|  | **Arquitectura de Computadores**  **Informe del Trabajo Final** |
| --- | --- |

**EL IMPACTO DE LA LOCALIDAD DE CACHÉ EN EL RENDIMIENTO DE LA MULTIPLICACIÓN DE MATRICES**

Jose Manuel Cardona - A00399980

Alejandro Mejía - A00399937

Martín Gómez Caicedo - A00399958

Juan Camilo Muñoz - A00399199

**Fecha de entrega**:

27 de noviembre de 2024

**Profesores**:

Juan Hernandez Delgado,

Duván García Cedeno

**INFORME DEL DESARROLLO DE EXPERIMENTO - EL IMPACTO DE LA LOCALIDAD DE CACHÉ EN EL RENDIMIENTO DE LA MULTIPLICACIÓN DE MATRICES**

**1. Introducción**

**1.1 Marco teórico**

**2. Definición del Problema y Objetivos**

**2.1 Identificación del problema**

**2.2 Objetivo general**

**2.3 Objetivos específicos**

**3. Selección de Factores**

**3.1 Identificación de factores primarios y secundarios**

**3.1.1 Especificaciones de computadores usados**

**4. Diseño Experimental**

**4.1 Elección del diseño experimental**

**5. Experimentación Preliminar**

**5.1 Desarrollo de experimentos preliminares**

**6. Ejecución del Experimento**

**6.1 Procedimiento experimental**

**7. Análisis de Datos**

**7.1 Registro y organización de los datos**

**7.2 Análisis estadístico**

**8. Juicio sobre los resultados**

**9. Conclusiones**

**9.1 Conclusión final y recomendaciones**

**10. Referencias**

1. INTRODUCCIÓN

La arquitectura de computadores define de manera fundamental el rendimiento de las aplicaciones al influir en el acceso y manejo de datos en la memoria. La interacción entre el procesador y la memoria caché es un aspecto clave para optimizar la eficiencia en operaciones intensivas como la multiplicación de matrices, en la que el tiempo de acceso a datos se convierte en un cuello de botella. A lo largo del curso, hemos estudiado cómo el principio de localidad —tanto espacial como temporal— afecta el desempeño, y cómo su aprovechamiento puede mejorar significativamente la ejecución de algoritmos. Este trabajo se centra en analizar el impacto de diferentes implementaciones de multiplicación de matrices sobre la caché para observar cómo cada versión aprovecha la jerarquía de memoria.

# 1.1 Marco teórico

El rendimiento computacional es un factor crítico en el desarrollo y ejecución de algoritmos, especialmente en aplicaciones que involucran operaciones matemáticas intensivas, como la multiplicación de matrices. La comprensión de la jerarquía de memoria y el principio de localidad resulta esencial para optimizar el desempeño de tales algoritmos.

**Jerarquía de memoria**

La jerarquía de memoria es una estructura que organiza los diferentes niveles de almacenamiento en un sistema computacional, desde los registros y cachés de alta velocidad hasta la memoria principal y dispositivos de almacenamiento masivo. Esta organización responde a la necesidad de equilibrar el costo y la velocidad de acceso, siendo la memoria caché un recurso clave para reducir la latencia.

**Principio de localidad**

Este principio describe cómo los programas acceden a datos de manera que se benefician del almacenamiento en caché. Existen dos tipos principales:

* **Localidad temporal:** Accesos repetidos a los mismos datos en un intervalo breve.
* **Localidad espacial:** Accesos secuenciales o cercanos en memoria.

**Multiplicación de matrices y acceso a memoria**

La multiplicación de matrices, una operación básica en muchas aplicaciones científicas y de aprendizaje automático, ilustra la importancia del principio de localidad. Algoritmos bien diseñados, como los que usan técnicas de multiplicación por bloques, maximizan la localidad, minimizando los costos de acceso a memoria y aprovechando las capacidades de la memoria caché. Las estrategias que ignoran estos principios suelen resultar en altos tiempos de espera, debido a accesos más frecuentes a la memoria principal

**Algoritmos de multiplicación de matrices**

La multiplicación de matrices es una operación intensiva en cómputo que involucra múltiples accesos a memoria. Existen diferentes implementaciones, incluyendo:

* **Algoritmo básico (Triple Bucle):** Implementa el cálculo directo de ] , con una complejidad
* **Bloqueo:** Divide las matrices en bloques para procesar subconjuntos que caben en caché, mejorando la localidad.

**Análisis de Varianza (ANOVA)**

Se trata de un método estadístico que se utiliza para evaluar si hay diferencias importantes entre los tratamientos o condiciones de un experimento. Facilita la identificación de los factores que influyen significativamente en los resultados obtenidos.

**Tiempo Normalizado**

Donde:

* es el tiempo normalizado
* es el tiempo observado
* es el tamaño de un problema
* es el número de operaciones llevadas a cabo.

El tiempo normalizado es una métrica de rendimiento que se utiliza para evaluar y comparar la eficiencia de diferentes algoritmos o implementaciones, especialmente cuando se manejan tareas con distintos tamaños o niveles de complejidad. Esta normalización ajusta el tiempo de ejecución teniendo en cuenta el tamaño del problema o la cantidad de trabajo realizado, lo que permite una comparación justa y significativa entre diversos escenarios.

# 1.2 Diseño y conducción de experimentos (DoE)

El diseño de experimentos es una metodología estructurada para investigar la relación entre variables de entrada (factores) y de salida (respuestas). En el contexto de este proyecto:

1. **Factores primarios:** Versiones del algoritmo, tamaño de la matriz, tipo de dato (float/double) y arquitectura (x86/x64).
2. **Factores secundarios o de ruido:** Variaciones en hardware y software que no se controlan directamente.
3. **Metodología:** Experimentos factoriales completos o fraccionados para explorar todas las combinaciones de factores

2. DEFINICIÓN DEL PROBLEMA Y OBJETIVOS

**2.1 Identificación del problema**

El problema que abordamos es cómo la estructura y la implementación de un algoritmo de multiplicación de matrices afecta su rendimiento en función de la interacción con la memoria caché. A pesar de que varias versiones de este algoritmo poseen la misma complejidad algorítmica, se espera que presenten diferencias en desempeño debido a la forma en que explotan o desaprovechan la localidad de caché.

**2.2 Objetivo general**

Evaluar el impacto de la localidad de caché en el rendimiento de distintas implementaciones de algoritmos de multiplicación de matrices equivalentes, que comparten la misma complejidad algorítmica, pero presentan distintos niveles de aprovechamiento del principio de localidad. Para ello se aplicará la metodología de conducción de experimentos y se apoyará en la teoría del curso de Arquitectura de Computadores. Al final, y de acuerdo a los resultados obtenidos, se propondrá una mejora en aquel algoritmo menos eficiente mediante la técnica de multiplicación por bloques, y se determinará si en efecto se obtiene un incremento en el rendimiento del mismo.

**2.3 Objetivos específicos**

1. Analizar el comportamiento de seis versiones de algoritmos de multiplicación de matrices y su relación con el sistema de caché.
2. Determinar la interacción entre las características de la arquitectura de memoria y los distintos algoritmos en términos de tiempos de ejecución.
3. Implementar una mejora basada en la multiplicación por bloques para la versión más ineficiente y evaluar su efectividad.

**2.4 Hipótesis Planteadas**

Para el ANOVA que se realizará más adelante evaluando si la interacción entre los 4 factores tiene un efecto sobre el tiempo normalizado se plantean las siguientes hipótesis:

1. **Hipótesis Nula (H₀):** Los factores version, size, type, y ISA\_type, así como sus interacciones, **no tienen un efecto significativo** en la media del tiempo normalizado del bloque.
2. **Hipótesis Alternativa (H₁):** Al menos uno de los factores (version, size, type, ISA\_type) o sus interacciones **tienen un efecto significativo** en la media del tiempo normalizado del bloque.

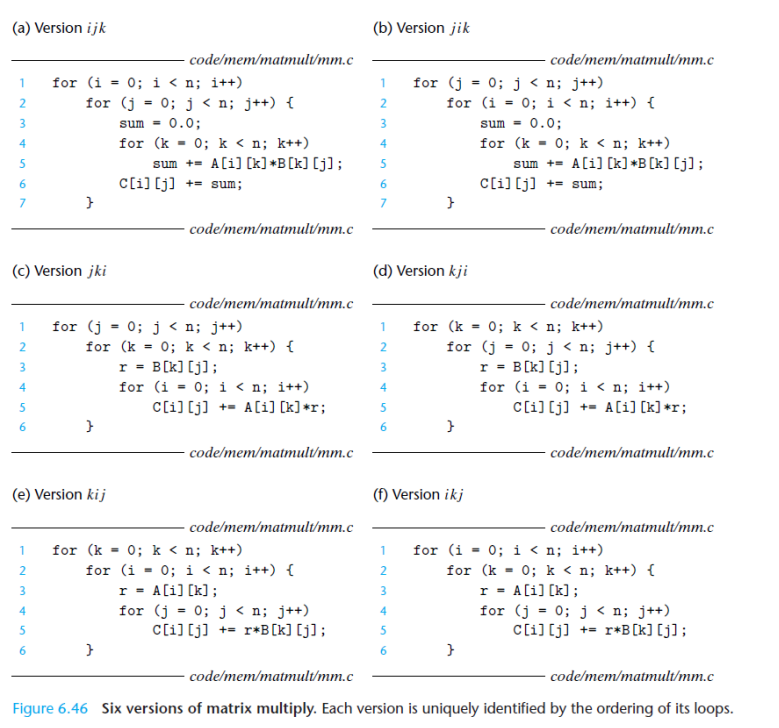
**3. SELECCIÓN DE FACTORES**

**3.1 Identificación de factores primarios y secundarios**

**Factores primarios:**

1. **Versión del Algoritmo**

Cada una de las seis versiones (a, b, c, d, e, y f) implementa la multiplicación de matrices utilizando diferentes técnicas. Estas versiones varían en su estructura de bucles, estrategias de acceso a memoria, y aprovechamiento del principio de localidad:



* **Relevancia:** Comparar estas versiones permite identificar cuál de ellas optimiza mejor el uso de la jerarquía de memoria y minimiza los fallos de caché.
* **Objetivo:** Evaluar cuál es más eficiente y proponer mejoras basadas en el análisis de su comportamiento

1. **Tamaño de la Matriz de Datos**

Este factor considera las dimensiones n×nn \times nn×n de las matrices cuadradas. Los tamaños pequeños pueden caber completamente en la memoria caché, mientras que los tamaños grandes obligan a realizar accesos frecuentes a la memoria principal.

* **Relevancia:** Explorar cómo el crecimiento del tamaño afecta la tasa de fallos de caché y los tiempos de ejecución.
* **Objetivo:** Determinar el umbral a partir del cual la memoria caché deja de ser efectiva, proporcionando datos clave para optimizar los algoritmos

1. **Tipo de Dato**

**Float (Flotante):** Ocupa 32 bits por valor**.**

**Double (Doble precisión):** Ocupa 64 bits por valor, permitiendo mayor precisión.

* **Objetivo:** Comparar el impacto del tipo de dato en el rendimiento y en la utilización eficiente de la memoria.

1. **Tipo de ISA (Instruction Set Architecture)**

Las arquitecturas x86 y x64 difieren en su capacidad para manejar instrucciones y datos. Las arquitecturas x64 soportan registros y palabras de memoria más grandes, lo que puede mejorar la eficiencia en ciertos casos.

* **Relevancia:** Identificar cómo las diferencias entre ISAs afectan el aprovechamiento de caché y el rendimiento general.
* **Objetivo:** Determinar si el cambio de ISA genera ventajas significativas para el rendimiento de los algoritmos

**Factores Secundarios:**

Los factores secundarios incluyen el estado de la caché antes de cada ejecución y las variaciones en el tiempo de acceso de la memoria. Estos factores serán controlados mediante la normalización de tiempos de ejecución y el uso de configuraciones de hardware específicas. Además se han identificado los siguientes factores secundarios agrupados en categorías, junto con sus medidas de control.

1. **Hardware:**

* **Estado de la caché:** El estado de la caché puede afectar el tiempo de ejecución, por ello se limpiará la caché antes de cada ejecución para mantener la consistencia.
* **Temperatura del procesador:** La temperatura puede influir en el rendimiento debido al throttling térmico. Este factor no es fácilmente controlable.
* **Variaciones en el rendimiento del disco duro o SSD:** Afecta el tiempo de carga de grandes volúmenes de datos.
* **Interferencia en el consumo de energía:** Subidas repentinas de energía pueden afectar la consistencia del hardware.

1. **Software:**

* **Procesos en segundo plano:** Procesos adicionales e independientes a los necesarios para realizar el experimento pueden consumir recursos que pueden traducirse en un gran ruido a la hora de tomar los datos, para esto se intentara suprimir la mayor cantidad de procesos en segundo plano como puede ser antivirus, aplicaciones de terceros, actualizaciones automáticas del sistema, etc.
* **Versión del sistema operativo:** Cambios recientes pueden afectar al rendimiento de un mismo computador.
* **Uso de distintos entornos de ejecución:** Se mantendrá constante el entorno de ejecución a usar.

1. **Entorno:**

* **Temperatura ambiental:** Puede afectar la capacidad de enfriamiento del sistema.
* **Ruido Electromagnético:** Las interferencias pueden afectar el hardware.

1. **Configuración del sistema:**

* **Frecuencia de reloj de la CPU:** Algunos sistemas ajustan dinámicamente la frecuencia según la carga que se le de al computador, por ello se establecerá una frecuencia de valor fijo durante todo el experimento.
* **Plan de energía:** Los ajustes de energía pueden influir en el rendimiento, por esto se usará el plan de energía “alto rendimiento” y se evitarán cambios posteriores en esta configuración.
* **Configuración de la memoria virtual:** El uso de paginación puede variar el acceso a memoria, entonces se ajustará la memoria virtual al mínimo requerido para las pruebas y así evitar sobrecarga de RAM.

