INFORME FINAL

DAVIDE FLAMINI CAZARAN - A00381665 - INGENIERÍA DE SISTEMAS
ÓSCAR ANDRÉS GÓMEZ LOZANO - A00394142 - INGENIERÍA DE SISTEMAS
BRIAN MATASCA DAGUA - A00378054 - INGENIERÍA DE SISTEMAS
ESTEBAN GAVIRIA ZAMBRANO - A00396019 - INGENIERÍA DE SISTEMAS

DOCENTE CARLOS ANDRADE DIAZ

UNIVERSIDAD ICESI
FACULTAD DE INGENIERÍA
ARQUITECTURA DE COMPUTADORES
5 DE JUNIO DE 2024

TABLA DE CONTENIDO

INTRODUCCIÓN

A lo largo del curso de Arquitectura de Computadores, se ha estudiado la estructura y funcionamiento de los componentes hardware y software que influyen en el rendimiento de los programas de aplicación. El objetivo final de este curso es comprender cómo estas estructuras afectan la eficiencia y utilidad de los programas, especialmente en términos de procesamiento, acceso y almacenamiento de datos. En este contexto, el análisis del rendimiento es crucial para evaluar las decisiones de diseño y optimización en sistemas de cómputo.

El problema que buscamos resolver en este trabajo es evaluar el impacto de la localidad de caché en el rendimiento de la multiplicación de matrices. Los procesadores modernos utilizan memorias caché para superar la brecha entre las velocidades del procesador y la memoria principal. Aprovechar el principio de localidad permite que la mayoría de las solicitudes de datos sean satisfechas por la memoria caché, mejorando significativamente el rendimiento. Sin embargo, la efectividad de esta optimización puede variar según el patrón de acceso a la memoria de diferentes algoritmos.

OBJETIVOS DEL EXPERIMENTO

1. **Cuantificar y analizar el grado de interacción entre seis versiones de algoritmos de multiplicación de matrices y el sistema de memoria.**

   - Medir el tiempo de ejecución y normalizar los datos para diferentes tamaños de matrices.

   - Comparar los patrones de acceso a memoria y su efecto en el rendimiento.

2. **Aplicar la metodología de diseño de experimentos para estructurar y llevar a cabo los experimentos.**

   - Identificar y controlar los factores primarios y secundarios que pueden afectar los resultados.

   - Minimizar la incertidumbre experimental y registrar los datos de manera adecuada.

3. **Evaluar la mejora en el rendimiento mediante la implementación de multiplicación por bloques.**

- Implementar y probar un esquema de multiplicación por bloques en el algoritmo con peor rendimiento.

- Analizar los datos recolectados y comparar con los resultados iniciales para determinar la efectividad de la optimización.

METODOLOGÍA

La metodología utilizada sigue los pasos del diseño de experimentos, incluyendo la identificación de factores de tratamiento y ruido, la selección del tipo de experimento, y la recolección y análisis de datos. En el experimento se ejecutaron seis versiones diferentes de algoritmos para la multiplicación de matrices en los computadores del laboratorio 501L de la Universidad Icesi. Los grupos de laboratorio involucrados emplearon los equipos LHW05, a cargo de Esteban y Brian, y LHW12, a cargo de Davide y Oscar. En los seis algoritmos se empleó el mismo lenguaje de programación c++ (C plus plus) y mismos equipos de cómputo para todas las pruebas, garantizando la coherencia en los resultados.

En este contexto, el enfoque del trabajo es documentar los resultados obtenidos por cada grupo y realizar un análisis comparativo tanto entre los grupos como de manera individual. Se busca analizar cómo la localidad de caché influye en el rendimiento de los algoritmos de multiplicación de matrices y proponer mejoras basadas en técnicas de optimización de memoria. Este análisis comparativo permitirá emitir un juicio informado sobre las mejores prácticas para maximizar el rendimiento en sistemas de cómputo.

MARCO TEÓRICO

Este trabajo se sustenta en dos áreas principales de la arquitectura de computadores: el diseño de experimentos (DoE) y la teoría relevante para explicar los experimentos realizados.

El Diseño de Experimentos (DoE) es una metodología esencial para planificar, ejecutar y analizar experimentos de manera sistemática, permitiendo identificar las relaciones causa-efecto y optimizar variables de diseño. Los productos y procesos diseñados con DoE tienden a presentar un mejor rendimiento, mayor confiabilidad y menores costos generales.

1. Variables de Diseño:

Son las variables que se manipulan o controlan durante un experimento para estudiar su efecto en la respuesta. Pueden ser variables continuas (como la temperatura o el tiempo) o categóricas (como el tipo de material o el método de procesamiento).

2. Respuesta:

Es la variable que se mide para evaluar el efecto de las variables de diseño. Por ejemplo, en un experimento de rendimiento de un sistema informático, la respuesta podría ser el tiempo de ejecución o la utilización de la CPU.

3. Factores:

Son las variables que el investigador manipula de manera sistemática para observar y medir su efecto en la variable de respuesta. Los factores pueden ser primarios (variables independientes cuyo efecto sobre la variable de respuesta se quiere estudiar) o secundarios (que se controlan o se bloquean para evitar que confundan los efectos de los factores primarios).

4. Réplicas:

Son las repeticiones del experimento bajo las mismas condiciones para verificar la reproducibilidad de los resultados y estimar la variabilidad experimental.

5. Bloqueo:

Es una técnica para controlar la variabilidad en el experimento al agrupar unidades experimentales similares en bloques y realizar las pruebas dentro de cada bloque.

6. Aleatorización:

Es el proceso de asignar aleatoriamente unidades experimentales a diferentes tratamientos o condiciones para reducir el sesgo y controlar el efecto de variables no controladas.

7. Análisis de Varianza (ANOVA):

Es una técnica estadística utilizada para analizar si existen diferencias significativas entre los tratamientos o condiciones en un experimento. Permite determinar qué factores tienen un efecto significativo en la respuesta.

8. Interacción:

Es el efecto conjunto de dos o más variables de diseño en la respuesta. La presencia de interacciones puede modificar el efecto individual de cada variable en la respuesta.

9. Diseño Factorial:

Es un tipo de diseño experimental en el que se estudian múltiples factores simultáneamente y se analiza su efecto principal y las interacciones entre ellos.

Estos conceptos son fundamentales para planificar, ejecutar y analizar experimentos de manera sistemática y obtener conclusiones válidas y significativas sobre el impacto de las variables de diseño en la respuesta del sistema o proceso estudiado.

En cuanto a la teoría relevante para este estudio, destacan varios conceptos clave:

Principios de Localidad:

- El principio de localidad se refiere a la tendencia de los programas informáticos a acceder a un conjunto relativamente pequeño de datos en un corto período de tiempo.
- Los principios de localidad espacial y temporal indican que los datos a los que se accede recientemente (temporal) o en proximidad física (espacial) tienen una alta probabilidad de ser accedidos nuevamente en el futuro cercano.
- Estos principios son fundamentales para comprender cómo se aprovecha la memoria caché y cómo afecta al rendimiento de los algoritmos, especialmente en operaciones como la multiplicación de matrices.

Jerarquía de Memoria:

- La jerarquía de memoria en un sistema informático describe la organización de diferentes niveles de memoria, desde la memoria caché hasta la memoria principal y el almacenamiento secundario.
- Los niveles más cercanos al procesador, como la memoria caché, tienen tiempos de acceso más rápidos, pero capacidades limitadas, mientras que los niveles más alejados tienen tiempos de acceso más lentos pero mayores capacidades de almacenamiento.
- La eficacia de la jerarquía de memoria está influenciada por el principio de localidad y afecta directamente al tiempo de ejecución de los algoritmos.

Static RAM

La RAM estática (SRAM) es más rápida y significativamente más cara que la RAM dinámica (DRAM). La SRAM se utiliza para las memorias caché, tanto dentro como fuera del chip de la CPU. La SRAM almacena cada bit en una celda de memoria biestable. Cada celda se implementa con un circuito de seis transistores. Este circuito tiene la propiedad de que puede permanecer indefinidamente en cualquiera de dos configuraciones de voltaje diferentes, o estados. Cualquier otro estado será inestable; partiendo de allí, el circuito se moverá rápidamente hacia uno de los estados estables.

Dynamic RAM

La DRAM se utiliza para la memoria principal y el búfer de cuadro de un sistema gráfico. La DRAM almacena cada bit como una carga en un condensador. Este condensador es muy pequeño, típicamente alrededor de 30 femtofaradios. El almacenamiento de DRAM puede ser muy denso, ya que cada celda consta de un condensador y un solo transistor de acceso. Sin embargo, a diferencia de la SRAM, una celda de memoria DRAM es muy sensible a cualquier perturbación. Cuando el voltaje del condensador se ve perturbado, nunca se recuperará.

