ESPE

UNIVERSIDAD DE LAS FUERZAS ARMADAS

INNOVACIÓN PARA LA EXCELENCIA

Nombres de los integrantes del grupo:	**- Blacio Julio - Lainez Ricardo - Manangón Zaith - Tufiño Erick**	**NOTA**
Carrera:	**Ingeniería en software**	
NRC:	**23254**	
Nombre del docente:	**Ing. Carlos Pillajo**	

PRIMER PARCIAL - ACTIVIDAD GRUPAL 1

Unidad 1

Laboratorio de Computación Paralela

Tema:

Explorando SIMD con operaciones vectoriales

Objetivos:

- Implementar y analizar el rendimiento de operaciones vectoriales con y sin SIMD.

Resultados de Aprendizaje:

- Análisis de ingeniería. La capacidad de identificar, formular y resolver problemas de ingeniería en su especialidad; elegir y aplicar de forma adecuada métodos analíticos, de cálculo y experimentales ya establecidos; reconocer la importancia de las restricciones sociales, de salud y de seguridad, ambientales, económicas e industriales.

- Proyectos de ingeniería. Capacidad para proyectar, diseñar y desarrollar productos complejos (piezas, componentes, productos acabados, etc.) procesos y sistemas de su especialidad, que cumplen con los requisitos establecidos, incluyendo tener

conciencia de los aspectos sociales, de salud y seguridad, ambientales, económicos e industriales; así como seleccionar y aplicar métodos de proyectos apropiados.

- Aplicación práctica de la ingeniería. Comprensión de las técnicas aplicables y métodos de análisis, proyecto e investigación y sus lineamientos en el ámbito de su especialidad.

- Comunicación y trabajo en equipo. Capacidad para comunicar eficazmente información, ideas, problemas y soluciones en el ámbito de ingeniería y con la sociedad en general.

Instrucciones:

- Implemente una suma de dos vectores de tamaño 1 millón de elementos:
    1. Usando un ciclo for tradicional.
    2. Usando instrucciones SIMD (como SSE).

- Medir el tiempo de ejecución con clock() y comparar. Por lo menos capturar 50 mediciones.

- Graficar los tiempos obtenidos con Python (matplotlib) o Tableau.

- Analice en un informe:
    1. ¿Cuánto se reduce el tiempo?
    2. ¿En qué casos SIMD no sería útil?
    3. ¿Qué ocurriría con un vector que no sea múltiplo de 4?

Objetivo General

- Analizar y comparar el rendimiento de una operación vectorial (suma) implementada tanto de forma secuencial como utilizando paralelismo a nivel de datos mediante instrucciones SIMD (SSE), evaluando su eficiencia en distintos entornos de hardware.

Objetivo Específico

- Implementar dos métodos de suma de vectores: una secuencial mediante un ciclo for tradicional y otro utilizando instrucciones SIMD (SSE), sobre vectores de un millón de elementos.
- Medir el tiempo de ejecución de cada método utilizando clock() en C, realizando al menos 50 ejecuciones por sistema para obtener resultados estadísticamente representativos.
- Visualizar y comparar los resultados mediante gráficos generados con Python (matplotlib), analizando la diferencia de rendimiento entre ambos enfoques.
- Identificar casos en los que SIMD no proporciona mejoras significativas y analizar el comportamiento cuando los vectores no son múltiplos del tamaño del registro SIMD.
- Proponer recomendaciones prácticas para la aplicación de SIMD en contextos reales, considerando la naturaleza de los datos, el tipo de operación y las capacidades del hardware disponible.

Herramientas Utilizadas

Las herramientas que se utilizaron para el desarrollo de la actividad fueron:

- Python
- C
- Codeblocks 25.03
- Librerías para python: pandas, matplotlib.
- Librerías para C: xmmintrin.h, time.h, stdlib.h,stdio.h

Desarrollo (explicación del código)

1.- El programa comienza con la inclusión de las librerías necesarias:

- stdio.h - stdlib.h: Para funciones básicas de entrada/salida y manejo de memoria.
- time.h: Para medir el tiempo de ejecución de los algoritmos usando la función clock().