A Continuación se especifica el hardware de los dos equipos que se usarán durante la toma de datos.

**3.1.1 Especificaciones de computadores usados**

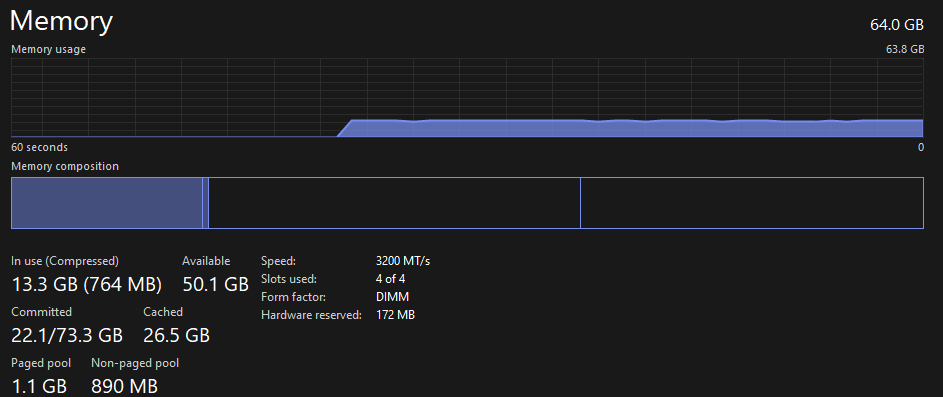
*Ambos computadores tienen de sistema operativo Windows 11 pro*

**Computador 1 (Escritorio):**

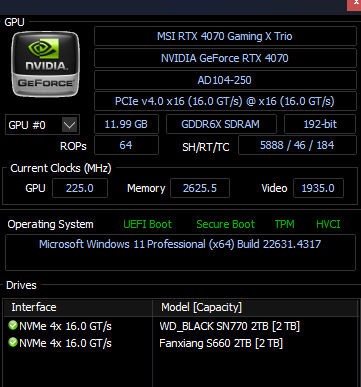
* CPU: Core I7 12700KF



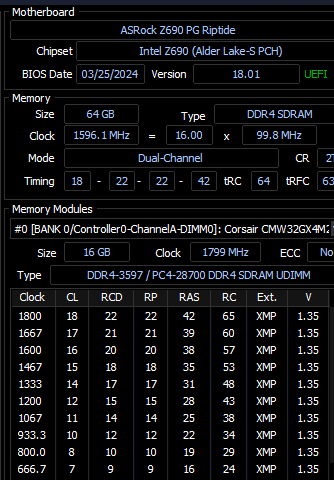
* RAM: 64GB DRR4 3200MT/s



* Tarjeta gráfica: Nvidia RTX 4070, Almacenamiento: 4TB (Nvme M.2 2TB x 2)



* Motherboard: ASRock Z690 PG Riptide

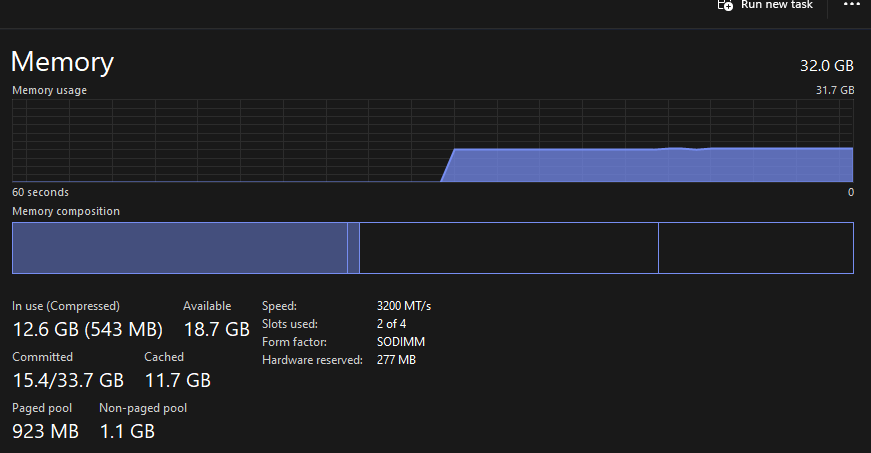


**Computador 2 (Pórtatil) :**

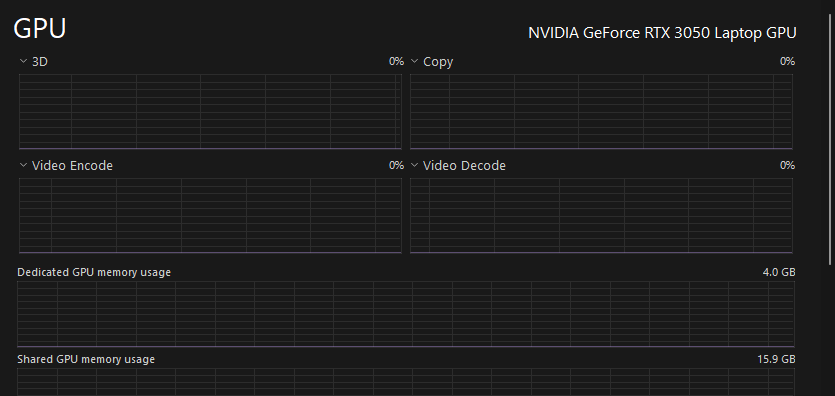
* CPU: Core i5-11260H

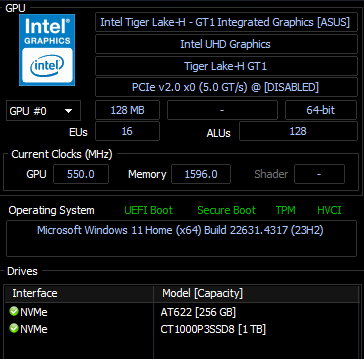


* RAM: 32GB DDR4 3200 MT/s

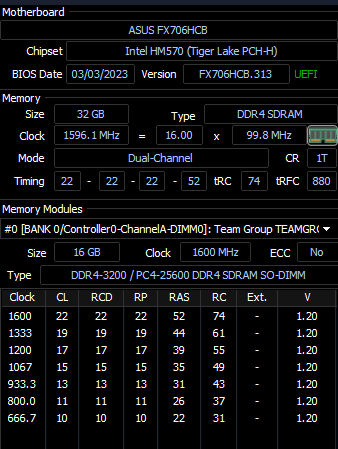


* Tarjeta Gráfica RTX 3050, Almacenamiento: 1.25TB (Nvme M.2)





* Motherboard: ASUS FX706HCB



4. DISEÑO EXPERIMENTAL

**4.1 Elección del diseño experimental**

El diseño experimental seleccionado para este estudio es un diseño factorial completo, debido a su capacidad para analizar simultáneamente los efectos principales e interacciones entre los diferentes factores identificados: la versión del algoritmo, el tamaño de las matrices, el tipo de dato (float o double) y el tipo de Instruction Set Architecture (ISA). Este enfoque es adecuado para nuestra hipótesis, ya que busca entender cómo cada uno de estos factores, tanto de manera individual como en combinación, afecta el rendimiento del sistema en términos de tiempo de ejecución.

La elección de este diseño se basa en las siguientes consideraciones:

* **Complejidad del fenómeno estudiado:** La interacción entre factores como la implementación del algoritmo y las características del hardware puede ser altamente no lineal. El diseño factorial permite identificar no solo los efectos aislados de cada factor, sino también cómo interactúan entre sí, algo crucial para comprender la relación entre la localidad de caché y el rendimiento.
* **Optimización del uso de recursos:** Si bien un diseño factorial implica un número considerable de pruebas, su eficiencia radica en la posibilidad de obtener información rica y detallada de los factores con un conjunto bien estructurado de experimentos.
* **Aplicabilidad en entornos controlados:** Los experimentos se realizarán en dos computadoras con configuraciones específicas y controladas (ver sección 3.1.1), lo que asegura que las variaciones observadas sean atribuibles a los factores seleccionados y no a variables externas.
* **Relevancia para la hipótesis:** Este diseño permite contrastar de manera directa la hipótesis planteada sobre cómo la localidad de caché influye en el rendimiento de las implementaciones de multiplicación de matrices. Además, las pruebas están organizadas para maximizar la validez interna de los resultados.

**Detalles del diseño**

* **Tamaño muestral y repeticiones:** Cada combinación de factores será evaluada en múltiples iteraciones (mínimo tres repeticiones por combinación) para garantizar la fiabilidad estadística de los resultados y reducir el impacto de la variabilidad aleatoria.
* **Orden de los ensayos:** Se emplea un orden aleatorizado para evitar sesgos debido a posibles efectos de arrastre o influencias no controladas durante el experimento.
* **Bloqueo:** Aunque el diseño factorial en su esencia no requiere bloqueo, en este caso se introducirá un criterio de bloqueo basado en el tipo de computadora (escritorio o portátil), ya que las especificaciones de hardware pueden influir significativamente en los resultados.
* **Matriz de diseño:** La matriz factorial incluirá todas las combinaciones posibles de los niveles de los factores, lo que garantiza un análisis exhaustivo ya que permite analizar cómo las distintas configuraciones de los niveles de los factores afectan al tiempo de ejecución de los algoritmos. Para facilitar dicha matriz se usará un código en Python Colab.

| Factor | Niveles |
| --- | --- |
| Versión del Algoritmo | a, b, c, d, e, f (6 Niveles) |
| Tamaño de la Matriz | 170, 440, 565, 670, 780, 915, 1185, 1470, 1700, 2050 (10 Niveles) |
| Tipo de Dato | float, double (2 Niveles) |
| ISA | x86, x64 (2 Niveles) |

El diseño factorial completo evalúa todas las combinaciones de estos niveles. El número total de combinaciones es el producto de los niveles de cada factor:

Estas combinaciones se realizarán por cada computador, es decir, PC y Laptop, además de que se harán en un orden completamente aleatorio.

Nuestra matriz de diseño, aleatorizada, quedó entonces como: [Matriz de Diseño](https://github.com/Electromayonaise/Rendimiento-y-Localidad-de-Cache/blob/main/doc/Results/Design%20Matrix.xlsx)