Memorias caché

Las jerarquías de memoria de los primeros sistemas informáticos tenían sólo tres niveles: registros de la CPU, memoria principal DRAM y almacenamiento en disco. Sin embargo, debido al aumento de la brecha entre la CPU y la memoria principal, los diseñadores de sistemas se vieron obligados a insertar una pequeña memoria caché SRAM, llamada caché L1 (caché de nivel 1) entre los registros de la CPU y la memoria principal. La caché L1 se puede acceder casi tan rápido como los registros, típicamente de 2 a 4 ciclos de reloj. A medida que la brecha de rendimiento entre la CPU y la memoria principal continuó aumentando, los diseñadores de sistemas respondieron insertando una caché adicional más grande, llamada caché L2, entre la caché L1 y la memoria principal, que se puede acceder en aproximadamente 10 ciclos de reloj. Algunos sistemas modernos incluyen una caché adicional aún más grande, llamada caché L3, que se sitúa entre la caché L2 y la memoria principal en la jerarquía de memoria y se puede acceder en 30 o 40 ciclos.

Organización de la caché

Una caché es un arreglo de conjuntos. Cada conjunto contiene una o más líneas. Cada línea contiene un valid bit, algunos tag bits y un bloque de datos. La organización de la caché consiste en una partición de los m bits de dirección en t tag bits, s set index bits y b block offset bits.

Impacto del tamaño de la caché

Por un lado, una caché más grande tiende a aumentar la tasa de aciertos. Por otro lado, siempre es más difícil hacer que las memorias grandes funcionen más rápido. Como resultado, las cachés más grandes tienden a aumentar el tiempo de acierto. Esto es especialmente importante para las cachés L1 en el chip, que deben tener un tiempo de acierto corto

Impacto del tamaño de los bloques de la caché

Los bloques grandes tienen ventajas y desventajas. Por un lado, los bloques más grandes pueden aumentar la tasa de aciertos al aprovechar la localidad espacial que pueda existir en un programa. Sin embargo, para un tamaño de caché dado, los bloques más grandes implican un menor número de líneas de caché, lo que puede afectar negativamente la tasa de aciertos en programas con más localidad temporal que espacial. Además, los bloques más grandes tienen un impacto negativo en la penalización por fallos, ya que causan tiempos de transferencia más largos. Los sistemas modernos suelen comprometerse con bloques de caché que contienen entre 32 y 64 bytes.

AMAT (Average Memory Access Time):

$$[AMAT = HitTime + MissRate \times MissPenalty]$$

El AMAT es una métrica que representa el tiempo promedio necesario para acceder a la memoria en un sistema jerárquico de memoria. Se calcula considerando los tiempos de acceso de cada nivel de memoria y la probabilidad de acceso a cada nivel, según los principios de localidad. El AMAT es una medida clave para evaluar el rendimiento de la jerarquía de memoria y su impacto en el rendimiento general del sistema.

Miss Rate (Tasa de Fallos en Caché):

$$[MissRate = \frac{Number\ of\ Cache\ Misses}{Total\ Memory\ Accesses}]$$

El Miss Rate, o tasa de fallos en caché, es una métrica que indica la frecuencia con la que el sistema no puede encontrar la información solicitada en la memoria caché y debe buscarla en niveles más lentos de la jerarquía de memoria. Se calcula dividiendo el número de fallos en caché (cuando la información no está en caché) entre el total de accesos a memoria.

Explicación de un Miss: Un miss ocurre cuando el procesador busca un dato en la caché, pero no lo encuentra porque no está almacenado en esa ubicación de memoria caché. Esto resulta en un acceso más lento a la información, ya que se necesita acceder a niveles más lentos de memoria (como la RAM o el disco duro) para obtener los datos necesarios.

Hit Rate (Tasa de Aciertos en Caché):

$$[HitRate = 1 - MissRate]$$

El hit rate, o tasa de aciertos en caché, es la probabilidad de que el sistema encuentre la información solicitada en la memoria caché y no tenga que buscarla en niveles más lentos de memoria. Se calcula como la complementaria del miss rate, es decir, restando el miss rate de 1.

Explicación de un Hit: Un hit ocurre cuando el procesador encuentra el dato buscado en la memoria caché, lo que resulta en un acceso rápido y eficiente a la información sin necesidad de acceder a niveles más lentos de la jerarquía de memoria.

Anova:

La ANOVA (Análisis de la Varianza) es una técnica estadística utilizada para analizar las diferencias significativas entre las medias de tres o más grupos. Es especialmente útil cuando se comparan los efectos de múltiples factores en una variable de interés. La fórmula básica de ANOVA es:

$$[F = \frac{MS_{Between}}{MS_{Within}}]$$

Donde:

- (F) es la estadística de prueba para la ANOVA.
- $(MS_{Between})$ es la media de los cuadrados de las diferencias entre los grupos.
- (MS_{Within}) es la media de los cuadrados de las diferencias dentro de los grupos.

¿Para qué se usa ANOVA?

- **Comparación de medias:** ANOVA permite determinar si hay diferencias significativas entre las medias de tres o más grupos.
- **Efecto de los factores:** Se utiliza para evaluar el efecto de uno o más factores (variables independientes) en una variable de interés (variable dependiente).
- **Identificación de diferencias:** ANOVA ayuda a identificar qué grupos tienen diferencias significativas entre sus medias y cuáles no.

Tabla ANOVA:

La tabla ANOVA muestra la descomposición de la variabilidad total en variabilidad entre grupos y variabilidad dentro de los grupos. Tiene varias columnas importantes, incluyendo:

1. **Fuente de Variación:** Indica los componentes de la variabilidad, como "Entre Grupos" y "Dentro de Grupos".
2. **Suma de Cuadrados (SS):** La suma de los cuadrados de las diferencias entre los datos y la media de cada grupo.
3. **Grados de Libertad (df):** Representa el número de valores independientes que pueden variar en una estadística sin que se afecte el valor de otra.
4. **Media de Cuadrados (MS):** Es la suma de cuadrados dividida por los grados de libertad correspondientes.
5. **F:** La estadística de prueba calculada como la división de la media de cuadrados entre grupos y la media de cuadrados dentro de grupos.

La interpretación de la tabla ANOVA se centra en comparar la varianza entre grupos (variabilidad debida a los efectos de los factores) con la varianza dentro de grupos (variabilidad

debida al azar o errores). Si la varianza entre grupos es significativamente mayor que la varianza dentro de grupos, se concluye que hay diferencias significativas entre al menos dos grupos en términos de la variable de interés.

Tiempo Normalizado:

El **tiempo normalizado** es una medida de rendimiento utilizada para comparar la eficiencia de diferentes algoritmos o implementaciones al ejecutar tareas que varían en tamaño o complejidad. La normalización permite ajustar el tiempo de ejecución para tener en cuenta el tamaño del problema o la cantidad de trabajo realizado, lo que facilita una comparación justa y significativa entre distintos escenarios. Para un problema de tamaño N que requiere $f(N)$ operaciones (donde $f(N)$ puede ser, por ejemplo, N^2 o N^3, el tiempo normalizado $T_{normalizado}$ se define como:

$$T_{normalizado} = \frac{T_{observado}}{f(N)}$$

donde:

- $T_{observado}$ es el tiempo de ejecución real medido.
- $f(N)$ es una función que describe la cantidad de operaciones en función del tamaño del problema N.

Estos conceptos, junto con la metodología de diseño de experimentos mencionada previamente, proporcionan el marco teórico necesario para comprender y analizar el impacto de la localidad de caché en el rendimiento de la multiplicación de matrices y para proponer mejoras basadas en técnicas de optimización de memoria. El análisis comparativo de los resultados, incluyendo los elementos del marco teórico, permitirán emitir un juicio informado sobre las mejores prácticas para maximizar el rendimiento en sistemas de cómputo.