- xmmintrin.h: Que permite el uso de instrucciones SIMD (SSE) para el procesamiento paralelo de datos en registros de 128 bits.

2.- A continuación, se define una constante N = 1000000 que representa el tamaño de los vectores a operar. Luego se declaran tres arreglos globales de tipo float:

- A y B, que serán los vectores de entrada,
- C, que almacenará el resultado de la suma elemento a elemento.

3.- La función inicializar_vectores() recorre los vectores A y B con un bucle for, asignando valores de prueba (A[i] = i y B[i] = 2*i). Esto permite verificar fácilmente que la suma se realiza correctamente.

4.- La función suma_secuencial() realiza la suma de vectores mediante un bucle tradicional for. Cada elemento i de los vectores se suma y el resultado se almacena en C[i].

5.- La función suma_simd_sse() utiliza instrucciones SSE (Streaming SIMD Extensions) para operar sobre 4 elementos float a la vez:

- __m128 va = _mm_loadu_ps(&A[i]); // carga 4 floats desde A[i]
- __m128 vb = _mm_loadu_ps(&B[i]); // carga 4 floats desde B[i]
- __m128 vc = _mm_add_ps(va, vb); // suma paralela
- _mm_storeu_ps(&C[i], vc); // guarda el resultado en C[i]

Este enfoque permite procesar datos en paralelo utilizando registros de 128 bits, lo que reduce el número de iteraciones y mejora considerablemente el rendimiento para vectores grandes.

6.- En la función main() se ejecuta lo siguiente:

- Inicialización de los vectores.
- Medición del tiempo de la suma secuencial con clock(), y almacenamiento del resultado.
- Medición del tiempo de la suma SIMD, también con clock().
- Cálculo y visualización de la aceleración lograda por SIMD, comparando ambos tiempos.
- Registro de los resultados en un archivo tiempos.csv en formato secuencial,simd para su posterior análisis gráfico.

Resultados :

Especificaciones Sistema 1 (Julio Blacio)

Sistema: Acer Nitro AN515-58

SO: Microsoft Windows 11 Home x64

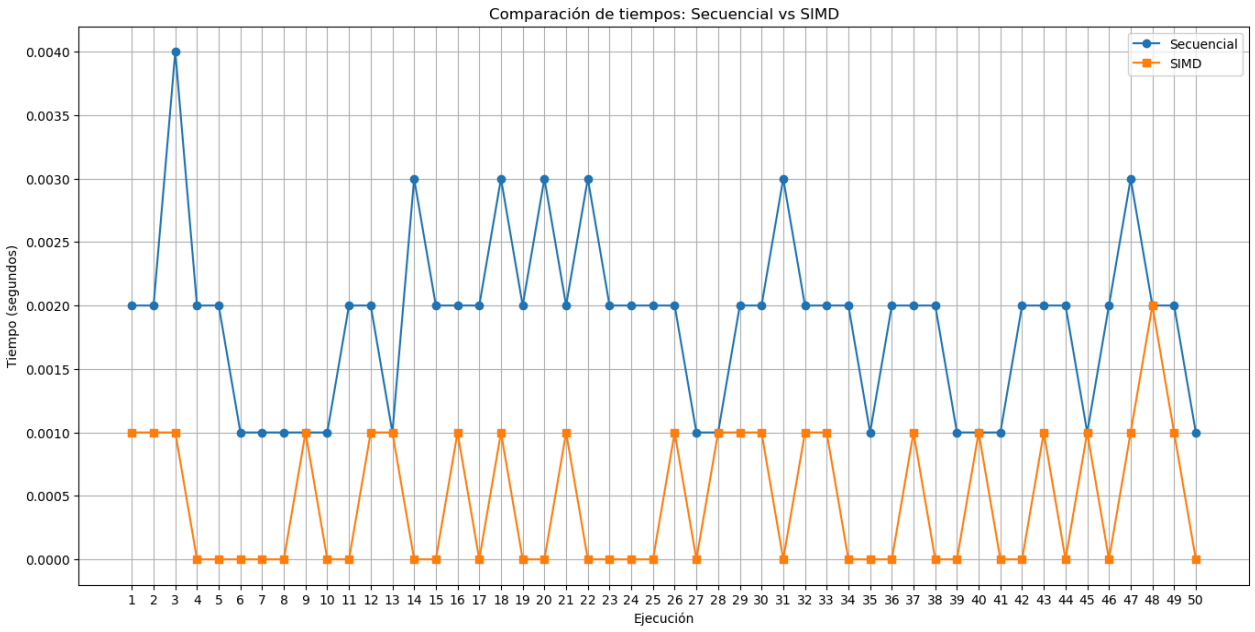
Procesador: 12th Gen Intel(R) Core(TM) i5-12500H, 3100 Mhz, 12 procesadores principales, 16 procesadores lógicos