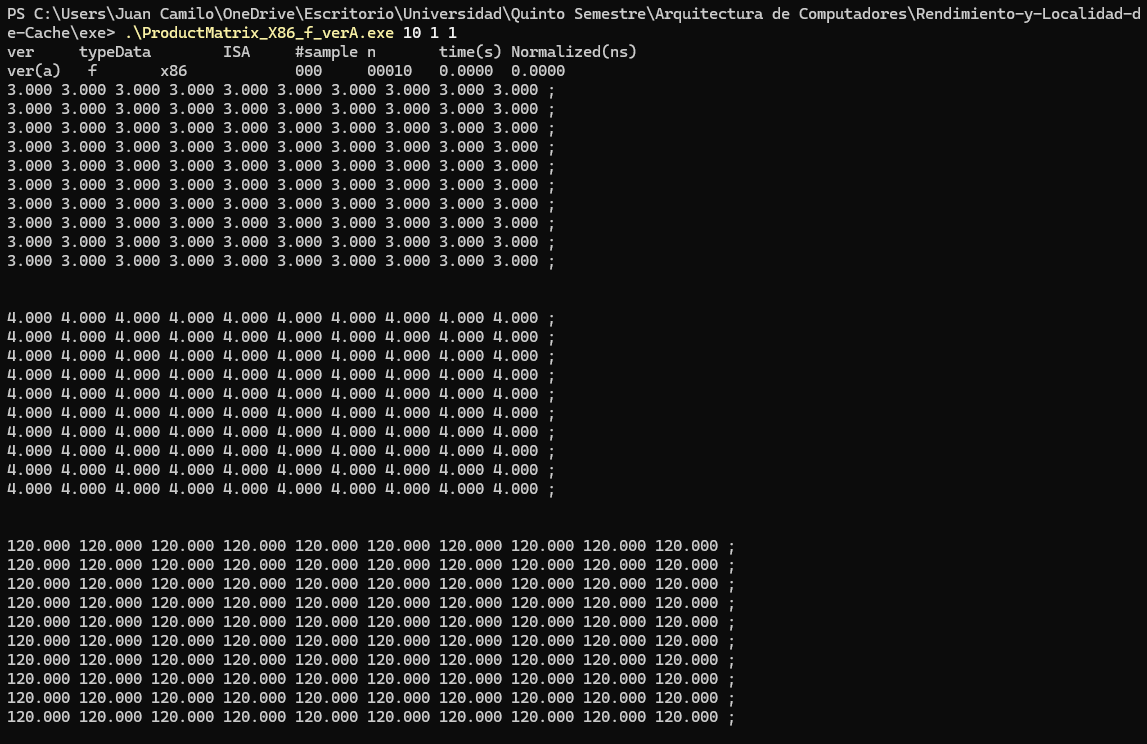
**5. E**XPERIMENTACIÓN PRELIMINAR

**5.1 Desarrollo de experimentos preliminares**

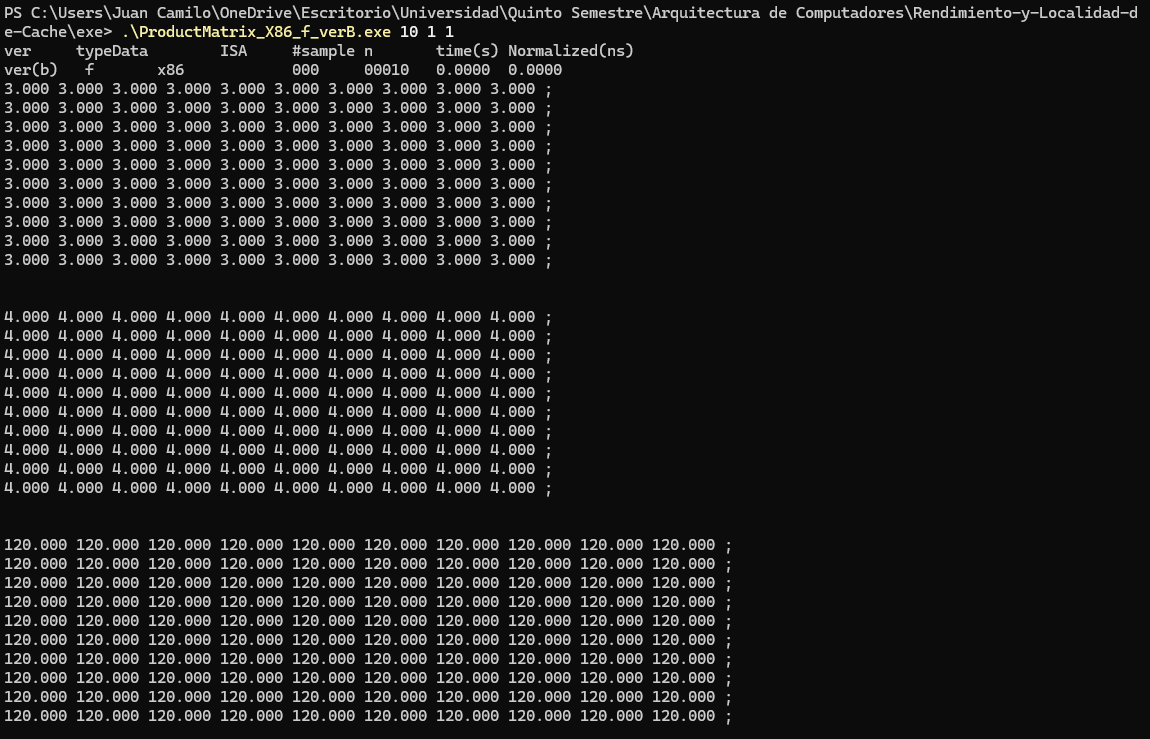
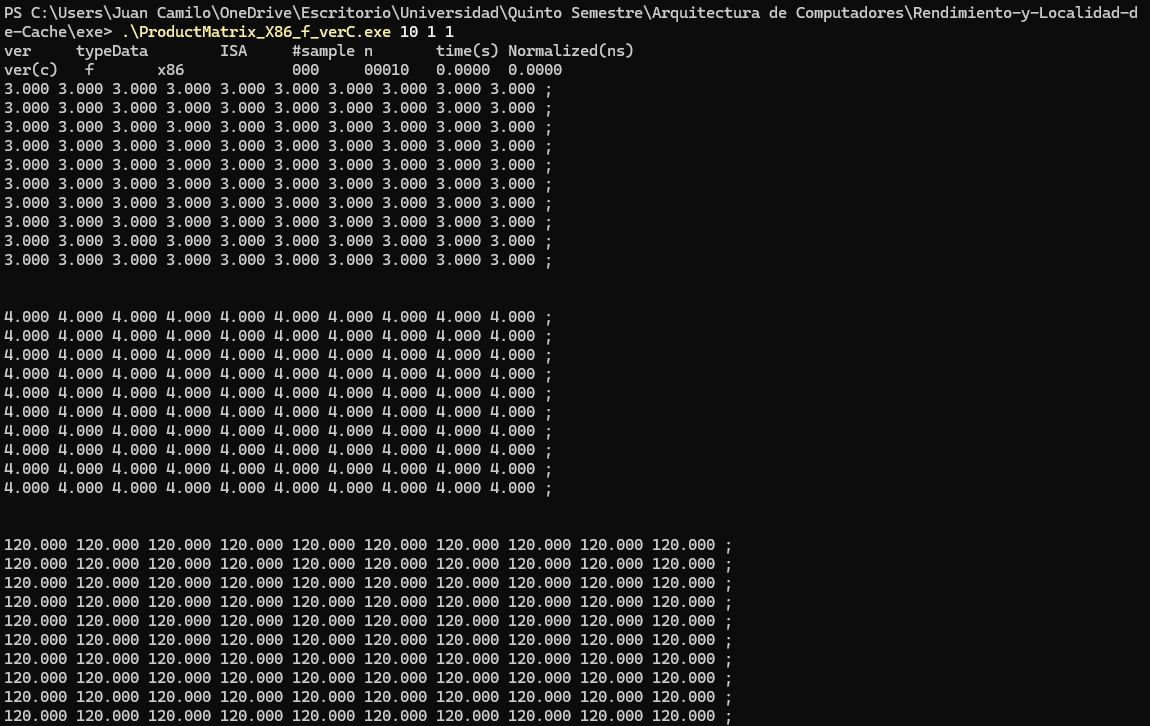
# Verificación del correcto funcionamiento de los Algoritmos

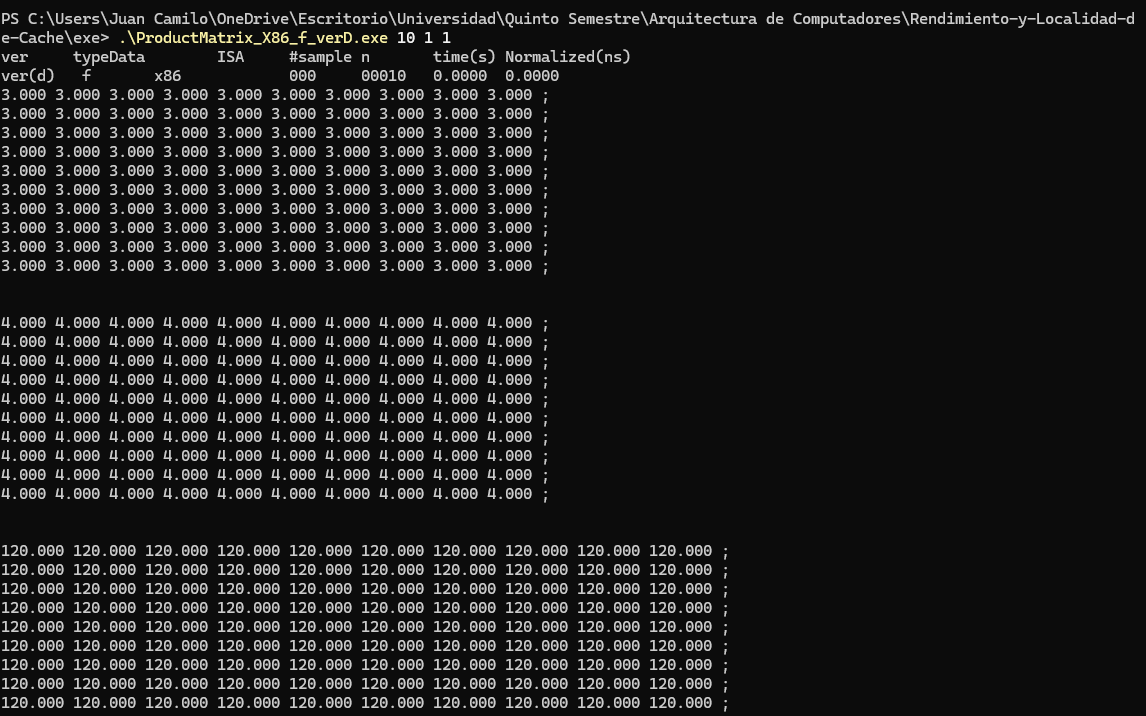
Como primer paso en la experimentación, luego de diseñar los códigos para cada versión del algoritmo de multiplicación de matrices en C++, se procedió a comprobar el correcto funcionamiento de estos. Para ello, se realizaron pruebas iniciales multiplicando una matriz 10x10 llena de 3 por otra matriz 10x10 llena de 4. El resultado esperado debe ser una matriz 10x10 llena de 120. En las siguientes imágenes se muestran los resultados obtenidos, confirmando la correcta ejecución de los algoritmos.

Algoritmo A

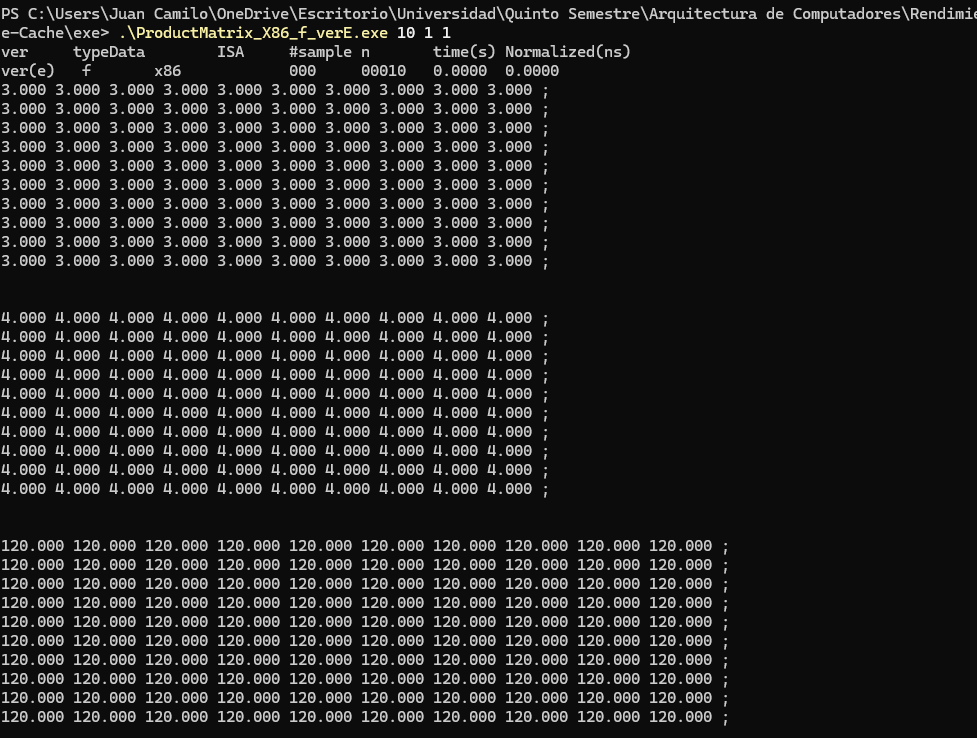


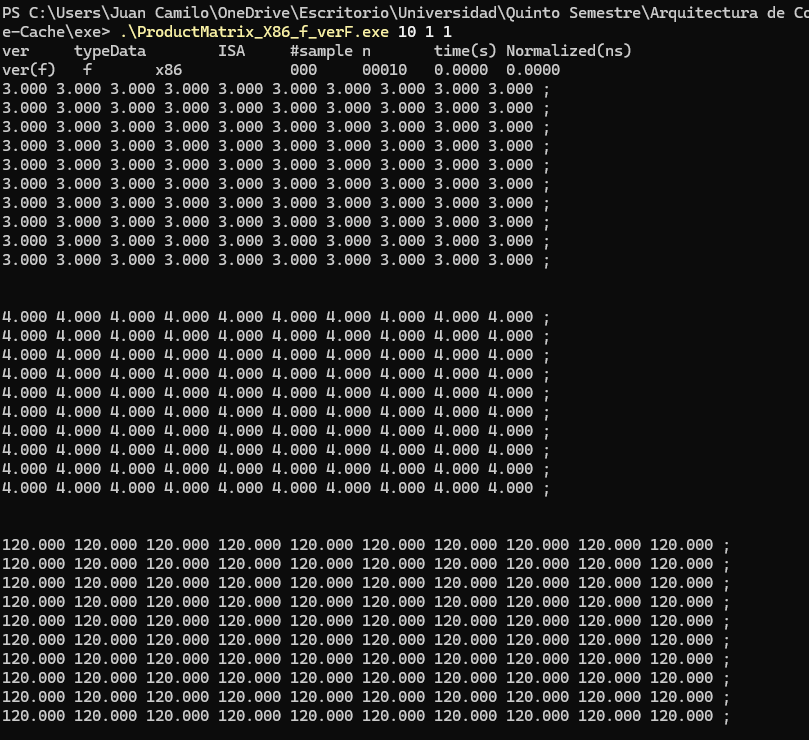
Algoritmo B

  
  
Algoritmo C  
  
  
Algoritmo D



Algoritmo E

  
  
