

Al-Khateeb, H., Benkhlifa, E., y Bounceur, A. (2018). An overview of task scheduling in cloud computing: Concepts and challenges. *Network and Computer Applications*, *113*, 1–18. doi: 10.1016/j.jnca.2018.04.023

Arabnejad, H., y Barbosa, J. G. (2014). List scheduling algorithm for heterogeneous systems by an optimistic cost table. *IEEE Transactions on Parallel and Distributed Systems*, *25*(3), 682–694. doi: 10.1109/TPDS.2013.57

Arfken, G. (1985). *Mathematical methods for physicists* (3.ª ed.). Oxford, Ohio: Academic Press Inc.

Arumugam, K., Godunov, A., Ranjan, D., Terzic, B., y Zubair, M. (2013). An efficient deterministic parallel algorithm for adaptive multidimensional numerical integration on gpus. En *2013 42nd international conference on parallel processing* (p. 486-491).

Arzi, Y., y Iaroslavitz, L. (2000). Operating an fmc by a decision-tree-based adaptive production control system. *International Journal of Production Research*, *38*(3), 675–697. doi: 10.1080/002075400189365

Asanović, K., Bodik, R., Catanzaro, B. C., Gebis, J. J., Husbands, P., Keutzer, K., ... Yelick, K. A. (2006). *The landscape of parallel computing research: A view from berkeley* (Inf. Téc. n.º UCB/EECS-2006-183). Berkeley.

Augonnet, C., Thibault, S., Namyst, R., y Wacrenier, P. A. (2009). Starpu: A unified platform for task scheduling on heterogeneous multicore architectures. En H. Sips, D. Epema, y H. X. Lin (Eds.), *Euro-par 2009 parallel processing* (pp. 863–874). Berlin, Heidelberg: Springer Berlin Heidelberg.

Berntsen, J., Espelid, T. O., y Genz, A. (1991a). An adaptive algorithm for the approximate calculation of multiple integrals. *ACM Trans. Math. Softw.*, *17*(4), 437-–451. doi: https://doi.org/10.1145/210232.210233

Berntsen, J., Espelid, T. O., y Genz, A. (1991b). Algorithm 698: Dcuhre: an adaptive multidemensional integration routine for a vector of integrals. *ACM Trans. Math. Softw.*, *17*(4), 452–456. doi: https://doi.org/10.1145/210232.210234

Brodtkorb, A., Dyken, C., Hagen, T., Hjelmervik, J., y Storaasli, O. (2010). State-of-the-art in heterogeneous computing. *Scientific Programming*, *18*, 1-5. doi: 10.1155/2010/540159

Buyya, R., Vecchiola, C., y Selvi, S. (2013). *Mastering cloud computing: Foundations and applications programming.* Morgan Kaufmann Publishers.

Coulouris, G., Dollimore, J., Kindberg, T., y Blair, G. (2005). *Distributed systems: Concepts and design* (5.ª ed.). Amsterdam: Addison-Wesley Longman.

Dahlquist, G., y Björck, A. (2008). *Numerical methods in scientific computing* (1.ª ed., Vol. 1). Philadelphia: Society for Industrial and Applied Mathematics.

Dastjerdi, A., Gupta, H., Calheiros, R., Ghosh, S., y Buyya, R. (2016). Chapter 4 - fog computing: principles, architectures, and applications. En *Internet of things* (p. 61-75). Morgan Kaufmann. doi: 10.1016/B978-0-12-805395-9.00004-6

Ding, S., Wu, J., Xie, G., y Zeng, G. (2017). A hybrid heuristic-genetic algorithm with adaptive parameters for static task scheduling in heterogeneous computing system. En *2017 ieee trustcom/bigdatase /icess* (pp. 761–766). doi: 10.1109/Trustcom/BigDataSE/ICESS.2017.310

Garey, M. R., y Johnson, D. S. (1976, julio). Scheduling tasks with nonuniform deadlines on two processors. *Association for Computing Machinery*, *23*(3), 461–467. doi: 10.1145/321958.321967

Gendreau, M., y Potvin, J. Y. (2005). Parallel tabu search. En E. K. Burke y G. Kendall (Eds.), *Search methodologies: Introductory tutorials in optimization and decision support techniques* (pp. 165–186). Boston, MA: Springer US. doi: 10.1007/0-387-28356-0_6

Genz, A. (1972). An adaptive multidimensional quadrature procedure. *Computer Physics Communications*, *4*(1), 11-15. doi: https://doi.org/10.1016/0010-4655(72)90024-0

Gibbs, D. (1915). *A course in interpolation and numerical integration for the mathematical laboratory* (1st ed.). Cornell University Library.

Hahn, T. (2005). Cuba—a library for multidimensional numerical integration. *Computer Physics Communications*, *168*(2), 78-95. doi: https://doi.org/10.1016/j.cpc.2005.01.010

Hansen, P. (1986). The steepest ascent mildest descent heuristic for combinatorial programming. En *Congress on numerical methods in combinatorial optimization* (pp. 70–145). Capri, Italy.

Hill, M. D., y Marty, M. R. (2008). Amdahl's law in the multicore era. *Computer*, *41*(7), 33-38. doi: 10.1109/MC.2008.209

Jeannot, E., y Zilinskas, J. (2014). *High-performance computing on complex environments*. Wiley.

Kebaier, A. (2005). Statistical romberg extrapolation: A new variance reduction method and applications to option pricing. *The Annals of Applied Probability*, *15*(4), 2681 – 2705. doi: https://doi.org/10.1214/105051605000000511

Keister, B. D. (1996). Multidimensional quadrature algorithms. *Computers in Physics*, *10*(2), 119–128.