Ram: 2X8 GB DDR5 4800Mhz

Disco Duro: NVME M.2 2280 512GB Samsung MZVL2512HCJQ-00B07

Compilador: CodeBlocks (GNU GCC Compiler)

Gráfico:



Especificaciones Sistema 2 (Erick Tufiño)

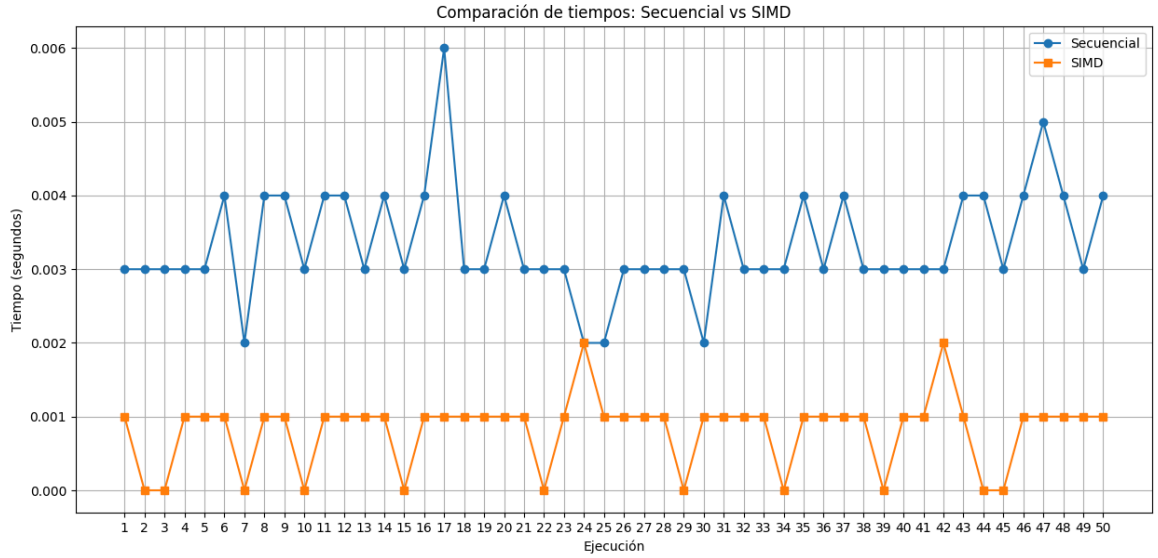
Sistema:Hp pavilion laptop 15-cc0xx

Procesador: Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz, 2904 Mhz, 2 procesadores principales, 4 procesadores lógicos

Ram: 16 GB

Disco Duro: Kingston snv2s1000g

Compilador: Dev C++



Especificaciones Sistema 3 (Ricardo Laínez)