Multiplicación de matrices por bloques:

Esta es una técnica utilizada para mejorar la eficiencia de la multiplicación de matrices, especialmente en términos de aprovechamiento de la memoria caché. En esta técnica, las matrices grandes se dividen en bloques más pequeños y se realiza la multiplicación de estos bloques, lo que permite un mejor uso de la localidad espacial y temporal de la caché. A continuación se describe en detalle en qué consiste esta estrategia:

1. División de Matrices en Bloques: Las matrices de entrada A y B se dividen en submatrices más pequeñas (bloques). Por ejemplo, si se tiene una matriz de tamaño $N \times N$ y se elige un tamaño de bloque de B, cada matriz se divide en $(N/B) \times (N/B)$ bloques de tamaño $B \times B$.
2. Multiplicación de Bloques: En lugar de multiplicar las matrices completas de una sola vez, se multiplican los bloques correspondientes. Para calcular un bloque de la matriz resultante C, se realiza la suma de los productos de los bloques correspondientes de A y B. Es decir:

$$C_{i,j} = \sum_{k} \quad A_{i,k} \times B_{k,j}$$

Donde $A_{i,k}$ y $B_{k,j}$ son bloques de tamaño $B \times B$.

3. Optimización del Uso de la Caché: Al trabajar con bloques más pequeños que caben completamente en la caché, se reduce el número de accesos a la memoria principal. La multiplicación de bloques permite que los datos necesarios se mantengan en la caché durante el cálculo de un bloque, lo que mejora significativamente el rendimiento debido a la reducción de los fallos de caché.

FACTORES PRIMARIOS

En el diseño experimental para evaluar el impacto de la localidad de caché en el rendimiento de la multiplicación de matrices, se han identificado tres factores primarios: el tipo de dato, el tipo de algoritmo y el tamaño de la matriz. A continuación, se describen estos factores junto con sus respectivos rangos y niveles.

Tipos de niveles variables

Tipo de Dato: Variable cualitativo.
Tipo de Algoritmo: Variable cualitativo.
Tamaño de la Matriz: Variable cuantitativo.

Tipo de Dato

El tipo de dato es un factor primario que puede influir significativamente en el rendimiento de la multiplicación de matrices debido a las diferencias en el tamaño y la precisión de los datos. En este experimento, se consideran dos niveles para el tipo de dato:

1. **Double**: Representa un dato de doble precisión (64 bits), proporcionando mayor exactitud en los cálculos pero también ocupando más espacio en memoria.
2. **Float**: Representa un dato de precisión simple (32 bits), ocupando menos espacio en memoria pero con menor exactitud en los cálculos.

Tipo de Algoritmo

El tipo de algoritmo es otro factor primario crítico en este experimento. Cada algoritmo puede tener un patrón de acceso a la memoria diferente, lo cual afecta su eficiencia en términos de aprovechamiento de la localidad de caché. Los algoritmos son:

1. Algoritmo 1:

(a) Version *ijk*

code/mem/matmult/mm.c

```
1   for (i = 0; i < n; i++)
2       for (j = 0; j < n; j++) {
3           sum = 0.0;
4           for (k = 0; k < n; k++)
5               sum += A[i][k]*B[k][j];
6           C[i][j] += sum;
7       }
```

code/mem/matmult/mm.c

2. Algoritmo 2:

(f) Version *ikj*

code/mem/matmult/mm.c

```
1   for (i = 0; i < n; i++)
2       for (k = 0; k < n; k++) {
3           r = A[i][k];
4           for (j = 0; j < n; j++)
5               C[i][j] += r*B[k][j];
6       }
```

code/mem/matmult/mm.c

3. Algoritmo 3:

(b) Version *jik*

code/mem/matmult/mm.c

```
1   for (j = 0; j < n; j++)
2       for (i = 0; i < n; i++) {
3           sum = 0.0;
4           for (k = 0; k < n; k++)
5               sum += A[i][k]*B[k][j];
6           C[i][j] += sum;
7       }
```

code/mem/matmult/mm.c

4. Algoritmo 4:

(c) Version *jki*

code/mem/matmult/mm.c

```
1   for (j = 0; j < n; j++)
2       for (k = 0; k < n; k++) {
3           r = B[k][j];
4           for (i = 0; i < n; i++)
5               C[i][j] += A[i][k]*r;
6       }
```

code/mem/matmult/mm.c

5. Algoritmo 5:

(e) Version *kij*

code/mem/matmult/mm.c

```
1   for (k = 0; k < n; k++)
2       for (i = 0; i < n; i++) {
3           r = A[i][k];
4           for (j = 0; j < n; j++)
5               C[i][j] += r*B[k][j];
6       }
```

code/mem/matmult/mm.c

6. Algoritmo 6:

(d) Version *kji*

code/mem/matmult/mm.c

```
1   for (k = 0; k < n; k++)
2       for (j = 0; j < n; j++) {
3           r = B[k][j];
4           for (i = 0; i < n; i++)
5               C[i][j] += A[i][k]*r;
6       }
```

code/mem/matmult/mm.c

Tamaño de la Matriz

El tamaño de la matriz es un factor primario que afecta directamente el tiempo de ejecución y la utilización de la memoria caché. Diferentes tamaños de matrices pueden influir en la eficacia del caché debido a la variación en la cantidad de datos procesados simultáneamente. En este experimento, se consideran los siguientes tamaños de matrices, ordenados de menor a mayor y sin repeticiones:

- 350
- 450
- 550
- 750
- 1000
- 1250
- 1500

Estos tamaños permiten evaluar el rendimiento de los algoritmos bajo diferentes cargas de trabajo, proporcionando un panorama completo de su comportamiento en relación con la memoria caché y el sistema de memoria en general.

Adicionalmente, con la incorporación de un algoritmo para la multiplicación de matrices por bloques que utiliza el recorrido del algoritmo con peor rendimiento y su comparación con este, hemos añadido el tamaño del bloque en el que se van a dividir las matrices como un factor primario adicional que se considera en la segunda parte del análisis. Esta es una variable cuantitativa que se describe a continuación:

Tamaño de Bloque

El tamaño de bloque determina cómo se dividen las matrices de entrada para la multiplicación por bloques. Al dividir las matrices en bloques más pequeños, se puede mejorar el aprovechamiento de las propiedades de localidad espacial y temporal de la caché, lo que reduce los tiempos de acceso a la memoria principal y mejora el rendimiento. Se consideran los

siguientes tamaños de bloque: 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192. Estos tamaños se consideraron utilizando potencias de dos sucesivas hasta llegar a 2^{13}, que es la cantidad máxima de datos por bloque que podemos ingresar en la memoria caché L3 (la más grande). La idea fue utilizar tamaños que no superaran en ningún caso los límites de la caché.

Equipo de cómputo

El equipo de cómputo en el que se realizan los experimentos puede alterar seriamente los resultados, a pesar de que no hacía parte del planteamiento del experimento inicial, al hacer un análisis preliminar sobre los datos, se notó una diferencia significativa en las respuestas del tiempo de los algoritmos, incluso cuando se intentaba controlar este factor en una sala donde todos los equipos tienen las mismas características, por lo que en vista de este suceso, más adelante verificaremos la interacción de este factor en el experimento.

Variable de Respuesta

La variable de respuesta en este experimento es el **tiempo de ejecución** de los algoritmos de multiplicación de matrices. Esta variable mide el tiempo total que toma cada algoritmo para completar la multiplicación de dos matrices de un tamaño específico. Debido a la variabilidad en los tamaños de las matrices, es crucial normalizar los datos de tiempo de ejecución para hacer comparaciones justas y significativas.

Para normalizar los tiempos de ejecución, se utilizará la fórmula:

$$[\text{double timeNormalized} = \frac{\text{seconds} \times 1.0e9}{N^3}]$$

donde seconds es el tiempo total de ejecución en segundos y N es el tamaño de la matriz. Esta normalización convierte el tiempo total en nanosegundos por operación, permitiendo una comparación objetiva del rendimiento de los algoritmos independientemente del tamaño de las matrices utilizadas. Tanto los valores de tiempo total de ejecución como los valores

normalizados serán reportados para proporcionar una visión completa y detallada del desempeño de los diferentes algoritmos bajo diversas condiciones experimentales.

FACTORES SECUNDARIOS

En este experimento, se identificaron varios factores secundarios que podrían influir en los resultados. Estos factores son considerados secundarios porque, aunque no son el foco principal del estudio, tienen el potencial de afectar la validez y consistencia de los resultados si no se controlan adecuadamente. A continuación, se describe por qué cada uno de estos factores es considerado secundario y cómo se controlaron para minimizar sus efectos.

1. **Ubicación y condiciones ambientales**: Factores como la temperatura y la humedad pueden afectar el rendimiento del hardware, especialmente en sistemas sensibles a las condiciones térmicas. Variaciones en el entorno físico podrían influir en la consistencia de las pruebas.

2. **Horario de ejecución**: La carga del sistema operativo y de la red puede variar a lo largo del día, afectando el rendimiento de los algoritmos. Las fluctuaciones en la carga del sistema pueden introducir variabilidad en los tiempos de ejecución.