Algoritmo F



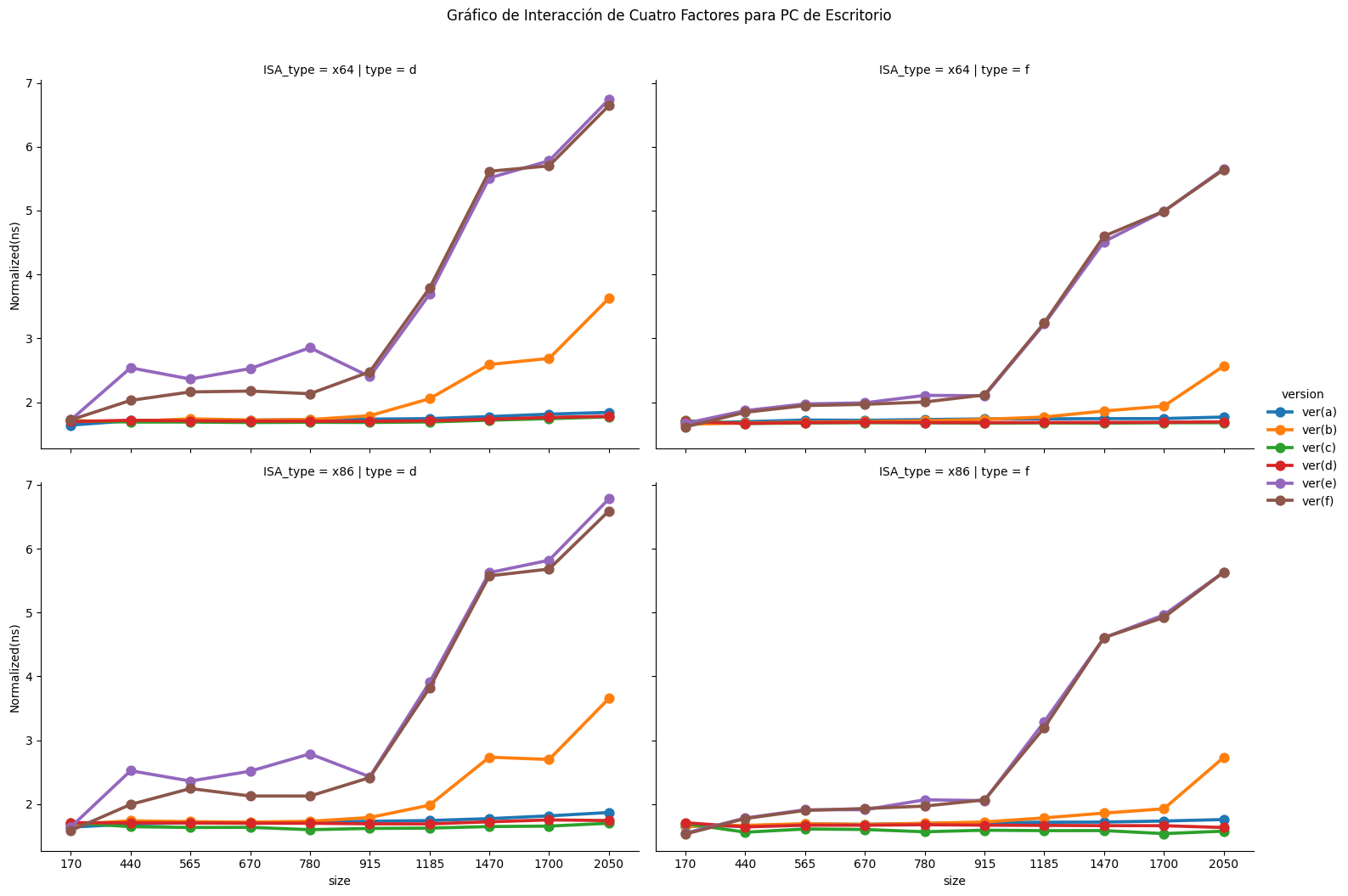
# Resultados Preliminares del ANOVA

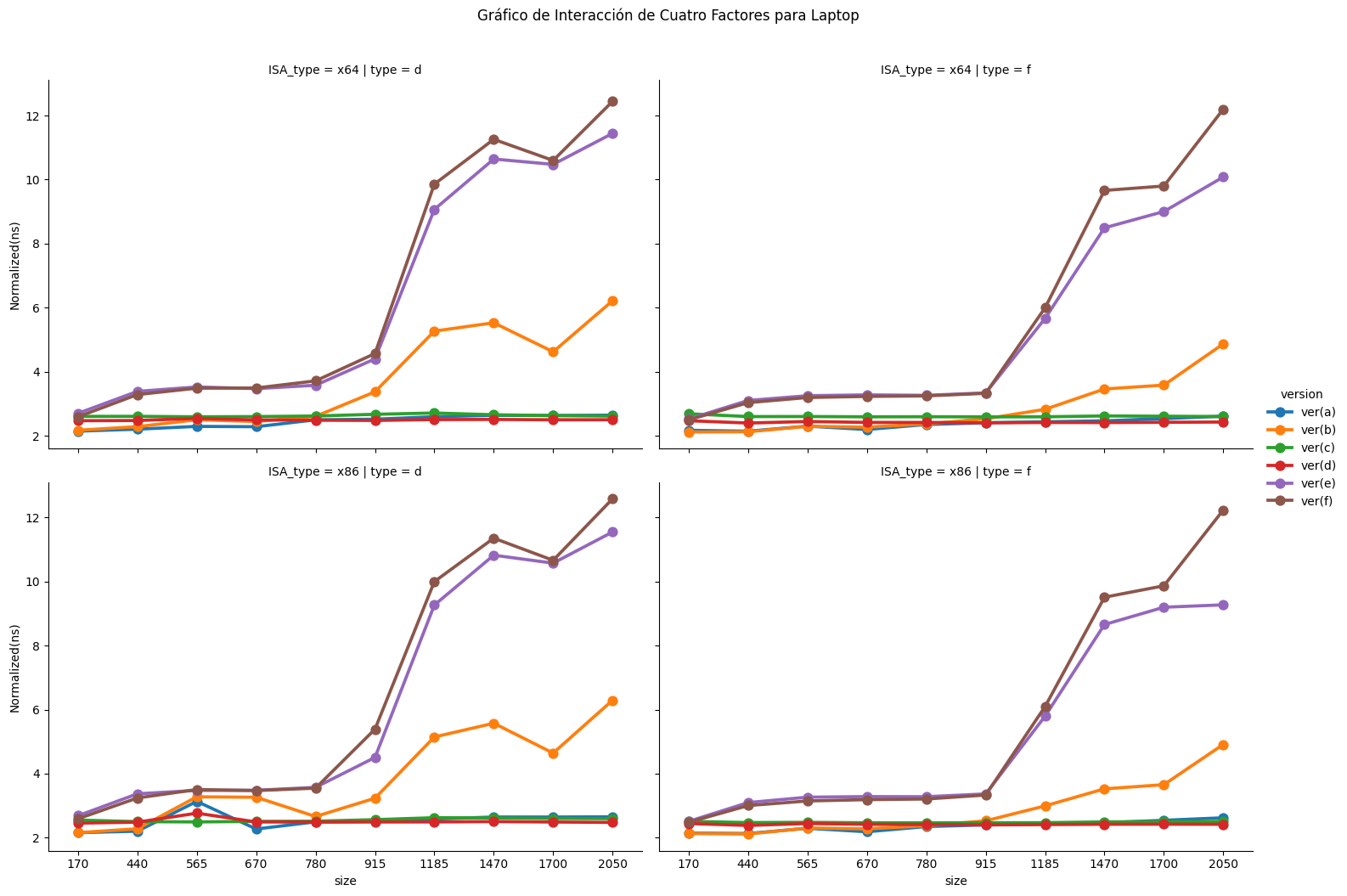
Se realizó un experimento preliminar de [ANOVA](https://colab.research.google.com/drive/18dCM5z5K6gqoZaHsqUFxp-jhTRmg9BOj#scrollTo=qDP6GplryicL) de cuatro vías con los datos dados inicialmente los cuales incluyen las combinaciones con los siguientes factores:

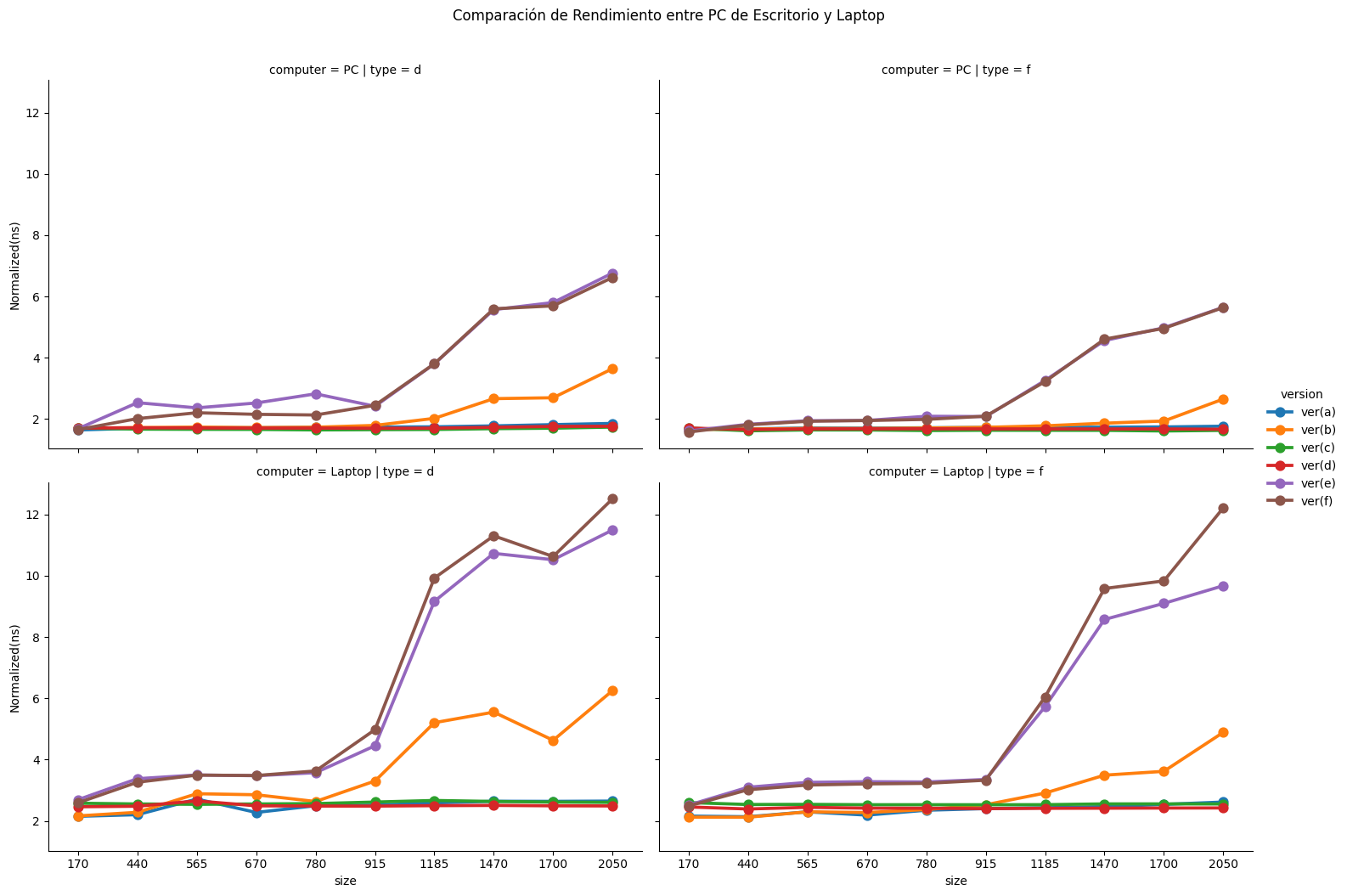
matrix\_sizes = [170, 440, 565, 670, 780, 915, 1185, 1470, 1700, 2050]

samples = [15, 15, 15, 15, 10, 10, 7, 5, 3, 3]

Los resultados fueron bastante prometedores a simple vista, con unas gráficas que mantiene cierta coherencia:







Las gráficas resultantes no tienen saltos bruscos ni cortes repentinos lo cual ya nos sugiere que los tamaños de las matrices son correctas y no necesitan una modificación posterior.

Según los datos arrojados del [ANOVA](https://colab.research.google.com/drive/18dCM5z5K6gqoZaHsqUFxp-jhTRmg9BOj#scrollTo=_PNd5L8mx3f_) se puede empezar a concluir lo siguiente:

# Conclusiones Preliminares Generales Basadas en los resultados del ANOVA

1. **Impacto de la Versión del Algoritmo (version):**

* El análisis ANOVA muestra que la versión del algoritmo tiene un efecto altamente significativo sobre el rendimiento, tanto en la PC de Escritorio como en la Laptop. Este resultado implica que las diferencias entre versiones del algoritmo afectan sustancialmente el tiempo de ejecución.
* En la PC de Escritorio, las versiones ver(e) y ver(f) registran tiempos promedio de 2.6021ns y 2.4726ns, respectivamente, siendo las menos eficientes. En la Laptop, estas mismas versiones alcanzan valores promedio de 4.3253ns y 4.4608ns, lo que sugiere un problema de rendimiento persistente en ambas máquinas, que podría abordarse optimizando estas versiones.

1. **Efecto del Tamaño de la Matriz (size):**

* El tamaño de la matriz también es un factor significativo, en ambos sistemas. Esto es consistente con la expectativa de que matrices más grandes requieren tiempos de procesamiento más largos.
* En la PC de Escritorio, el tiempo promedio aumenta de 1.6662ns para matrices de tamaño 170 a 3.4469ns para matrices de tamaño 2050. La Laptop muestra un patrón similar, aunque con tiempos más elevados, desde 2.4154 ns hasta 6.0315ns. Estos resultados destacan la necesidad de adaptar el algoritmo para manejar eficientemente tamaños de matriz grandes.

1. **Efecto del Tipo de Dato (type):**

* El tipo de dato (float o double) tiene un impacto significativo en el rendimiento, con float superando a double en ambas máquinas. Esto se refleja en los valores de F altos y p < 0.001, indicando que el uso de float es generalmente más eficiente en términos de tiempo de ejecución.
* En la PC de Escritorio, el promedio para float es de 1.9024 ns frente a 2.0797ns para double. En la Laptop, los tiempos son más altos, con un promedio de 2.9654 ns para float y 3.3662ns para double. Este resultado sugiere que float es una opción preferible cuando la precisión de double no es esencial, especialmente en contextos donde el rendimiento es una prioridad.