Sistema: Dell Inc. Vostro 3400

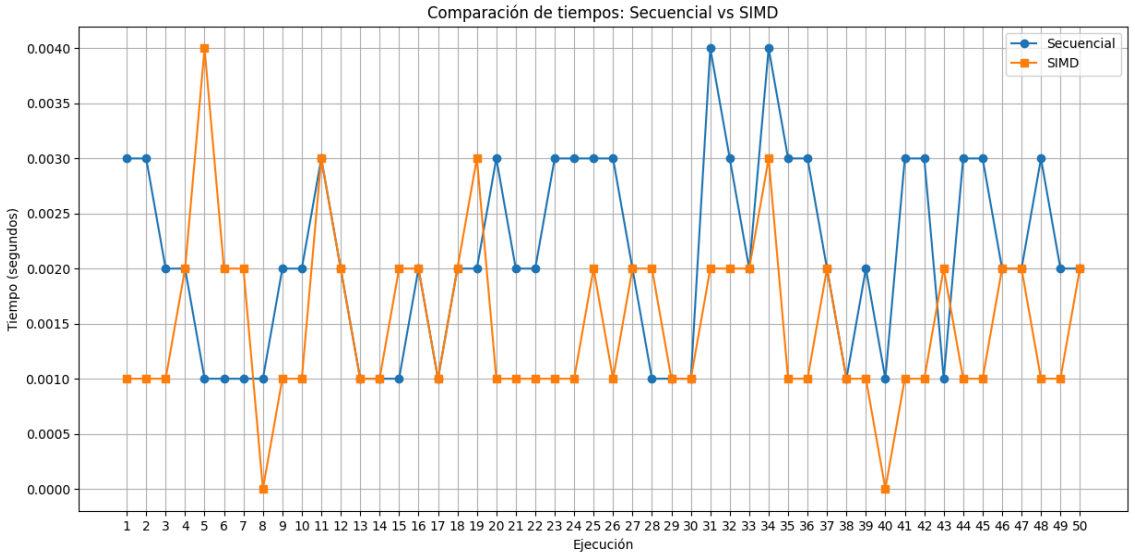
Procesador: 11th Gen Intel(R) Core(TM) i5-1135G7 @2.40GHz, 2419 Mhz, 4 procesadores principales, 8 procesadores lógicos

Ram: 8GB

Disco Duro: ST1000LM035-1RK172

Compilador: CodeBlocks

Gráfico:



Especificaciones Sistema 4 (Zaith Manangón)

Sistema: Dell Inc. Vostro 3400

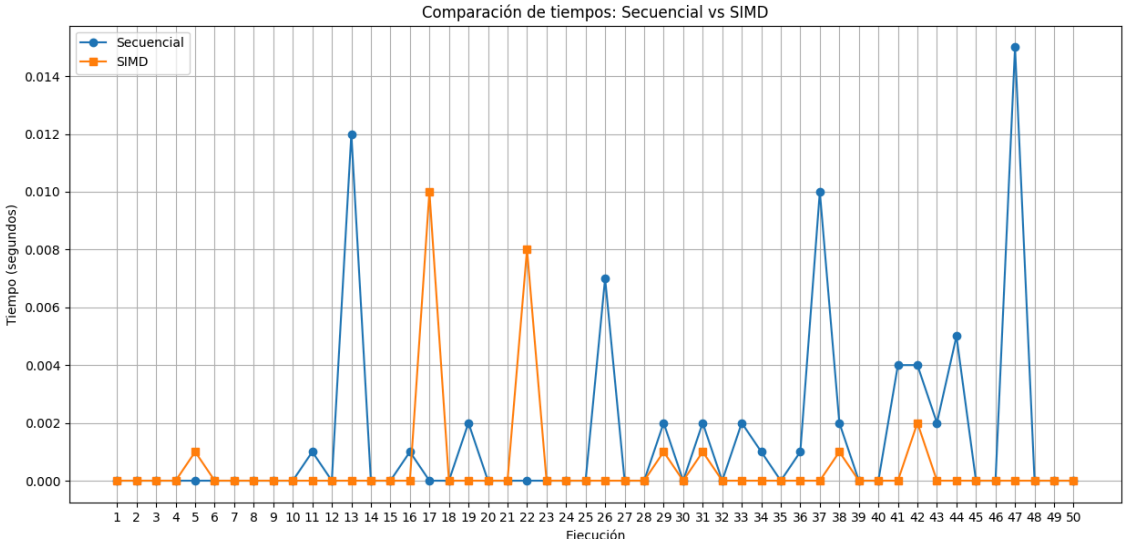
Procesador:11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 2.42 GHz, 2419 Mhz, 4 procesadores principales, 8 procesadores lógicos