3. **Sistema operativo**: Diferentes versiones o configuraciones del sistema operativo pueden gestionar los recursos del sistema de manera distinta, afectando el rendimiento de las pruebas. Las actualizaciones del sistema operativo pueden cambiar el comportamiento del sistema.

4. **Compilador y configuración de compilación**: Las opciones de compilación y las versiones del compilador pueden optimizar el código de manera diferente, afectando el rendimiento de los algoritmos. Sin un control uniforme, las diferencias en el rendimiento podrían atribuirse erróneamente a los algoritmos en lugar de a las configuraciones del compilador.

5. **Secuencia de ejecución**: El orden en el que se ejecutan las pruebas puede influir en los resultados debido a factores como la memoria caché y otros efectos temporales del sistema. No ejecutar las pruebas de manera aleatoria puede introducir sesgos en los resultados.

6. **Uso exclusivo del computador**: La ejecución de otras aplicaciones o procesos en el computador puede consumir recursos del sistema (CPU, memoria, E/S de disco), afectando el rendimiento de los algoritmos. La multitarea podría introducir variabilidad y ruido en los tiempos de ejecución.

Control de los factores secundarios

Para minimizar los efectos de estos factores secundarios, se adoptaron las siguientes medidas de control:

- **Condiciones ambientales constantes**: Realizar las pruebas en un entorno controlado para que las condiciones ambientales sean las mismas en todas las corridas.
- **Horario de ejecución constante**: Ejecutar las pruebas a la misma hora del día para evitar variaciones en la carga del sistema.
- **Uniformidad del sistema operativo**: Utilizar la misma versión del sistema operativo y mantener todas las actualizaciones al mismo nivel.
- **Consistencia en el compilador:** Usar la misma versión del compilador con las mismas configuraciones de compilación para todas las pruebas. El compilador utilizado para todo el experimento fue Visual Studio 2022 con arquitectura de 64 bits (x64).
- **Automatización del orden de ejecución**: Utilizar un script de PowerShell para automatizar la secuencia de ejecución de las pruebas según el orden predefinido por la matriz de pruebas en un archivo de texto plano.
- **Uso exclusivo del computador**: Asegurar que durante la ejecución de las pruebas, el equipo no se utilice para ninguna otra tarea, evitando así que otras aplicaciones interfieran con el rendimiento de los algoritmos. Esto incluyó evitar el uso de dispositivos de entrada, como el ratón y el teclado.

RECOLECCIÓN DE DATOS

El procedimiento de recolección de datos en este experimento se llevó a cabo de manera sistemática y cuidadosa para garantizar la precisión y la fiabilidad de los resultados. A continuación, se detallan los pasos que se siguieron:

1. Preparación del Entorno de Pruebas:

- Verificar que los equipos de cómputo en el laboratorio Computador lab LHWXX estén configurados correctamente y tengan el mismo entorno de ejecución.
- Asegurarse de que los algoritmos están implementados y listos para ejecutarse en cada equipo.

2. Selección de Datos de Entrada:

- Definir una variedad de tamaños de matrices para la multiplicación, siguiendo el rango y niveles establecidos anteriormente.
- Generar matrices aleatorias para cada tamaño definido, asegurando la diversidad de datos de entrada.

3. Ejecución de los Algoritmos:

- Ejecutar cada algoritmo seleccionado utilizando los datos de entrada generados.
- Registrar el tiempo de ejecución total para cada combinación de algoritmo y tamaño de matriz.

4. Normalización de Datos:

- Aplicar la fórmula de normalización de tiempo para convertir los tiempos de ejecución en nanosegundos por operación, como se mencionó anteriormente.
- Registrar tanto los tiempos de ejecución originales como los tiempos normalizados en una hoja de cálculo para cada ejecución del algoritmo.

5. Repetición de Ejecuciones:

- Repetir los pasos de ejecución y registro para cada algoritmo y tamaño de matriz varias veces, asegurando la consistencia y obteniendo datos representativos.
- Realizar las repeticiones en intervalos de tiempo adecuados para minimizar las variaciones externas.

6. Control de Variables:

- Mantener constantes todas las variables que no sean las específicamente variables de tratamiento, como el lenguaje de programación, el equipo de cómputo y el entorno de ejecución.
- Registrar cualquier cambio o ajuste realizado durante las ejecuciones para una documentación completa.

7. Análisis y Documentación:

- Analizar los datos recolectados, calcular promedios y desviaciones estándar para cada combinación de algoritmo y tamaño de matriz.
- Generar gráficos y tablas para visualizar los resultados y facilitar la interpretación.
- Documentar todo el proceso de recolección de datos, incluyendo las condiciones de prueba, los resultados obtenidos y cualquier observación relevante durante las ejecuciones.

Este procedimiento garantiza la rigurosidad y la objetividad en la recolección de datos, proporcionando información sólida para el análisis comparativo del rendimiento de los algoritmos de multiplicación de matrices y la evaluación de la efectividad de la optimización propuesta.

UNIDAD EXPERIMENTAL

Las unidades experimentales utilizadas en el experimento fueron los equipos LHW05 y LHW12 del laboratorio, los cuales están equipados con procesadores Intel Core i7.

Características del Procesador:

- **Modelo**: Intel Core i7
- **Cache L1**:
  - Datos: 32KB
  - Instrucciones: 32KB
- **Cache L2**: 256KB
- **Cache L3**: 8MB

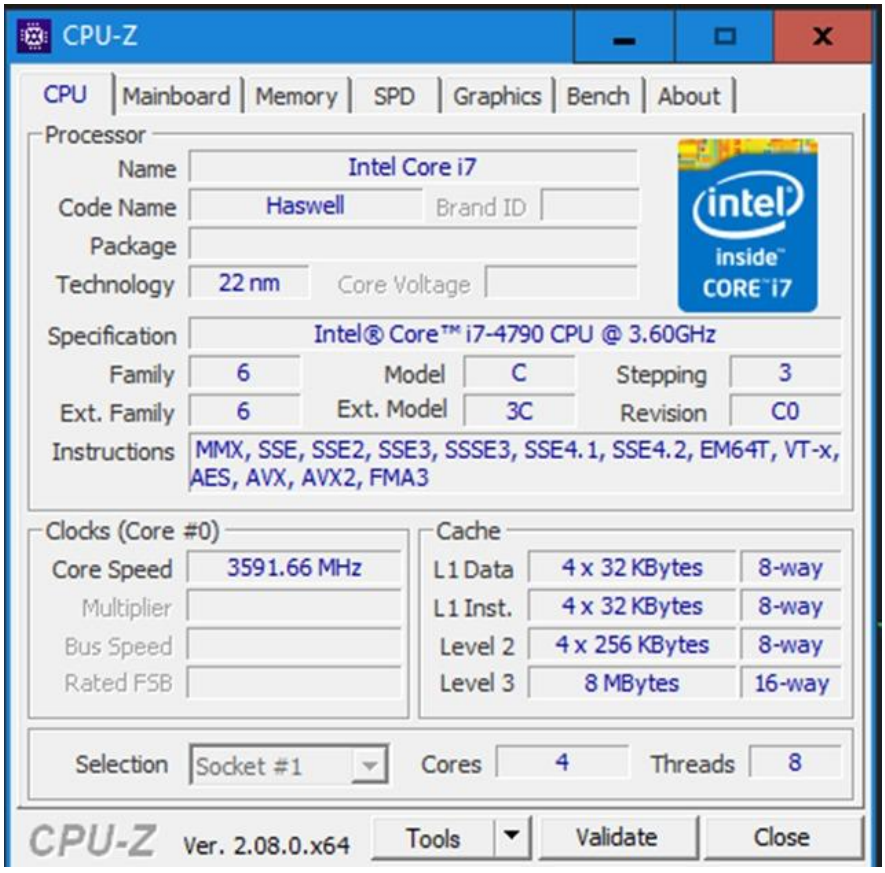

Fig. CPU unidad experimental LHWXX Intel Core i7

ANÁLISIS PRELIMINAR

En el presente estudio, se ejecutaron seis versiones diferentes de algoritmos de multiplicación de matrices en el laboratorio de la Universidad Icesi, utilizando la unidad de ejecución Computador lab LHWXX. Los grupos de laboratorio involucrados en la realización de estos experimentos son: el grupo a cargo del PC LHW05, compuesto por Esteban y Brian, y el grupo a cargo del PC LHW12, conformado por Davide y Oscar.