1. **Efecto de la Arquitectura de Instrucción (ISA\_type):**

* Aunque el tipo de arquitectura (x64 vs. x86) también es significativo (p < 0.001), su impacto en el rendimiento es menor en comparación con los otros factores.
* En la PC de Escritorio, x64 registra un tiempo promedio de 2.0034ns, mientras que x86 tiene un tiempo ligeramente menor de 1.9786ns. En la Laptop, las diferencias son aún más sutiles, con 3.1542ns para x64 y 3.1774ns para x86. Estos resultados sugieren que la arquitectura no es el principal determinante del rendimiento en este contexto.

1. **Interacciones entre Factores:**

* Las interacciones entre version, size, y type son significativas, lo que sugiere que el efecto de un factor puede depender de los valores de los otros. Por ejemplo, algunas versiones del algoritmo están mejor optimizadas para ciertos tamaños de matriz y tipos de dato.
* Las interacciones de tres y cuatro factores también son significativas, aunque con valores F más bajos. Esto sugiere que existe una combinación óptima de version, size, type, y ISA\_type que puede maximizar el rendimiento en cada sistema.

# Comparaciones Preliminares de Desempeño entre PC de Escritorio y Laptop

1. **Tiempos Promedio por Versión del Algoritmo:**

* Los tiempos promedio de ejecución son consistentemente mayores en la Laptop en comparación con la PC de Escritorio. Por ejemplo, ver(e) en la PC tiene un tiempo de 2.6021ns, mientras que en la Laptop es de 4.3253ns, lo cual representa un aumento significativo.
* La diferencia en el rendimiento entre ambas plataformas podría atribuirse a la menor capacidad de procesamiento y las limitaciones de la Laptop, indicando que la PC de Escritorio es una opción más adecuada para cargas de trabajo intensivas.

1. **Desviación Estándar y Consistencia:**

* La Laptop muestra una mayor variabilidad en los tiempos de ejecución, especialmente para ver(e) y ver(f), con desviaciones estándar de 2.3768ns y 2.7221ns, respectivamente. En contraste, la PC de Escritorio tiene desviaciones estándar mucho menores para estas mismas versiones, de 1.1549ns y 1.1702ns, lo que sugiere un rendimiento más estable en este equipo.
* Esta diferencia en variabilidad destaca que el rendimiento en la Laptop es menos predecible, probablemente debido a factores de hardware como la gestión de memoria y la potencia de procesamiento.

1. **Diferencia en Rendimiento por Tipo de Dato:**

* Ambos sistemas muestran tiempos menores al utilizar float en lugar de double, pero esta diferencia es más pronunciada en la PC de Escritorio. Esto podría reflejar una optimización de hardware superior en la PC para manejar tipos de datos de precisión simple.
* En la PC, float reduce el tiempo promedio en aproximadamente 0.1773ns frente a double, mientras que en la Laptop esta diferencia es de 0.4008ns, lo cual refuerza la ventaja de float en ambos sistemas.

Aunque de manera general, atribuimos estos resultados a las diferencias en hardware, dado que las especificaciones del PC de Escritorio son considerablemente superiores.

# Análisis del Número de Repeticiones

Ahora, si bien los datos iniciales parecen estar bien, y de acuerdo a la realidad del experimento, debemos de tener en cuenta el número de repeticiones, dado que puede ser que se pierda cierta información o durante la ejecución de los algoritmos se haya añadido cierto ruido que desvió nuestros datos a, tal vez, una conclusión incorrecta.

Nuestras desviaciones estándar de manera general son muy pequeñas, lo que refleja unas diferencia entre los datos muy pequeña y, por tanto, una calidad de datos muy buena, es necesario hacer un análisis estadístico profundo para ver la realidad de los datos recolectados.

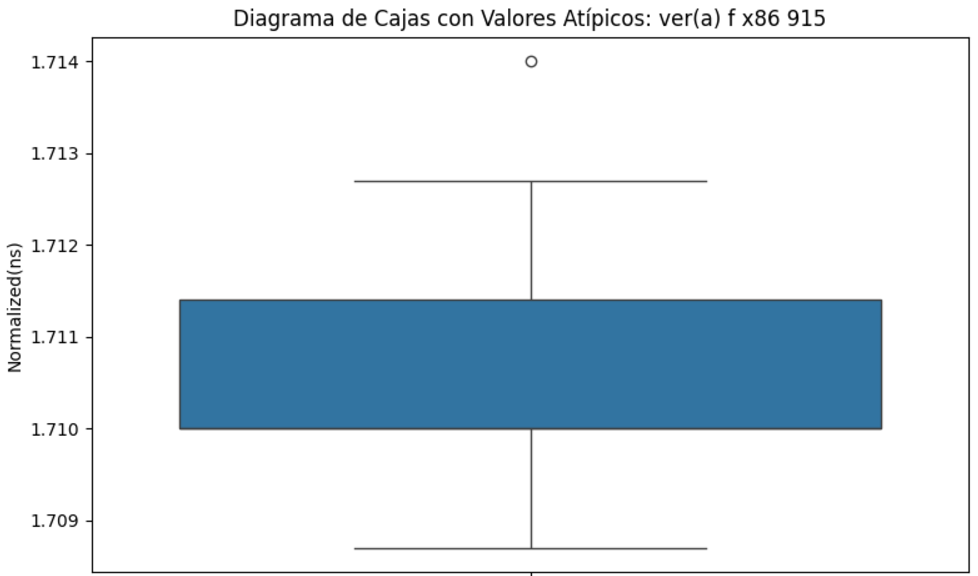
Primero que nada es necesario revisar posibles datos atípicos, para ello, se desarrolló un [código en python](https://colab.research.google.com/drive/1PzltJOCxLobjxXuZEbCspA2bAHN9XvB0#scrollTo=MSQ_czPWTCcq) que muestra un diagrama de cajas con aquellos datos atípicos que cumplan con alguna de las siguientes condiciones:

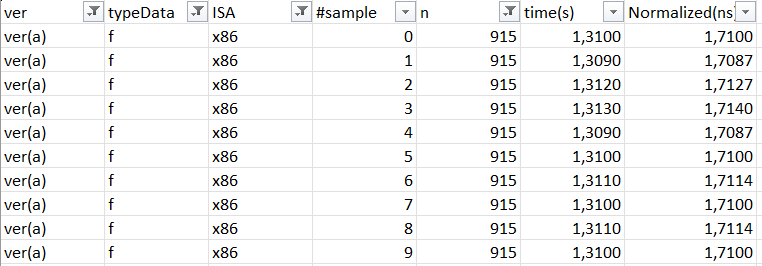
* **Por encima del límite superior:** Si el máximo valor atípico redondeado menos el límite superior (Q3) es mayor a 1.

O

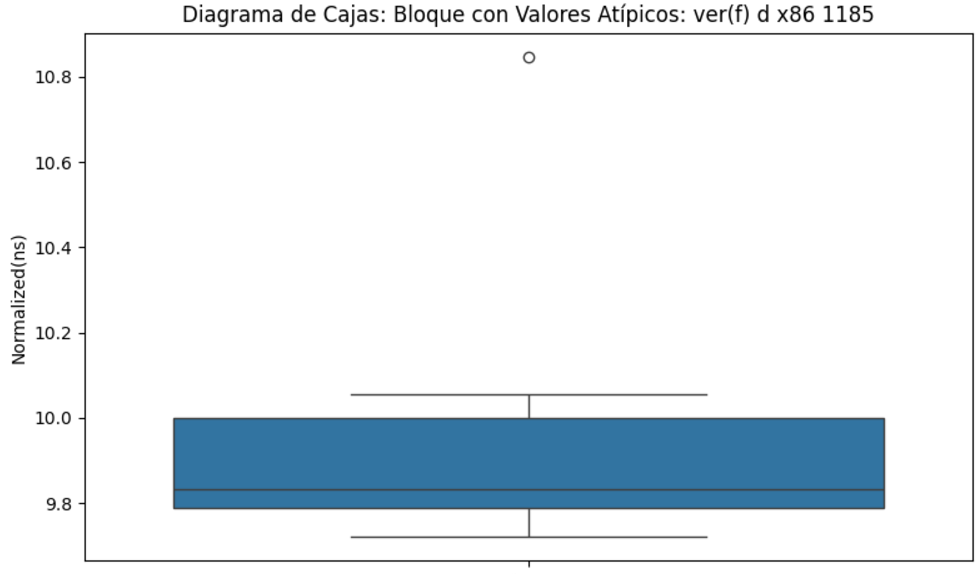
* **Por debajo del límite inferior:** Si el límite inferior (Q1) menos el mínimo valor atípico redondeado es mayor a 1.

Estas condiciones fueron decididas puesto que a veces se clasificaban como datos atípicos valores que realmente variaban demasiado poco, por ejemplo:





Con estas condiciones el único valor atípico que se detectó fue en los datos de la Laptop:



Esto nos muestra que nuestra primera toma de datos fue muy precisa con apenas factores secundarios, así como demuestra que la capacidad del PC de Escritorio y el Laptop son muy buenas.

Ahora, dicho valor se reemplaza por la mediana del bloque, puesto que es la medida de dispersión que no se ve afectada por los valores atípicos.

\*Se define como **bloque** a un conjunto de factores, por ejemplo: ver(a) d x64 1185, y dado que cada combinación de factores se corre una cantidad n de veces entonces un bloque puede tener tamaño n.

Una vez detectados y corregidos estos valores ahora se debe de analizar si el número de corridas es suficiente o es necesario aumentarse. Para ello se usarán las siguiente fórmulas:

**Coeficiente de Variación (CV) (Detección temprana de repeticiones insuficientes):**

El CV se calculará por bloque, y será una medida inmediata de detección de número de repeticiones insuficientes. Esto nos da una medida relativa de la dispersión con respecto a la media. Si el CV es bajo, para nuestro caso va a ser menor al 6% para el PC de Escritorio, y menor al 4% para la Laptop, entonces los datos son consistentes y no es necesario un aumento en el número de corridas. Si es mayor a 6%, para el PC de Escritorio, o mayor al 4%, para la Laptop, entonces sugiere que los datos de las muestras muestran cierta variación relativa que debe de tenerse en cuenta, entonces un mayor número de corridas podría llegar a capturar el comportamiento real del bloque y minimizar la cantidad de ruido.

Cabe resaltar que para algunos bloques el CV podría ser menor al 6% o al 4% pero aún así necesitar un aumento en el número de repeticiones, por tal motivo solo se usará como una medida previa a la revisión manual.

**Fórmula para determinar el número de muestras necesarias:**

* Z: Valor de la distribución normal estándar asociado con el nivel de confianza deseado (96% de confianza entonces, 1,96)
* : Desviación Estándar del tiempo normalizado.
* E: Margen de error permitido, es decir, la precisión deseada. (0,05)

Esta fórmula se realizará por bloque, y decidirá si el número de repeticiones es insuficiente y necesita ser aumentado, entonces, si se aumentará el número de corridas del bloque aproximando siempre al siguiente entero más cercano, en caso contrario no se cambiará. Si para un mismo hay diferentes aumentos en el número de repeticiones se elegirá el promedio de las repeticiones, esto con el objetivo de ahorrar recursos y tiempo.

(Revisar excel “[Resultados por Bloque](https://github.com/Electromayonaise/Rendimiento-y-Localidad-de-Cache/blob/main/doc/Results/Resultados%20por%20Bloque.xlsx)”, para más detalles)

Entonces, las nuevas repeticiones quedarían:

matrix\_sizes = [170, 440, 565, 670, 780, 915, 1185, 1470, 1700, 2050]

samples = [19, 15, 15, 15, 10, 15, 18, 9, 7, 5]

6. EJECUCIÓN DEL EXPERIMENTO

6.1 **Procedimiento experimental**

Ahora se detalla el procedimiento seguido para llevar a cabo el experimento final. Es esencial describir el paso a paso de cómo se realizó el experimento, desde la planeación y preparación, los equipos que se usaron, y cómo se ejecutó el experimento. Esto es debido a que se debe enfatizar lo importante que es seguir un plan experimental de manera estricta que garantice una toma de datos confiable y con el menor ruido posible.

El experimento se logró siguiendo las siguientes etapas:

6.1.1 **Planeación y Preparación**

**Revisión y validación del diseño experimental:** Primero que nada, se revisó si los factores primarios y secundarios identificados al inicio de este documento si eran todos los que se deben de considerar, garantizando a su vez que todos los niveles y las combinaciones se incluyeran en la matriz de diseño para evitar sesgos o resultados que no coincidían con la realidad.

**Configuración del entorno de Ejecución:** Con la ayuda del administrador de tareas de los equipos de prueba, se lograron suprimir una gran cantidad de servicios y aplicaciones en segundo plano que pudiesen introducir algún tipo de factor secundario indeseado. Entre las subtareas y otros servicios suprimidos se incluyeron:

* Mantener los equipos en una misma posición y en un mismo lugar, evitando posibles efectos por cambios en las condiciones ambientales, como temperatura, iluminación o ruido, que pudiesen influir en el rendimiento del hardware o el software.
* Se configuró el perfil, para el caso de la Laptop, en modo de energía de Alto Rendimiento, además de que se mantuvo enchufada durante todo el experimento.
* Se desactivaron las actualizaciones automáticas del sistema operativo y del software para evitar interrupciones inesperadas durante la ejecución. Así como desconexión total del internet en los dos equipos de pruebas.
* Finalmente se realizó una limpieza previa de los registros del sistema, para reducir tiempos de acceso y mejorar la estabilidad de los entornos de ejecución.