Ram: 24 GB, 2667 Mhz

Disco Duro: ST1000LM035-1RK172

Compilador: CodeBlocks

Gráfico:



Análisis de resultados

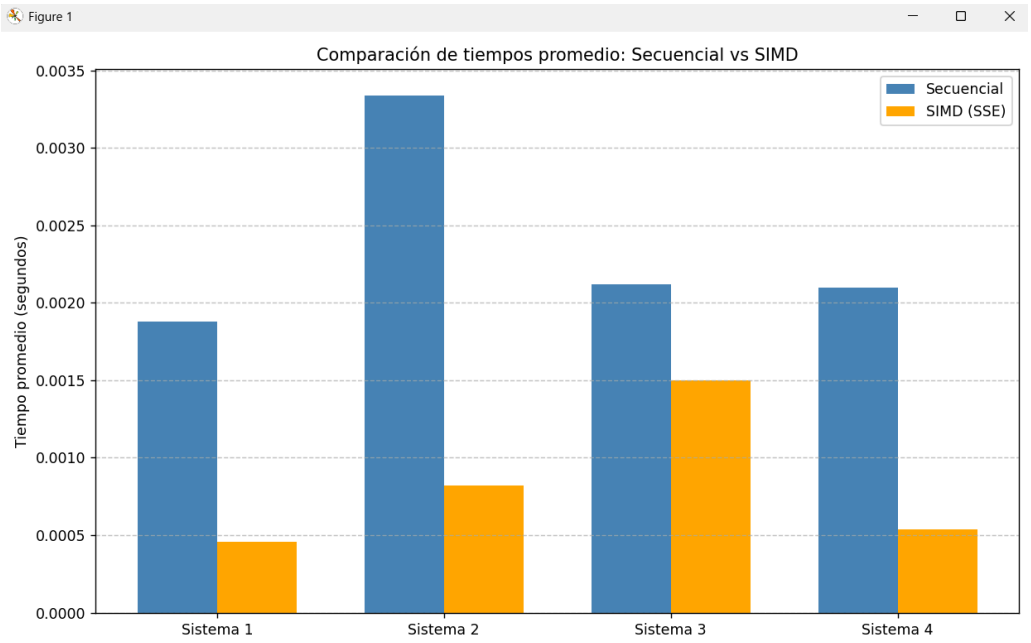
Se realizaron 50 ejecuciones por sistema para comparar dos enfoques de suma de vectores: uno secuencial (ciclo for) y otro usando instrucciones SIMD (SSE). Se utilizaron vectores de un millón de elementos, y los tiempos fueron medidos con clock(), luego registrados y graficados para evaluar el rendimiento relativo.

Análisis Comparativo:

Se construyó una tabla resumen con los tiempos promedio obtenidos por sistema, tanto para la versión secuencial como para la paralela con SIMD. Adicionalmente, se calculó el factor de aceleración (speedup) para cada uno.

Resumen de tiempos promedio y aceleración

Sistema	Secuencial (s)	SIMD (s)	Aceleración
Sistema 1	0.00188	0.00046	4.09x
Sistema 2	0.00334	0.00082	4.07x
Sistema 3	0.00212	0.00150	1.41x
Sistema 4	0.00210	0.00054	3.89x



- Los sistemas **1 (Blacio)** y **2 (Tufiño)** presentaron aceleraciones superiores a **4×**, evidenciando una excelente capacidad de paralelización. Esto se debe en gran parte a sus **procesadores modernos** y **RAM en doble canal con frecuencias elevadas**, que permiten un mayor flujo de datos hacia los registros SIMD. En estos casos, el tiempo

de ejecución se redujo en más del 75%, demostrando la eficiencia de SIMD cuando se aplica a operaciones vectoriales masivas y bien alineadas.