Para llevar a cabo un análisis preliminar de los algoritmos, se decidió utilizar un tamaño de matriz $(n = 1000)$. Este tamaño fue seleccionado con el objetivo de equilibrar la complejidad computacional y la capacidad de observación de los efectos de la localidad de caché en el rendimiento de los algoritmos. La elección de $(n = 1000)$ permite generar una carga de trabajo lo suficientemente significativa para destacar las diferencias en el comportamiento de los algoritmos bajo condiciones controladas, sin que el tiempo de ejecución se vuelva excesivamente largo.

Además, para asegurar la validez estadística de los resultados, se realizó un cálculo del número de muestras necesarias. Este cálculo se basó en la variabilidad observada en las ejecuciones preliminares y en el nivel de confianza deseado. Se determinó que un número adecuado de repeticiones por algoritmo es esencial para obtener una estimación precisa y confiable del rendimiento medio. Por lo tanto, se ejecutaron múltiples iteraciones de cada versión de los algoritmos para minimizar el impacto de posibles fluctuaciones aleatorias en los tiempos de ejecución.

El análisis preliminar busca proporcionar una visión inicial sobre cómo la localidad de caché impacta el rendimiento de cada versión de los algoritmos de multiplicación de matrices. Los resultados obtenidos servirán como base para un análisis comparativo más detallado y para la propuesta de mejoras en técnicas de optimización de memoria.

Número de muestras preliminar:

En el diseño de experimentos, es crucial determinar el tamaño de muestra necesario para obtener resultados confiables y representativos. Uno de los factores clave en este proceso es el margen de error tolerable, que indica la precisión aceptable en las estimaciones.

El primer paso en nuestro experimento fue realizar una corrida de los algoritmos para obtener datos preliminares y poder calcular el tamaño de muestra necesario. En este paso, se pondrán solo los resultados de un grupo de laboratorio debido a que se llegaron a los mismos resultados. A continuación, presentamos los datos obtenidos durante esta fase inicial:

Datos de la Corrida de Algoritmos

Variable: Normalized (ns)
Número de Muestras (N): 60.0
Media (Mean): 5397.7833
Desviación Estándar (SD): 2989.8544
Error Estándar (SE): 385.9885
Intervalo de Confianza del 95%: (4625.422, 6170.1446)
Estos datos fueron obtenidos del Collab "ANOVA_one_way_Icesi_S3"

Fórmulas Generales

1. **Fórmula para el Margen de Error (E)**:

$$E = Z \times \frac{\sigma}{\sqrt{n}}$$

2. **Fórmula para el Tamaño de Muestra (n)**:

$$n = \left(\frac{Z \times \sigma}{E} \right)^2$$

Donde:

- n : Tamaño de muestra.
- Z : Valor crítico correspondiente al nivel de confianza.

- σ : Desviación estándar de la población.
- E : Margen de error.

Dado que el margen de error tolerable era menor al 10%, inicialmente se planteó un margen de error del 8%. Usando la fórmula del tamaño de muestra con $Z = 1.96$ para un nivel de confianza del 95%, y los datos proporcionados de la variable "Normalized (ns)" con media (μ) de 5397.7833 y desviación estándar (σ) de 2989.8544, obtenemos:

$$E = 0.08 \times 5397.7833 = 431.82266$$

$$n = \left(\frac{1.96 \times 2989.8544}{431.82266} \right)^2$$

Calculando:

$$\frac{1.96 \times 2989.8544}{431.82266} \approx 13.5667$$

$$(13.5667)^2 \approx 184.17$$

El cálculo indicó que se necesitarán al menos 185 muestras para alcanzar el margen de error del 8%. Sin embargo, se optó por elegir 180 muestras, aunque esto implique un margen de error ligeramente superior al 8%. Esta decisión se tomó considerando la necesidad de mantener un margen de error dentro de límites razonables, aun siendo mayor al 8% inicialmente planteado, y garantizar resultados significativos y confiables en el experimento.

Selección de algoritmos:

A partir de la elección del número de muestras, se procedió a ejecutar los algoritmos con el tamaño de matriz definido en ($n = 1000$). Esta configuración permite realizar un análisis estadístico mediante ANOVA (Análisis de Varianza) para evaluar y comparar el rendimiento de las diferentes versiones de los algoritmos.

Para la selección de los algoritmos, se usó ANOVA de una vía donde se intentaba medir si las medias entre distintos algoritmos después de la ejecución de la matriz eran distintas entre sí y luego se hizo ANOVA entre los resultados de los experimentos similares para ver si eran similares.

Estos datos se encuentran en la carpeta Datos en el excel RerpoteDeDatosSemana16LHW05 en la hoja DatosAnovaOneWay, el código para ANOVA está en la carpeta Collabs en el archivo ANOVA_two_way_S4.ipynb. El código fuente ejecutado se encuentra en la carpeta Código Fuente en el archivo matrixProduct_105vFloat_write.c.

Hipótesis:

H0 (Hipótesis nula): El tiempo de ejecución promedio de todos los algoritmos es igual.

H1 (Hipótesis alterna): El tiempo de ejecución promedio de por lo menos un algoritmo es diferente.

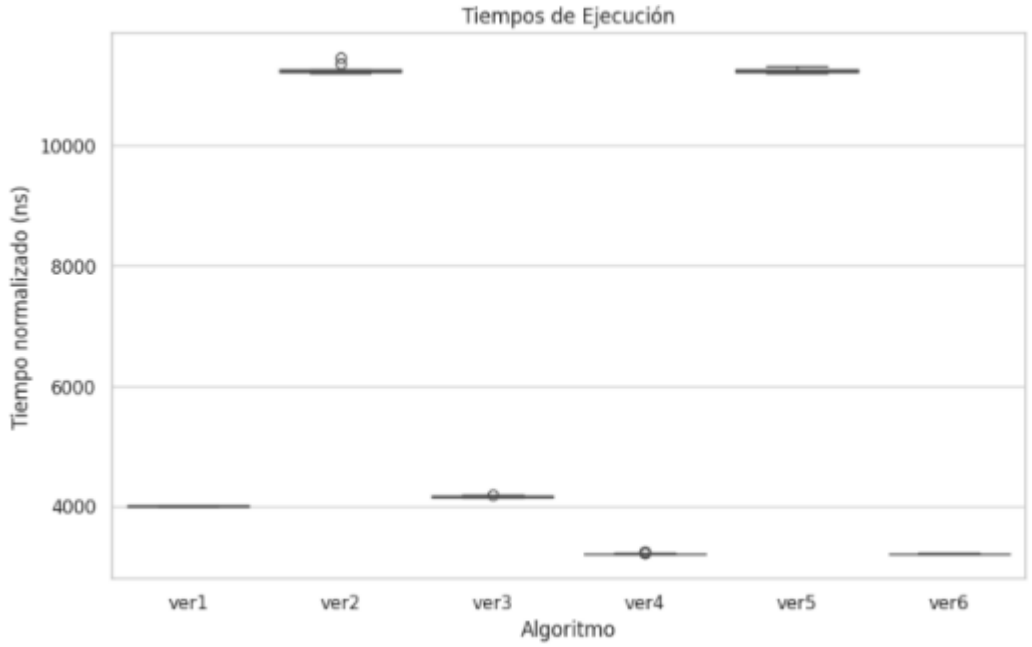

Fig. Gráfico Tiempos de Ejecución de Algoritmos

Comparación entre algoritmo 4 y 6:

Hipótesis:

***H*0:** $\mu 4 = \mu 6$: El tiempo promedio de ejecución del algoritmo de multiplicación de matrices versión 4 es igual al del algoritmo versión 6.

***H*1:** $\mu 4 \neq \mu 6$: El tiempo promedio de ejecución del algoritmo de multiplicación de matrices versión 4 es diferente al del algoritmo versión 6.