6.1.2 **Ejecución del Experimento**

**Ejecución de los Algoritmos:** Los algoritmos se ejecutaron en dos equipos diferentes, una PC de Escritorio y una Laptop, con arquitectura, procesador, y memoria RAM distintas**.** Los detalles de estos equipos ya se describieron al inicio de este documento. Cada Algoritmo se ejecutó siguiendo el orden establecido por la matriz de diseño aleatorizada, además de que cada algoritmo fue probado con varias repeticiones para obtener unos resultados confiables y representativos.

**Almacenamiento de los resultados:** Toda la información arrojada por las ejecuciones de los algoritmos se guardó en archivos .txt, uno por cada equipo de pruebas, en los que se almaceno información como: Versión del algoritmo, tipo de dato, arquitectura, tamaño de la matriz, número de la repetición, tiempo de ejecución, y tiempo de ejecución normalizado. Dicha información se pasó posteriormente a Excel para ser organizada y facilitar el análisis con la ayuda de Colab. También se creó, a partir de los datos recolectados, un Excel que almacenará la información en formato de Bloques, facilitando posteriormente la verificación e identificación de valores atípicos.

6.1.3 **Control de Calidad para la información recolectada**

**Verificación de los resultados:** Después de cada proceso de recolección de los datos, se revisaron los resultados con el fin de identificar posibles inconsistencias o valores atípicos, con la ayuda de diagramas de cajas. Estos fueron analizados y se realizaron ajustes usando la mediana del bloque para asegurar la validez de la información.

**Control de Desvíos:** Si durante la ejecución de los algoritmos hubo algún tipo de interrupción inesperada se revisó el motivo de la interrupción, se soluciono, y posteriormente se volvió a correr desde cero todos los algoritmos, repitiendo el procedimiento experimental.

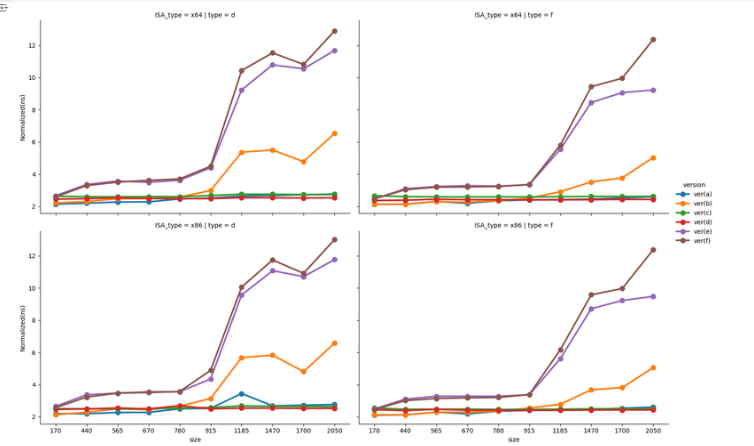
7. ANÁLISIS DE DATOS

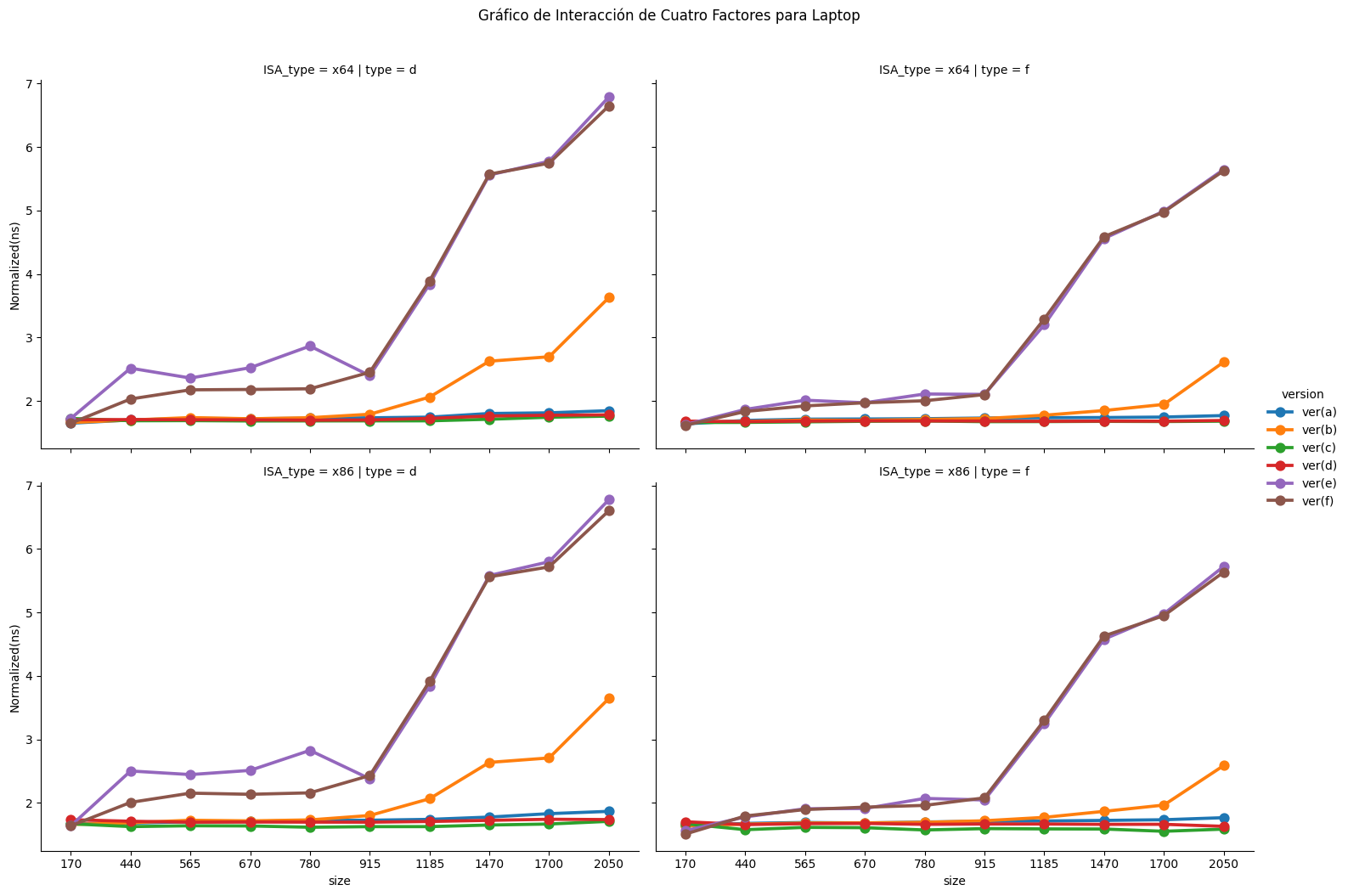
Ahora se analizarán los datos finales recolectados, esta sección constituye una etapa crucial en el desarrollo de este proyecto, ya que permite examinar en profundidad los factores que afectan el rendimiento de las distintas implementaciones del algoritmo de multiplicación de matrices. Se usarán herramientas estadísticas y representaciones visuales para identificar patrones, tendencias, y las relaciones que existen entre los factores.

7.1 **Registro y organización de los datos**

En esta sección se presentarán diferentes gráficos que proporcionarán una visión integral de las variables estudiadas, incluyendo su distribución y relaciones clave, este enfoque no solo facilitará la comprensión de los resultados, sino que también permitirá identificar patrones que puedan guiar futuras optimizaciones del sistema, además se hará un análisis detallado de los factores que influyen en el rendimiento, como la versión del algoritmo, el tamaño de la matriz, el tipo de dato y la ISA.

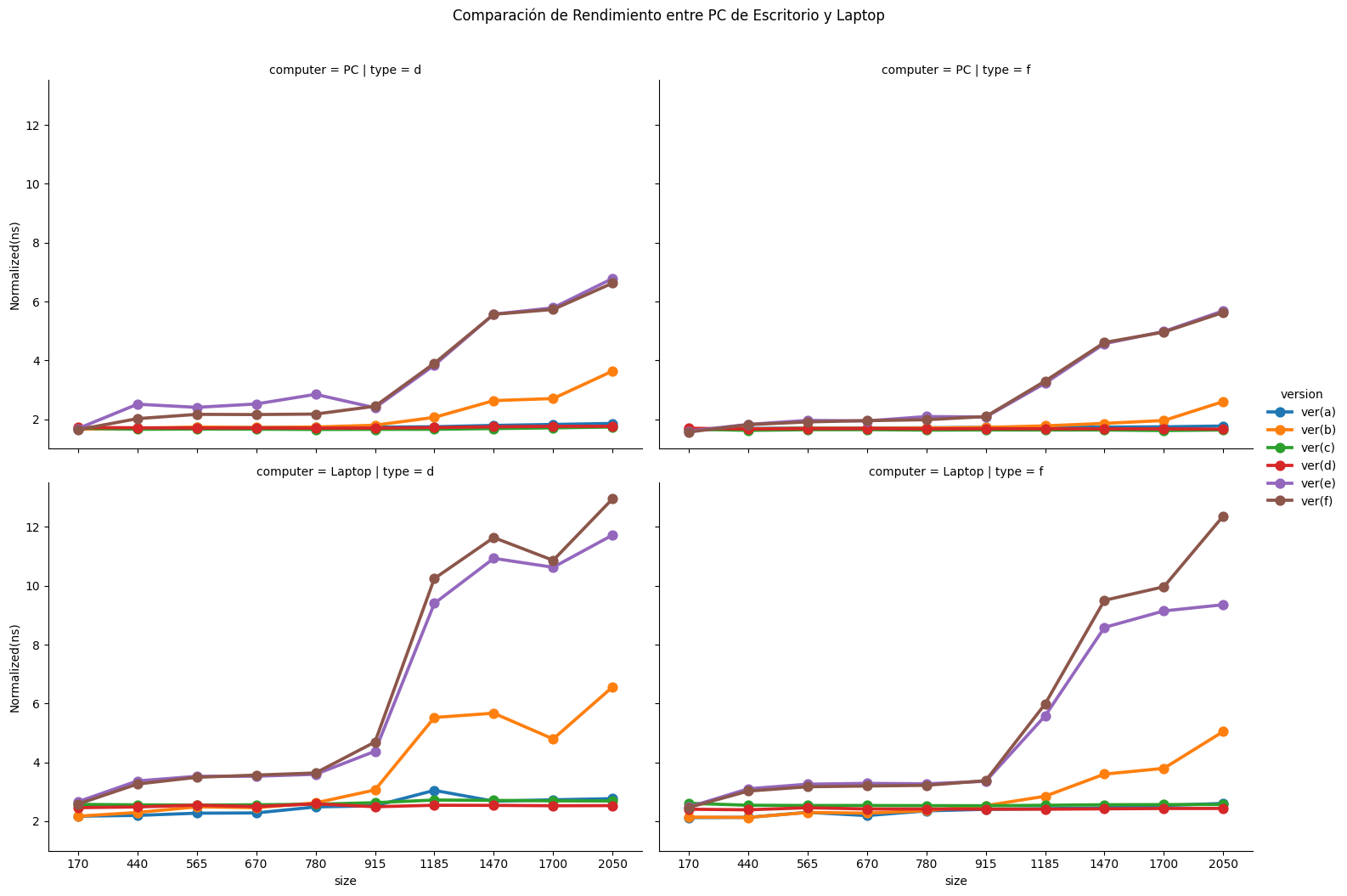
Gráficos de Interacción entre los 4 factores





Las interacciones entre los 4 factores demuestran que el rendimiento del algoritmo no depende únicamente de un solo factor, si no de la combinación de varios de ellos. Por ejemplo, ciertas versiones muestran una mayor sensibilidad al tamaño de la matriz de configuraciones con datos tipo **double** mientras que otras se mantienen más consistentes con **float**. Esto nos indica que se puede alcanzar una cierta combinación de factores o una configuración específica que pueda mejorar significativamente la eficiencia de los algoritmos.

Gráfico de Comparación de Rendimiento entre PC de Escritorio y Laptop



La PC de Escritorio supera consistentemente a la Laptop en términos de tiempos promedio de ejecución, especialmente en configuraciones de matrices grandes y con datos de tipo **double.** Esto refleja las ventajas de un hardware más robusto, con mayor capacidad de procesamiento, lo que a su vez demuestra cómo la Laptop tiene una menor estabilidad en su rendimiento.

Aunque ambos dispositivos son adecuados, al menos para esta experimentación, los resultados subrayan la importancia de seleccionar hardware más robusto para aplicaciones intensivas, ya que ofrece tiempos más consistentes y menor sensibilidad a configuraciones desfavorables.

7.2 **Análisis estadístico**

### Planteamiento de Hipótesis y Análisis Combinado de ANOVA (PC de Escritorio y Laptop)

Dado que buscamos evaluar si los factores analizados afectan el rendimiento (Normalized\_ns) considerando los resultados de [ANOVA](https://colab.research.google.com/drive/1bLRl5CWwlEYGY33I88kCzst-68QvxWev?usp=sharing) para PC de Escritorio y Laptop, presentamos las hipótesis globales:

#### Hipótesis Planteadas

1. **Hipótesis Nula (H₀):** Ninguno de los factores (version, size, type, ISA\_type) ni sus interacciones tienen un efecto significativo sobre el rendimiento (Normalized\_ns).
2. **Hipótesis Alternativa (H₁):** Al menos uno de los factores (version, size, type, ISA\_type) o sus interacciones afecta significativamente el rendimiento (Normalized\_ns).