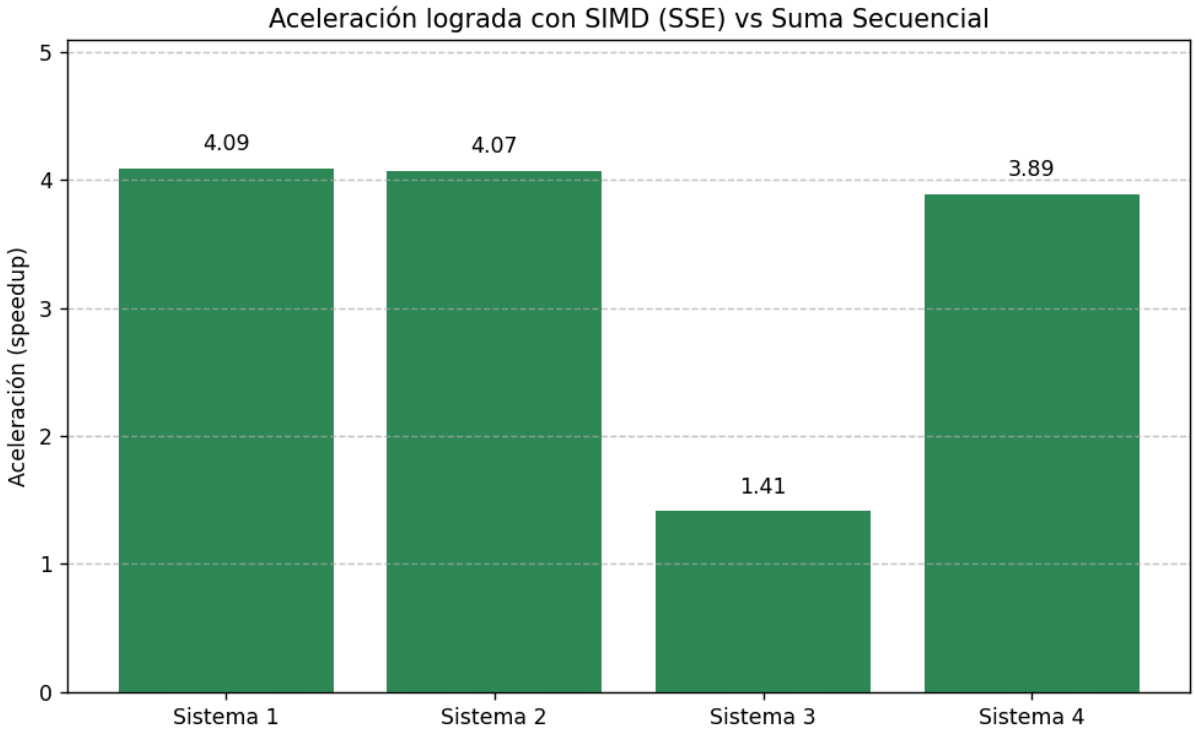
- El **sistema 3 (Laínez)** mostró un rendimiento inferior, con una aceleración de apenas **1.41×**. Esto puede atribuirse a su configuración de **RAM en un solo canal (8 GB)** o a **interferencias del sistema operativo** durante las mediciones. Este tipo de entorno limita el acceso a memoria eficiente, haciendo que el uso de SIMD no siempre resulte ventajoso. De hecho, en **tareas pequeñas o con cuellos de botella en memoria**, el overhead de carga y almacenamiento puede contrarrestar los beneficios del paralelismo.

- El sistema **4 (Manangón)**, pese a tener una configuración muy similar al sistema 3 pero con mayor capacidad de memoria (24 GB), presentó **tiempos anómalos cercanos a 0 segundos**. Este comportamiento puede explicarse por:

    - Limitaciones del método de medición, ya que clock() en C no siempre tiene resolución suficiente para capturar tiempos extremadamente cortos.
    - La ejecución SIMD puede ser tan rápida que se ubica por debajo del umbral mínimo de detección temporal del sistema.
    - Carga nula del sistema o condiciones de ejecución extraordinariamente limpias que afectan el realismo de las métricas.