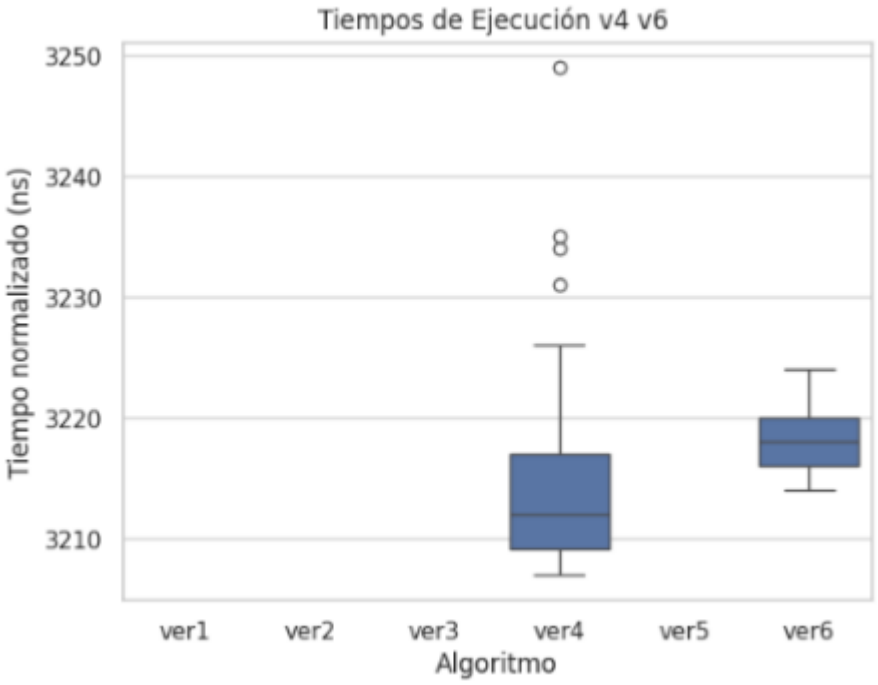

Fig. Gráfico Tiempos de Ejecución v4 v6

```
Estadística de prueba F: 1.864150208028092
Valor p: 0.17741893313955512
alpha: 0.05
================ ---------------- ========================
```

Fig. Prueba Anova v4 v6

Decisión: Dado que p > α = 0.05, aceptamos la hipótesis *H*0.

Conclusión: El tiempo promedio de ejecución del algoritmo de multiplicación de matrices versión 4 es significativamente igual al del algoritmo versión 6. Esto indica que la conclusión del análisis inicial era correcta.

Comparación entre algoritmo 2 y 5

Hipótesis:

***H*0:** $\mu 2 = \mu 5$: El tiempo promedio de ejecución del algoritmo de multiplicación de matrices versión 2 es igual al del algoritmo versión 5.

***H*1:** $\mu 2 \neq \mu 5$: El tiempo promedio de ejecución del algoritmo de multiplicación de matrices versión 2 es diferente al del algoritmo versión 5

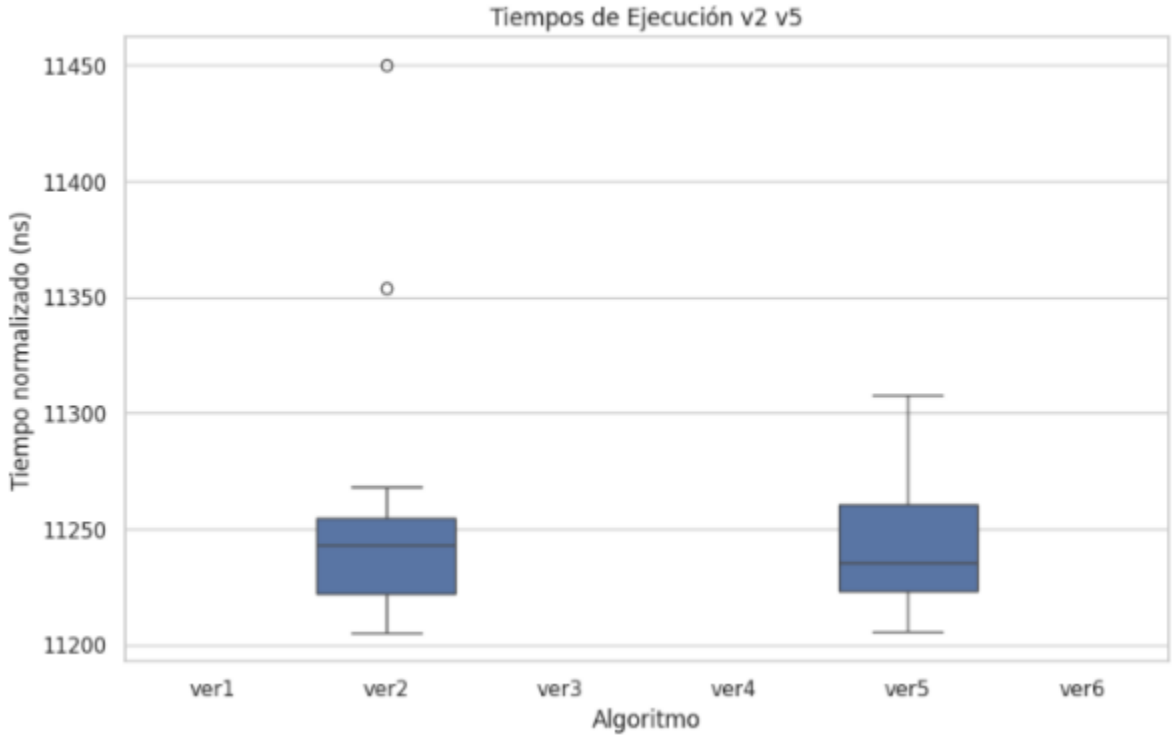

Fig. Gráfico Tiempos de Ejecución v2 v5

```
Estadística de prueba F: 0.367780123087469
Valor p: 0.5465832380357982
alpha: 0.05
================ ---------------- ========================
```

Fig. Prueba Anova v2 v5

Decisión: Dado que p > α = 0.05, entonces se acepta la hipótesis *H*0.

Conclusión: El tiempo promedio de ejecución del algoritmo 2 y 5 para la multiplicación de matrices es significativamente similar. A diferencia del análisis anterior, en este caso la hipótesis nula sí se aceptó. Esto sugiere que el impacto de los datos atípicos en el análisis inicial fue considerable, lo que casi nos llevó a cometer un error tipo I.

Comparación entre algoritmo 1 y 3

Hipótesis:

$\boldsymbol{H0}$**:** $\mu 1 = \mu 3$: El tiempo promedio de ejecución del algoritmo de multiplicación de matrices versión 1 es igual al del algoritmo versión 3.

$\boldsymbol{H1}$**:** $\mu 1 \neq \mu 3$: El tiempo promedio de ejecución del algoritmo de multiplicación de matrices versión 1 es diferente al del algoritmo versión 3.

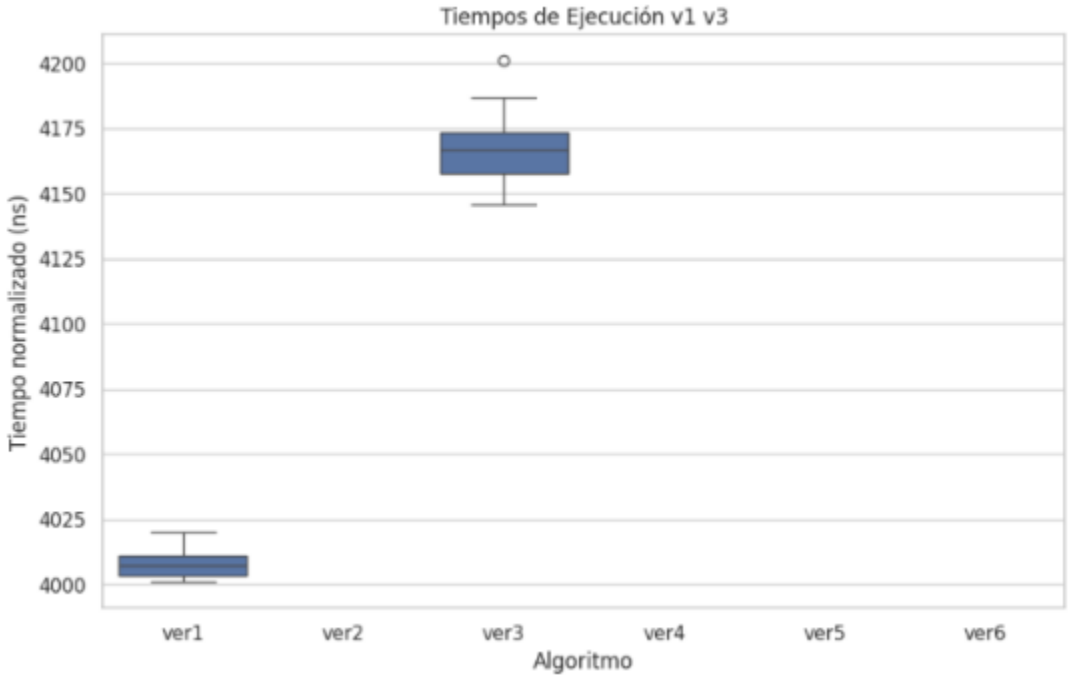

Fig. Gráfico Tiempos de Ejecución v1 v3

```
Estadística de prueba F: 4393.13410792358
Valor p: 2.2639608325709456e-56
alpha: 0.05
================ ---------------- ========================
```

Fig. Prueba Anova v1 v3

Decisión: Dado que $p < \alpha = 0.05$, entonces se rechaza $H0$.