Kim, C. O., Min, H. S., y Yih, Y. (2010). Integration of inductive learning and neural networks for multi-objective fms scheduling. *International Journal of Production Research*, *36*(9), 2497–2509. doi: 10.1080/002075498192652

Kim, S. I., y Kim, J. K. (2019). A method to construct task scheduling algorithms for heterogeneous multi-core systems. *IEEE Access*, *7*, 142640–142651. doi: 10.1109/ACCESS.2019.2944238

Kotsiantis, S. B. (2013). Decision trees: A recent overview. *Artificial Intelligence Review*, *39*(4), 261—283. doi: 10.1007/s10462-011-9272-4

Kronrod, A. S. (1964). *Nodes and weights of quadrature formulas* (1.ª ed.). New

York: Consultants Bureau.

Lavaei, J., Noghabi, B., Chen, Q., y Xue, G. (2018). Online optimization of heterogeneous datacenters with resource-efficient workloads. *IEEE Transactions on Cloud Computing*, *8*(4), 1346–1359. doi: 10.1109/TCC.2018.2816479

Liu, E. H. L. (2006). Fundamental methods of numerical extrapolation with applications. *Mitopencourseware, Massachusetts Institute Of Technology*, *209*.

Liu, J., Li, J., Li, D., Qian, D., y 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*, 101–112. doi: 10.1016/j.jpdc.2018.10.013

Mollick, E. (2006). Establishing moore's law. *Annals of the History of Computing, IEEE*, *28*(3), 62–75. doi: 10.1109/MAHC.2006.45

Moore, G. E. (1965). Cramming more components onto integrated circuits. *Electronics*, *38*(8), 114–117. doi: 10.1109/N-SSC.2006.4785860

Morales, A. I., y Puga, J. J. (2022). *Planificador inteligente para integración numérica multidimensional en ambientes heterogéneos* (Tesis de Master no publicada). Instituto Politécnico Nacional, Ciudad de México, México.

Nilsson, N. J. (1933). *Principles of artificial intelligence*. Los Altos, CA.: Morgan Kaufmann.

Notaris, S. E. (2016). Gauss-kronrod quadrature formulae-a survey of fifty years of research. *Electronic Transactions on Numerical Analysis*, *45*(1), 371–404.

Park, S. C., Raman, N., y Shaw, M. J. (1997). Adaptive scheduling in dynamic flexible manufacturing systems: a dynamic rule selection approach. *IEEE Transactions on Robotics and Automation*, *13*(4), 486–502. doi: 10.1109/70.611301

Patterson, T. (1968). The optimum addition of points to quadrature formulae. *Mathematics of Computation*, *22*, 847–856.

Piessens, R., y Branders, M. (1974). A note on the optimal addition of abscissas to quadrature formulas of gauss and lobatto type. *Mathematics of Computation*, *28*(125), 135-139.

Quintero-Monsebaiz, R., Meneses-Viveros, A., Carranza, F., Cortés, C. G., González-Zamudio, A., y Vela, A. (2021). Multidimensional adaptative and deterministic integration in cuda and openmp. *Supercomputing*, *77*, 12075–12097.

Salas-González, R. (2023). *Caracterización de tareas y recursos para la simulación de un calendarizador en un ambiente heterogéneo* (Tesis de Master no publicada). Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional.

Shaw, M. J., Park, S., y Raman, N. (1992). Intelligent scheduling with machine learning capabilities: The induction of scheduling knowledge. *IIE Transactions*, *24*(2), 156–168. doi: 10.1080/07408179208964213

Shiue, Y.-R., y Su, C.-T. (2003). An enhanced knowledge representation for decision-tree based learning adaptive scheduling. *International Journal of Computer Integrated Manufacturing*, *16*(1), 48–60. doi: 10.1080/713804978

Silberschatz, A., Galvin, P. B., y Gagne, G. (2012). *Operating system concepts* (9th

ed.). Wiley Publishing.

Song, Y., Li, C., Tian, L., y Song, H. (2023). A reinforcement learning based job scheduling algorithm for heterogeneous computing environment. *Computers and Electrical Engineering*, *107*. doi: https://www.sciencedirect.com/science/article/pii/S0045790623000782

Sotiriades, E., Petraki, E., Kartsakli, E., Souravlias, D., y Bouganis, C.-S. (2015). A survey of task scheduling in multicore and accelerator-based systems. *ACM Computing Surveys*, *48*(1).

Sterling, T., Anderson, M., y Brodowicz, M. (2017). *High performance computing: Modern systems and practices*. Elsevier. doi: 10.1016/C2013-0-09704-6

Szegö, G. (1935). Über gewisse orthogonale polynome, die zu einer oszillierenden belegungsfunktion gehören. *Mathematische Annalen*, *110*(1), 501–513.

Szegő, G. (1975). *Orthogonal polynomials* (4.ª ed., Vol. 23). Rhode Island: American Mathematical Society.

Topcuoglu, H., Hariri, S., y W., M.-Y. (2002). Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, *13*(3), 260–274. doi: 10.1109/71.993206

Zheng, S., Liang, Y., Wang, S., Chen, R., y Sheng, K. (2020). Flextensor: An automatic schedule exploration and optimization framework for tensor computation on heterogeneous system. En *Proceedings of the twenty-fifth international conference on architectural support for programming languages and operating systems* (p. 859–873). New York, NY, USA: Association for Computing Machinery. doi: https://doi.org/10.1145/3373376.3378508