Además, es importante considerar que las instrucciones SSE operan sobre bloques de **128 bits**, es decir, **4 valores float por instrucción**. Por lo tanto, si el tamaño del vector **no es múltiplo de 4**, los últimos elementos no pueden procesarse directamente con SIMD. En este experimento se evitó este problema al trabajar con exactamente **1 millón de elementos** (múltiplo de 4), pero en casos reales sería necesario agregar una rutina secuencial para manejar los residuos.

Para facilitar la interpretación, también se graficó el **factor de aceleración**, el cual muestra cuántas veces más rápido fue el método SIMD comparado con el método secuencial.



La gráfica muestra que **SIMD logra una aceleración significativa** en la mayoría de los sistemas, destacándose los sistemas 1 y 2, mientras que el sistema 3 resalta como caso atípico por su baja ganancia relativa.

1. **¿Cuánto se reduce el tiempo?**
    - La reducción del tiempo de ejecución al usar SIMD con instrucciones SSE fue significativa en la mayoría de los sistemas evaluados.
    - En los mejores casos (Sistema 1 y Sistema 2), el tiempo promedio se redujo de aproximadamente **0.00188 s a 0.00046 s** y de **0.00334 s a 0.00082 s**, respectivamente.
    - Esto representa una **aceleración de más del 75%**, alcanzando **speedups superiores a 4×**, lo que demuestra la efectividad de SIMD en operaciones vectoriales masivas y paralelizables.

2. **¿En qué casos SIMD no sería útil?**

SIMD puede no ser útil o incluso contraproducente en los siguientes casos:

- Cuando se trabaja con **tamaños de datos pequeños**, el overhead de cargar y almacenar en registros SIMD puede superar el beneficio obtenido.
- En operaciones que **no son paralelizables** (por ejemplo, aquellas con dependencias entre elementos).
- Si el sistema tiene **limitaciones en el acceso a memoria**, como una RAM de un solo canal o baja frecuencia (como ocurrió en el Sistema 3), lo que reduce el aprovechamiento de los registros SIMD.
- Cuando el compilador o el hardware **no soportan adecuadamente instrucciones SIMD**, o estas no se activan con los flags de optimización apropiados.

3. **¿Qué ocurriría con un vector que no sea múltiplo de 4?**

Las instrucciones SSE trabajan con registros de 128 bits, es decir, procesan 4 valores float (32 bits) simultáneamente. Si el tamaño del vector no es múltiplo de 4, los elementos finales no pueden ser procesados directamente con SIMD. En estos casos se deben aplicar soluciones como:

- Ejecutar una última parte del bucle de forma secuencial para los elementos sobrantes.
- Usar padding (relleno) para completar el múltiplo de 4.
- Detectar esa condición en tiempo de ejecución y adaptar el algoritmo (por ejemplo, con un if que cubra el resto).

En el presente experimento se evitó este problema utilizando exactamente 1,000,000 elementos, que sí es múltiplo de 4.

Conclusiones

- SIMD ofrece una **mejora considerable en el rendimiento** para operaciones vectoriales simples, especialmente cuando se trabaja con grandes volúmenes de datos.
- El impacto de SIMD varía con la configuración del sistema, incluyendo el **tipo de RAM, número de núcleos, y eficiencia del compilador**.
- Se validó que SIMD es especialmente útil cuando los datos pueden procesarse **en bloques alineados y sin dependencias**.

Recomendaciones

- Utilizar SIMD en tareas con **estructuras de datos homogéneas y sin dependencia entre elementos**.
- Asegurarse de que los vectores sean **múltiplos del tamaño del registro SIMD**, o implementar una rutina mixta para los residuos.
- Activar las optimizaciones del compilador (-msse, -03) para garantizar el uso de instrucciones SIMD.
- Considerar el uso de tecnologías **más modernas** como **AVX** (256 bits) o **AVX-512**, disponibles en procesadores de última generación, para ampliar la eficiencia del procesamiento paralelo.