Conclusión: El tiempo promedio de ejecución del algoritmo 1 y 3 para la multiplicación de matrices es significativamente diferente. Esto sugiere que, a pesar de que el análisis anterior se realizó con datos que incluían varios valores atípicos, la conclusión extraída era correcta y no se estaba cometiendo error de ningún tipo.

Tras la recolección de esos datos, tenemos una idea clara de los algoritmos con rendimientos similares: el 4 y el 6, y el 2 y el 5. Para continuar con el experimento, podemos seleccionar uno de cada par (asumiendo que con uno es suficiente para extraer conclusiones sobre ambos) y pasar al siguiente paso, que implica la incorporación de nuevos factores de estudio: el tamaño de la matriz y el tipo de dato (double o float). De este modo, sólo consideraremos el impacto de este nuevo factor en cuatro algoritmos: el 1, 3, 4 y 5.

Tendremos adicionalmente los resultados de otra de las parejas que participaron, pero los resultados son idénticos y se tomaron los mismos algoritmos.

HIPÓTESIS

- Hipótesis Nula (H_0): No hay diferencias significativas en las medias del tiempo promedio de ejecución de las cuatro versiones del algoritmo de multiplicación de matrices.
- Hipótesis Alterna (H_1): Hay diferencias significativas en las medias del tiempo promedio de ejecución de las cuatro versiones del algoritmo de multiplicación de matrices.

Tipo experimento:

Para abordar nuestra hipótesis y las cuestiones por resolver, el tipo de experimento seleccionado para este proyecto es el diseño factorial completo. Este tipo de experimento es adecuado porque permite evaluar todas las combinaciones posibles de los factores primarios y sus niveles, proporcionando una visión completa del impacto de cada versión del algoritmo en el tiempo de ejecución. Además, este diseño es robusto ante variaciones y permite el control efectivo de los factores secundarios.

Factores Primarios

Los factores primarios considerados en este experimento son:
- **Dimensión de las matrices:** 7 tamaños de matrices.
- **Versiones del algoritmo:** Seis versiones diferentes del algoritmo de multiplicación de matrices.
- **Tipo de dato de la matriz:** En este caso, tipo float (32 bits) y double (64 bits).

Limitaciones Existentes

Algunas limitaciones y restricciones consideradas incluyen:

- **Recursos de Hardware:** El experimento se ejecutará en un hardware específico con características constantes para evitar variabilidad en los resultados.
- **Condiciones de Ejecución:** Las pruebas se realizan bajo condiciones controladas para minimizar el ruido experimental, como la carga del sistema y las condiciones ambientales.
- **Tiempo y Recursos Disponibles:** Aunque un diseño factorial completo puede ser costoso en términos de tiempo y recursos, se considera viable dentro del marco del proyecto dado el control de las condiciones de ejecución.

MATRIZ DE DISEÑO

Se utilizó una matriz de diseño completamente aleatorizada para asegurar la validez de los resultados y minimizar los sesgos. A continuación, se presenta la matriz de diseño que incluye el orden de las corridas:

OrdenEst	OrdenCorrida	TipoPt	Bloques	Algoritmo	N
22	1	1	1	5	350
28	2	1	1	5	1500
16	3	1	1	4	450
21	4	1	1	4	1500
23	5	1	1	5	450

2	6	1	1	1	450
25	7	1	1	5	750
9	8	1	1	3	450
8	9	1	1	3	350
13	10	1	1	3	1250
27	11	1	1	5	1250
14	12	1	1	3	1500
19	13	1	1	4	1000
3	14	1	1	1	550
1	15	1	1	1	350
24	16	1	1	5	550
5	17	1	1	1	1000
15	18	1	1	4	350
12	19	1	1	3	1000
7	20	1	1	1	1500
26	21	1	1	5	1000
11	22	1	1	3	750
6	23	1	1	1	1250
20	24	1	1	4	1250
10	25	1	1	3	550
4	26	1	1	1	750
18	27	1	1	4	750
17	28	1	1	4	550

Tabla. Matriz de diseño (Excel hoja MatrizDiseño)

El uso de una matriz completamente aleatorizada permite que cada combinación de dimensión de matriz y versión de algoritmo se pruebe en un orden aleatorio, eliminando posibles sesgos debido a la secuencia de ejecución. Esto asegura que cualquier efecto temporal o de memoria caché se distribuye aleatoriamente, contribuyendo a la fiabilidad de los resultados obtenidos.

Conclusión tipo experimento

El tipo de experimento seleccionado y la matriz de diseño propuesta permiten resolver las hipótesis planteadas de manera efectiva, considerando todos los factores primarios y sus niveles. Este enfoque asegura que las limitaciones existentes no comprometan la validez del experimento y proporciona una base sólida para evaluar las diferencias significativas en el tiempo promedio de ejecución de las diferentes versiones del algoritmo de multiplicación de matrices.

MINIMIZACIÓN DE LA INCERTIDUMBRE EXPERIMENTAL

Para garantizar la validez y confiabilidad de los resultados obtenidos en el experimento, se implementaron medidas y precauciones efectivas para minimizar los efectos de las principales fuentes de incertidumbre experimental identificadas. Estas acciones se diseñaron específicamente para abordar cada fuente de incertidumbre de manera proactiva y consistente, asegurando la consistencia y fiabilidad de los datos recopilados.

Para mitigar la variabilidad en el hardware, se seleccionaron computadoras con las mismas especificaciones técnicas y características, eliminando así posibles diferencias en el rendimiento que podrían surgir de hardware heterogéneo.

Con respecto a las inconsistencias en el entorno ambiental, se llevaron a cabo todas las pruebas en un entorno controlado donde se mantuvieron constantes factores como la temperatura, la humedad y la iluminación. Esto aseguró que las condiciones ambientales no introdujeran variabilidad adicional en los resultados y garantizó la coherencia en la ejecución de las pruebas.

Para abordar las fluctuaciones en la carga del sistema, se ejecutaron todas las pruebas en el mismo horario del día para minimizar las diferencias en la carga del sistema operativo o de la red. Además, se tomaron medidas para cerrar todas las aplicaciones innecesarias y deshabilitar actualizaciones automáticas durante el período de prueba, reduciendo así la interferencia de otros procesos en el rendimiento del sistema.

La posibilidad de errores de medición se mitigó mediante el uso de métodos de medición precisos y calibrados, garantizando así la consistencia y fiabilidad de los datos registrados. Además, se realizaron pruebas de validación cruzada y verificación para confirmar la precisión de los resultados obtenidos.

Para evitar la variabilidad en la configuración del software, se estableció una configuración de software estándar y consistente para todas las pruebas, utilizando la misma versión del sistema operativo, compilador y configuraciones de compilación para todos los algoritmos evaluados.

Finalmente, se implementó un diseño experimental aleatorizado para determinar el orden de las pruebas, asegurando así que no hubiera sesgos introducidos por la secuencia de ejecución y que cualquier efecto temporal o de memoria caché se distribuyera de manera uniforme entre todas las muestras.

Estas medidas y precauciones garantizaron que se minimizaran efectivamente los efectos de todas las principales fuentes de incertidumbre experimental identificadas, permitiendo una evaluación precisa y confiable de los resultados obtenidos en el experimento.

RESULTADOS Y ANÁLISIS

Ahora, procederemos con el análisis del experimento, se adjuntan los archivos de los resultados en la carpeta entrega, donde irán 3 carpetas llamadas Datos, Collabs y Código Fuente. donde se ejecutó de acuerdo con la matriz de diseño y sus factores, lo que resultó en un total de 560 datos. Esto se logró considerando los siguientes parámetros:

- **Dimensión de las matrices**: Se utilizaron 7 tamaños diferentes de matrices.
- **Versiones del algoritmo**: Se emplearon seis versiones distintas del algoritmo de multiplicación de matrices.
- **Tipo de dato de la matriz**: Se exploraron los tipos float (32 bits) y double (64 bits).
- **Número de muestras por algoritmo**: Se fijó un valor de $(n = 10)$.

Para garantizar un buen alcance en nuestro experimento, se seleccionó un tamaño de $n = 10$ para las matrices. Este valor nos permite obtener datos representativos y significativos sin exceder los límites de la capacidad de almacenamiento y procesamiento.

El conjunto resultante de datos, obtenidos mediante la combinación de todos los niveles de los factores mencionados, consiste en 560 registros que conforman nuestra base de datos para el análisis.