#### Análisis Combinado de los Resultados

1. **Efecto de la Versión del Algoritmo (version):**
   * Tanto en PC de Escritorio como en Laptop, version tiene un efecto altamente significativo (p < 0.001) según los valores F elevados y los p-valores obtenidos.
   * En PC de Escritorio, las versiones ver(e) y ver(f) presentan los tiempos más altos (4.9704 ns y 5.2189 ns, respectivamente). En Laptop, estas mismas versiones también tienen los mayores tiempos promedio (2.8592 ns y 2.7657 ns, respectivamente). Esto indica que estas versiones son menos eficientes en ambas plataformas.
2. **Efecto del Tamaño de la Matriz (size):**
   * En ambos dispositivos, size tiene un efecto significativo sobre el rendimiento (p < 0.001). El tiempo de ejecución aumenta con matrices más grandes, como era de esperarse.
   * En PC de Escritorio, el tiempo promedio crece desde 2.4023 ns (170) hasta 6.1337 ns (2050). En Laptop, este incremento es similar, desde 1.6560 ns (170) hasta 3.4479 ns (2050). Esto refleja que el costo computacional es proporcional al tamaño de la matriz en ambas máquinas.
3. **Efecto del Tipo de Dato (type):**
   * En ambos sistemas, type tiene un efecto significativo sobre el rendimiento (p < 0.001 en PC de Escritorio y Laptop). Los cálculos con float (type = f) son más rápidos que con double (type = d).
   * Por ejemplo, en Laptop, el promedio para float es 1.9896 ns, mientras que para double es 2.1941 ns. En PC de Escritorio, float tiene un promedio de 3.1611 ns frente a 3.7262 ns para double.
4. **Efecto de la Arquitectura de Instrucción (ISA\_type):**
   * Aunque el efecto de ISA\_type es significativo en ambos sistemas, su impacto es relativamente menor comparado con otros factores.
   * En Laptop, las diferencias entre x64 (2.1030 ns) y x86 (2.0807 ns) son pequeñas. Similarmente, en PC de Escritorio, las diferencias entre x64 (3.4376 ns) y x86 (3.4497 ns) también son mínimas.
5. **Interacciones entre Factores:**
   * En ambos dispositivos, las interacciones entre version, size, y type son significativas (p < 0.001). Esto sugiere que el efecto de un factor depende de los valores de los otros.
   * Las interacciones de mayor orden, como version:size:type:ISA\_type, también son significativas, aunque con valores F más bajos. Esto indica que existen combinaciones óptimas de factores que pueden mejorar el rendimiento.

#### Decisión sobre las Hipótesis

**Rechazo de H₀:** Dado que los factores principales (version, size, type, ISA\_type) y varias de sus interacciones tienen efectos significativos (p < 0.001) sobre el rendimiento en ambos dispositivos, rechazamos la hipótesis nula.

Los resultados muestran que los factores analizados afectan significativamente el rendimiento en términos de tiempo de ejecución (Normalized\_ns) tanto en PC de Escritorio como en Laptop. Las interacciones sugieren la necesidad de optimizar combinaciones específicas para mejorar el desempeño en ambas plataformas.

8. JUICIO SOBRE LOS RESULTADOS

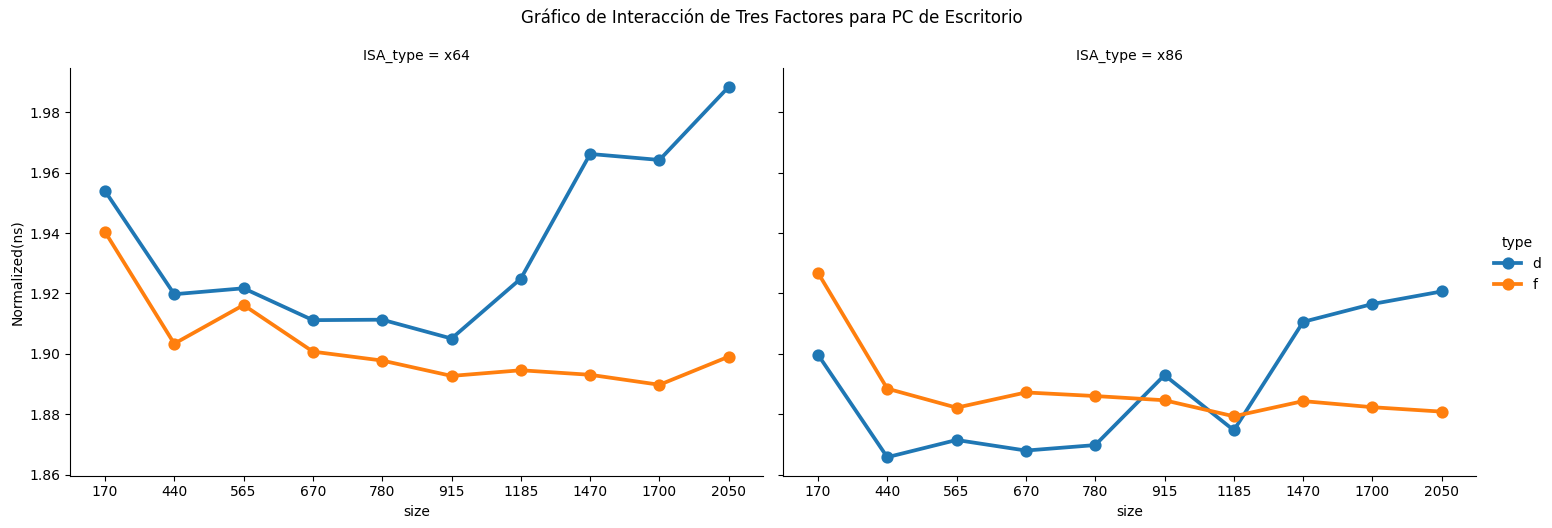
Los factores principales (version, size, type, ISA\_type) muestran un impacto significativo en el rendimiento, especialmente en las interacciones size:type y version:size:type. Esto confirma que optimizar un solo factor no es suficiente; el rendimiento depende de la combinación específica de todos ellos. Aunque las arquitecturas x86 y x64 tienen efectos significativos, su impacto relativo es menor. Esto sugiere que la elección de la ISA debe considerarse solo en escenarios específicos con restricciones de hardware.

Ahora, en cuanto a las limitaciones del experimento se identificó lo siguiente:

* **Ruido residual:** Aunque se controlaron muchos factores secundarios, la presencia de variaciones menores en los tiempos indica que aún podría haber ruido ambiental o del sistema, especialmente en la Laptop.
* **Representatividad de los tamaños de matrices:** Si bien se incluyeron tamaños representativos, la distribución de estos podría ampliarse para cubrir valores intermedios y extremos.
* **Uso de hardware:** La diferencia en el rendimiento entre la PC de Escritorio y la Laptop evidencia cómo las limitaciones de hardware pueden influir significativamente en los resultados. Esto subraya la necesidad de incluir más plataformas para una evaluación más robusta.

**Optimización de las versiones más ineficientes**

Las versiónes ver(d) y ver(f) mostraron el peor desempeño en ambas plataformas. Se pensó implementar una optimización basada en la multiplicación por bloques para mejorar la localidad de caché. Una posible estrategia era dividir matrices grandes en submatrices que puedan procesarse dentro de la caché. Esto es lo que se realizó y se obtuvieron los siguientes resultados



Podemos observar en la gráfica y posterior al [ANOVA](https://colab.research.google.com/drive/1bLRl5CWwlEYGY33I88kCzst-68QvxWev?usp=sharing) de 3 vías que para el caso del PC de escritorio:

Promedios por Tipo de Dato (type):

* Los tiempos promedio de ejecución para float (1.8992 ns) y double (1.9047 ns) son ahora casi indistinguibles, con una diferencia mínima de 0.0055 ns. Esto indica que la optimización mejoró de manera significativa la gestión de memoria caché, reduciendo el impacto del tipo de dato.

Promedios por Tamaño de Matriz (size):

* La mejora en los tiempos de ejecución es evidente en todos los tamaños de matrices, especialmente en matrices medianas y grandes (565 a 1185).
* Por ejemplo, para matrices de tamaño 565, el tiempo promedio es de 1.8979 ns, lo que representa una mejora respecto a los resultados previos donde ver(e) y ver(f) registraban tiempos cercanos a 2.6 ns.
* La varianza más alta se presenta en matrices pequeñas (170), lo cual puede deberse a una sobreoptimización para configuraciones de caché que son menos críticas en este caso.

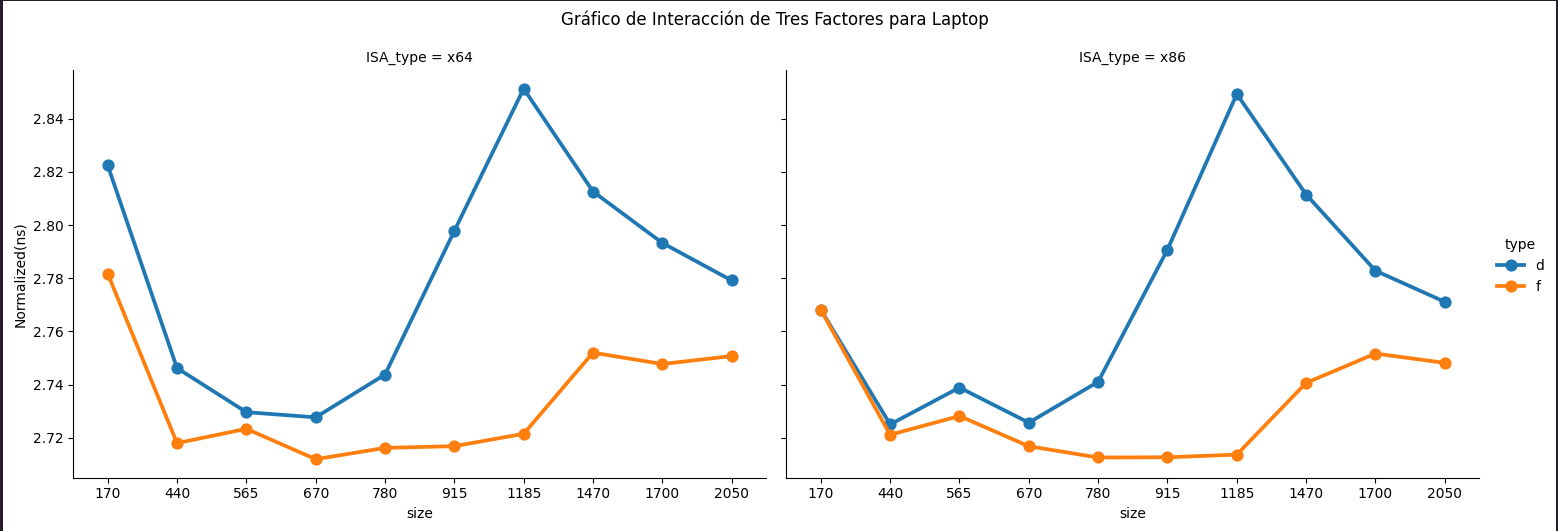
Promedios por Tipo de ISA (ISA\_type):

* La arquitectura x64 sigue mostrando un mejor rendimiento promedio (1.9175 ns) que x86 (1.8864 ns). Sin embargo, la diferencia es pequeña, lo que refleja que la optimización se aplica de manera efectiva a ambas arquitecturas.

Resultados del ANOVA:

* Tamaño de matriz (size): Tiene un efecto significativo (p = 2.7e-06), lo que confirma que la optimización es sensible al tamaño de las matrices.
* Interacciones size:type: También significativas (p = 0.021), sugiriendo que las configuraciones específicas de tamaño y tipo de dato influyen en el rendimiento.
* Interacciones de tres factores (size:type:ISA\_type): No significativas, indicando que el impacto de la ISA es menor en este contexto optimizado.

Podemos observar en la gráfica y posterior al ANOVA de 3 vías que para el caso de la laptop:



Promedios por Tipo de Dato (type):

* La optimización también redujo las diferencias entre float (2.7303 ns) y double (2.7648 ns), con una diferencia de 0.0345 ns. Esto es una mejora notable en comparación con los tiempos iniciales donde las diferencias eran más marcadas.

Promedios por Tamaño de Matriz (size):

* Se observa una mejora significativa en matrices medianas (440 a 1185), donde el tiempo promedio disminuyó considerablemente en comparación con los resultados originales.
* Para matrices grandes (1700 y 2050), el tiempo promedio sigue siendo mayor, pero la desviación estándar se redujo, mostrando una mayor consistencia en los resultados.

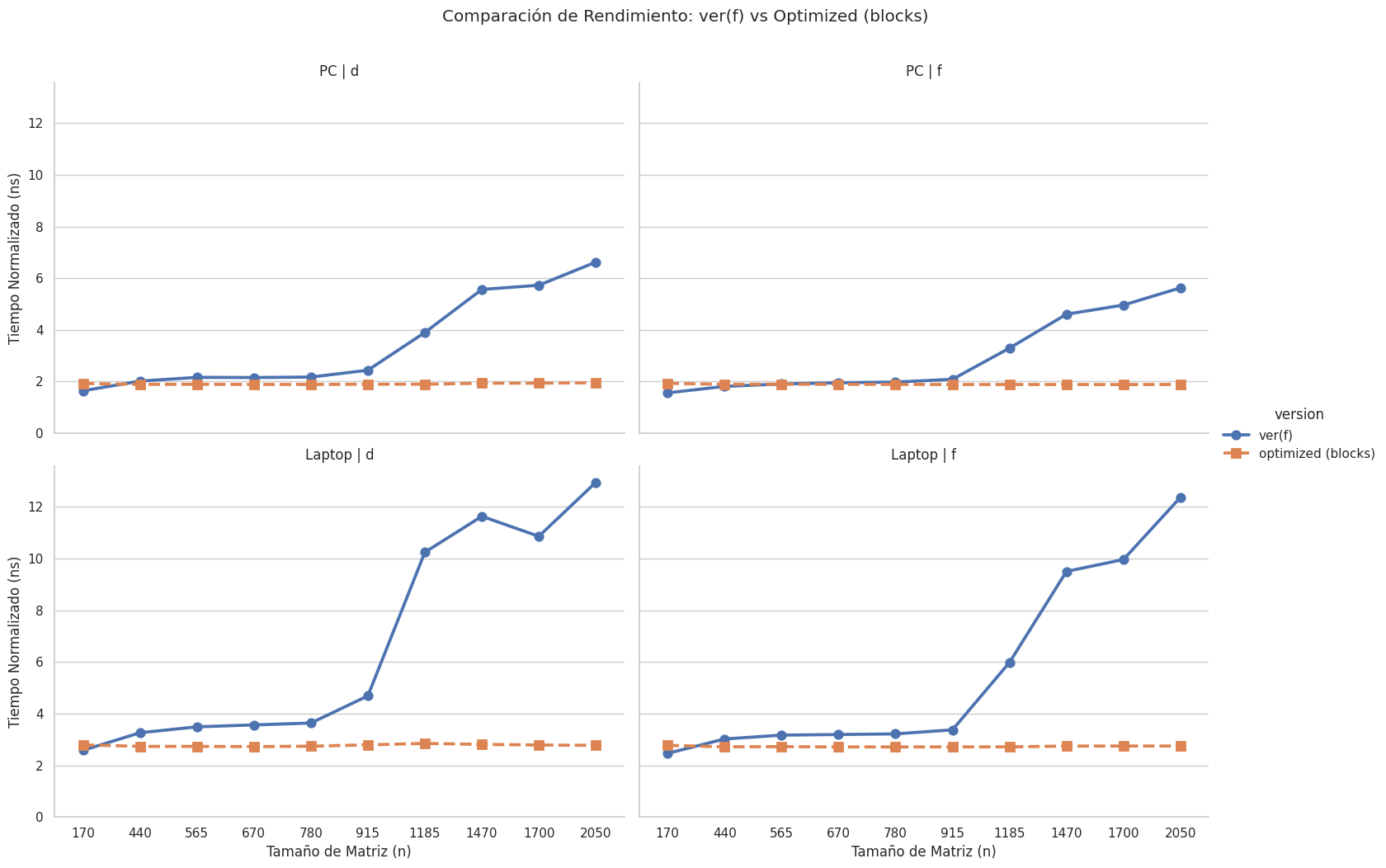
Promedios por Tipo de ISA (ISA\_type):

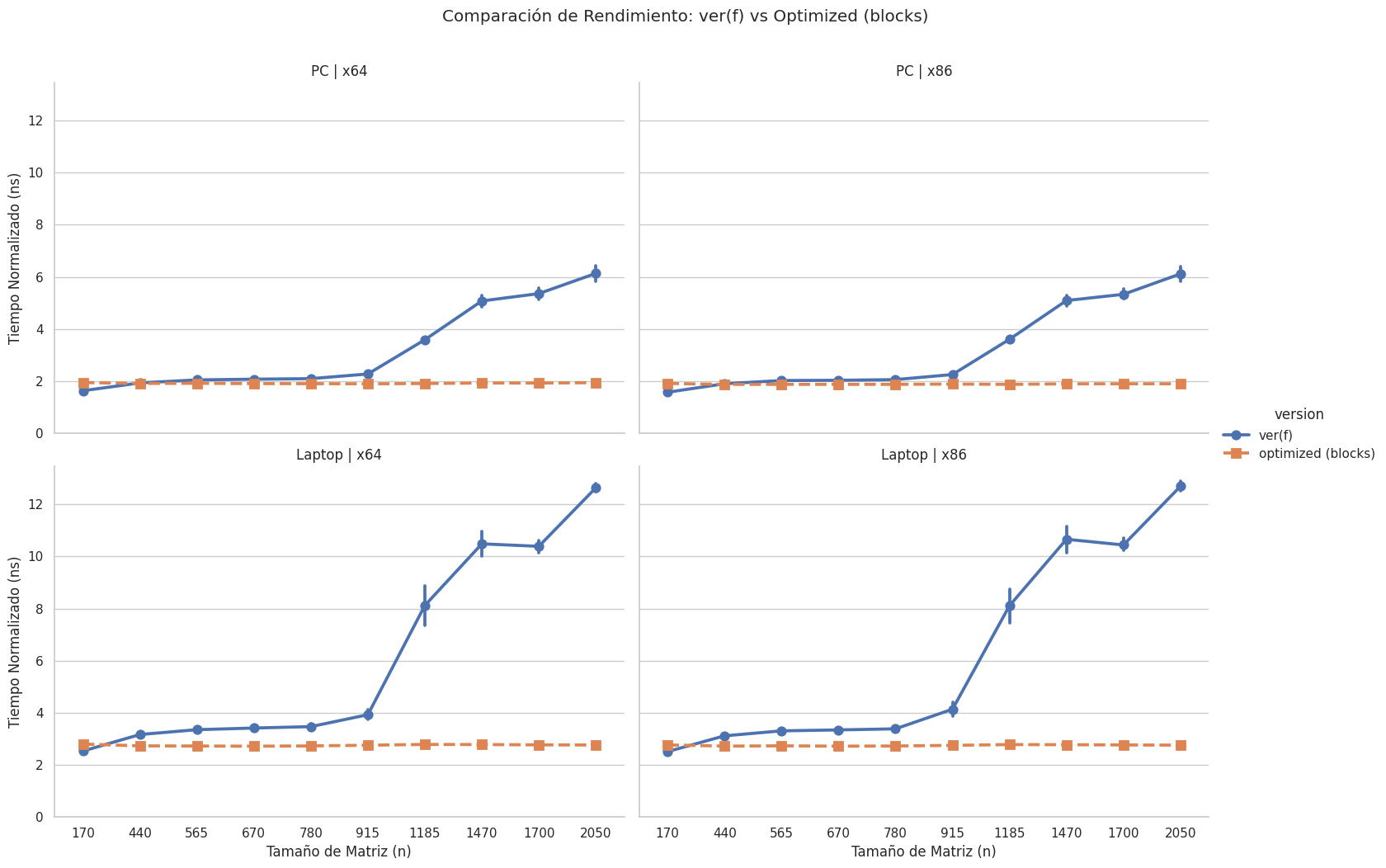
* Similar a la PC, x64 (2.7511 ns) muestra un rendimiento apenas superior al de x86 (2.7440 ns). Esto demuestra que la optimización no depende fuertemente de la arquitectura ISA.

Resultados del ANOVA:

* Tamaño de matriz (size): Efecto altamente significativo (p = 1.55e-15), validando que el tamaño de la matriz sigue siendo un factor determinante.
* Tipo de dato (type): Muy significativo (p = 3.23e-11), aunque menos pronunciado que en configuraciones sin optimización.
* Interacciones de mayor orden (size:type:ISA\_type): No significativas, lo que indica que las optimizaciones lograron neutralizar estas interacciones en gran medida.

Ahora, en cuanto a los resultados en comparación con el algoritmo f original se obtuvo lo siguiente:





Podemos ver que la implementación de la multiplicación por bloques mejoró significativamente el rendimiento de la versión ver(f) previamente ineficiente, especialmente para tamaños de matrices medianas y grandes. Además, las diferencias entre tipos de datos (float vs. double) se redujeron notablemente, lo que refleja un uso más eficiente de la jerarquía de memoria.

Podemos también resaltar que la PC de Escritorio sigue mostrando tiempos promedio más bajos que la Laptop, lo que destaca las ventajas de un hardware más robusto. La Laptop, aunque más limitada, también se benefició de las optimizaciones, mostrando menores variaciones en los tiempos de ejecución.

En esta sección terminamos por proponer dos escenarios de mejora en los que se puede profundizar en estudios posteriores:

* Realizar un análisis más profundo en matrices pequeñas para entender por qué presentan mayor variabilidad tras la optimización.
* Explorar técnicas adicionales, como prefetching de datos, para reducir aún más los tiempos en matrices grandes.

9. CONCLUSIONES

**9.1 Conclusión final y recomendaciones**  
A partir del análisis de los resultados obtenidos en los experimentos de multiplicación de matrices, es posible concluir que el rendimiento de los algoritmos varió de manera significativa dependiendo de diversos factores, como el tamaño de la matriz, el tipo de dato empleado y la arquitectura del conjunto de instrucciones (ISA). Sin embargo, lo más relevante de los resultados es que algunos algoritmos demostraron aprovechar mejor la localidad espacial y temporal que otros, lo que a su vez tuvo un impacto directo en el rendimiento general.

En primer lugar, se observó que el rendimiento de los algoritmos estuvo muy influenciado por la capacidad de las cachés de la CPU. La localidad espacial y temporal, en términos generales, son principios fundamentales que permiten a los algoritmos de multiplicación de matrices beneficiarse de la estructura jerárquica de las cachés. Los algoritmos que pudieron organizar las iteraciones de tal forma que favorecieran el uso secuencial de los datos fueron los que demostraron una mayor eficiencia (como las versiones a, c y d). Esto es particularmente cierto cuando el tamaño de la matriz es suficientemente pequeño para que los datos puedan almacenarse en las cachés de nivel 1 o 2, lo que minimiza la latencia de acceso a memoria. En contraste, los algoritmos que no aprovecharon adecuadamente esta estructura de memoria (como las versiones a, e y f), ya sea por un acceso no secuencial o por operar sobre submatrices grandes, experimentaron un mayor costo en tiempo debido a las constantes operaciones de "fallo de caché", lo que resultó en accesos frecuentes a la memoria principal.

Es importante destacar que el tipo de dato empleado, ya sea float o double, también tuvo un impacto relevante en los resultados. El tipo double exigió mayor espacio de memoria y aumentó el volumen de datos que debían transferirse entre la memoria y las cachés, lo que incrementó los tiempos de acceso. Aunque los algoritmos que trabajaron con double no fueron necesariamente menos eficientes, el aumento de los requerimientos de memoria limitó la capacidad de aprovechar las cachés de manera tan efectiva como los algoritmos que usaron el tipo float.

En cuanto al impacto del tamaño de la matriz, los algoritmos que trabajaron con matrices más grandes no pudieron mantener los datos en las cachés de nivel 1, lo que llevó a un aumento en el número de accesos a la memoria principal. Sin embargo, la relación no fue completamente lineal. A medida que el tamaño de la matriz crecía, algunos algoritmos fueron capaces de reordenar sus cálculos para minimizar los accesos a memoria, lo que permitió que siguieran aprovechando parcialmente la localidad espacial y temporal. Por otro lado, algoritmos menos eficientes en este aspecto sufrieron más por la incapacidad de reorganizar los cálculos, lo que resultó en una degradación del rendimiento.

Respecto a la arquitectura del ISA, el impacto en el rendimiento fue mucho menor en comparación con otros factores como el tamaño de las matrices y el tipo de dato empleado. Aunque las arquitecturas x64, con su mayor capacidad de registros y soporte para operaciones de mayor precisión, ofrecieron ventajas al manejar grandes volúmenes de datos, estas diferencias no fueron tan significativas como las limitaciones impuestas por la presión sobre las cachés o el uso de datos de tipo double. Del mismo modo, las limitaciones en los registros y el ancho de banda de memoria de las arquitecturas x86 generaron cuellos de botella en ciertas condiciones, pero este efecto resultó ser menos determinante para el rendimiento general que el diseño del algoritmo y su capacidad para aprovechar la localidad espacial y temporal.

A partir de este análisis, se puede sugerir que el algoritmo más optimizado debe ser aquel que no solo aproveche de manera eficiente la localidad espacial y temporal, sino que también tenga en cuenta las limitaciones de las cachés y el tamaño de los datos. Un diseño adecuado de algoritmo debería incluir una estrategia de agrupamiento de matrices que minimice los accesos a memoria y maximice la reutilización de datos en las cachés. Además, para aprovechar al máximo las capacidades de la arquitectura, el algoritmo debe ser capaz de adaptarse a diferentes tamaños de matrices y tipos de datos, con un enfoque en reducir la presión sobre las cachés.

Una recomendación importante para futuros experimentos es la evaluación más exhaustiva de técnicas como el bloqueo de matrices y el uso de algoritmos más avanzados, como la multiplicación de matrices de Strassen, que podría reducir el número de operaciones necesarias y mejorar el rendimiento en matrices grandes. Además, sería útil realizar experimentos adicionales con diferentes configuraciones de caché, como la variación en los niveles de caché o la implementación de cachés específicas para cada tipo de dato.

10. REFERENCIAS

Diseño de Experimentos (DoE). (n.d.). *Capítulo 1: Terminología y Conceptos*. [Documento PDF].<https://www.icesi.edu.co/moodle/pluginfile.php/1135690/mod_resource/content/1/Cap%C3%ADtulo%201%20DOE%20Terminolog%C3%ADa%20y%20Conceptos%20v1.3%20%281%29.pdf>

Parthasarathi, R. (s. f.). *Cache Optimizations i – Computer Architecture*. <https://www.cs.umd.edu/~meesh/411/CA-online/chapter/252/index.html>

Creel. (2017, 1 agosto). *Performance x64: Cache Blocking (Matrix Blocking)* [Vídeo]. YouTube. <https://www.youtube.com/watch?v=G92BCtfTwOE>

Franchitti, J. (2024, 13 noviembre). *5.5 Memory Hierarchy - Introduction to Computer Science | OpenStax*. <https://openstax.org/books/introduction-computer-science/pages/5-5-memory-hierarchy>