La combinación de 7 tamaños de matrices, 6 versiones de algoritmos y 2 tipos de datos de matriz (float y double) con ($n = 10$) proporciona una buena cobertura para el experimento. Esta variedad de factores permite explorar diferentes dimensiones y complejidades en la multiplicación de matrices, desde matrices pequeñas hasta grandes, algoritmos variados y tipos de datos distintos para evaluar el rendimiento en términos de precisión y eficiencia. Con un total de 560 puntos de datos resultantes de la multiplicación de todos los niveles de estos factores, tenemos una base sólida y completa para el análisis, lo que garantiza una visión exhaustiva de cómo estos elementos afectan el proceso de multiplicación de matrices y nos permite extraer conclusiones significativas sobre las mejores prácticas y optimizaciones en este contexto específico.

En esta sección, presentaremos los resultados obtenidos del experimento sobre la multiplicación de matrices y su relación con la localidad de caché. Los datos recopilados se han organizado en diferentes tablas para facilitar su análisis y comprensión. A continuación, explicaremos cada columna que se encuentra en estas tablas presentes en el Excel:

1. **Version:** Esta columna indica la versión específica del algoritmo de multiplicación de matrices que se ha evaluado. Cada versión representa una configuración única del algoritmo, considerando factores como el tipo de dato (double o float), el tamaño de la matriz, y el tipo de algoritmo utilizado.

2. **Type:** Indica el tipo de dato utilizado en la versión del algoritmo, siendo "Double" para datos de doble precisión y "Float" para datos de precisión simple.

3. **Sample:** Representa el número de muestra o ejecución del experimento. Cada muestra corresponde a una ejecución del algoritmo bajo condiciones específicas, como el tamaño de la matriz y el tipo de dato utilizado.

4. **N:** Indica el tamaño de la matriz utilizado en la ejecución del algoritmo.

5. **Time(s):** Esta columna muestra el tiempo de ejecución en segundos del algoritmo para la matriz correspondiente.

6. **Normalized (ns):** Muestra el tiempo normalizado en nanosegundos por operación del algoritmo. Este valor proporciona una medida estandarizada del rendimiento del algoritmo, teniendo en cuenta el tamaño de la matriz y el tipo de dato utilizado.

En la página de Excel "*DatosAnovaOneWay*" se han recopilado los datos utilizados para la selección de los algoritmos que fueron estadísticamente diferentes. La página "*DatosFloat*" contiene los resultados específicos de los algoritmos con datos de tipo float, mientras que la página "*DatosDouble*" presenta los resultados para los algoritmos con datos de tipo *double*.

En la página "*TresFactores*" se encuentran los datos tanto para Double como para Float en la matriz de diseño, lo que permite comparar el rendimiento de los algoritmos en diferentes condiciones. La página "*MatrizDiseño*" muestra la orden de corrida de las muestras, proporcionando información sobre la distribución de las ejecuciones del experimento.

La tabla de ejecutables se encuentra en la página "*Ejecutables*", mostrando detalles específicos de cada versión del algoritmo utilizada en el experimento. Finalmente, en las páginas de "*RegresionDouble*" y "*RegresionFloat*" se presenta el análisis de regresión de los algoritmos, ofreciendo una visión más profunda de la relación entre las variables y el rendimiento de los algoritmos de multiplicación de matrices en términos de la localidad de caché.

Estas tablas y análisis de los resultados permiten una evaluación detallada del impacto de la localidad de caché en el rendimiento de los algoritmos de multiplicación de matrices, así como la identificación de patrones y tendencias significativas en los datos obtenidos durante el experimento.

En los archivos matrixProduct_Double_write_105.c y matrixProduct_105vFloat_write.c se encuentra el código fuente ejecutado y los datos se registraron en la carpeta Datos en el excel RerpoteDeDatosSemana16LHW05. Para estos análisis el archivo de Anova utilizado es el "three_Factors_S16".

Análisis para el grupo del LHW 05:

Es necesario examinar la interacción entre los algoritmos seleccionados después del análisis de varianza (ANOVA). En el gráfico siguiente, se ilustra la relación entre el tamaño de la matriz y el tiempo normalizado para cada algoritmo. Los datos almacenados en las matrices para este análisis inicial son de tipo float de 32 bits y posteriormente se hacen para datos del tipo double de 64 bits.

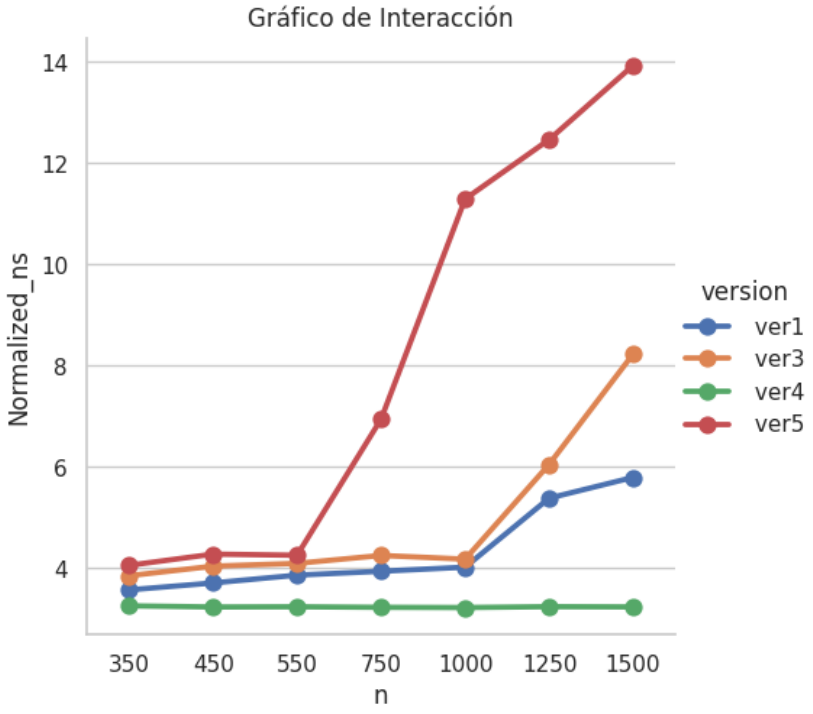

Fig. Gráfico interacción matriz float

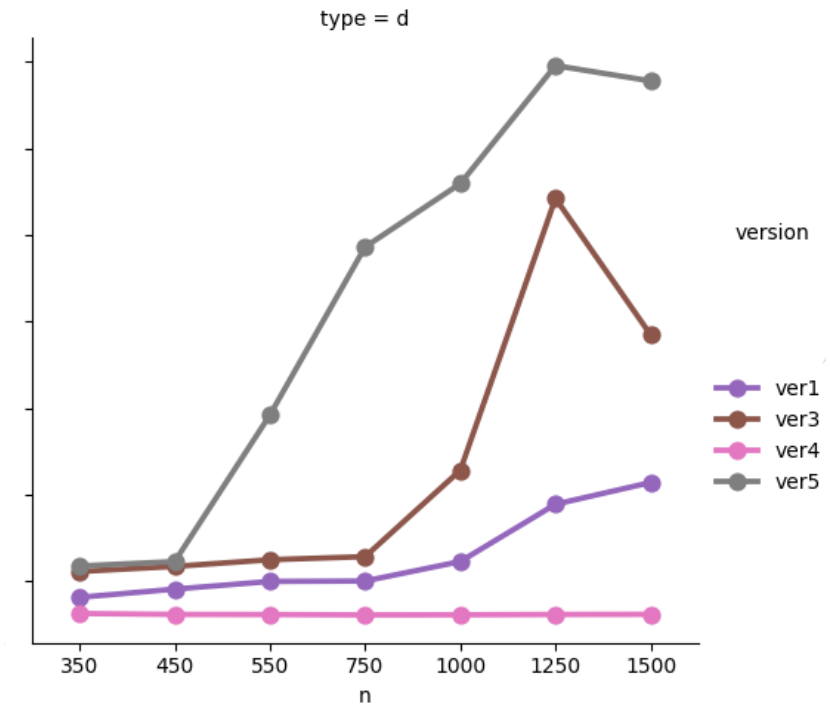

Fig. Gráfico interacción matriz double

En el libro se agrupan los algoritmos (jki(4),kji(6)), (ijk(1), jik(3)), (kij(5), ikj(2)) y nosotros agrupamos (2-5), (4-6), (1), (3), es decir se hacen las misma agrupaciones menos el 1 y 3 que lo tomamos como separado

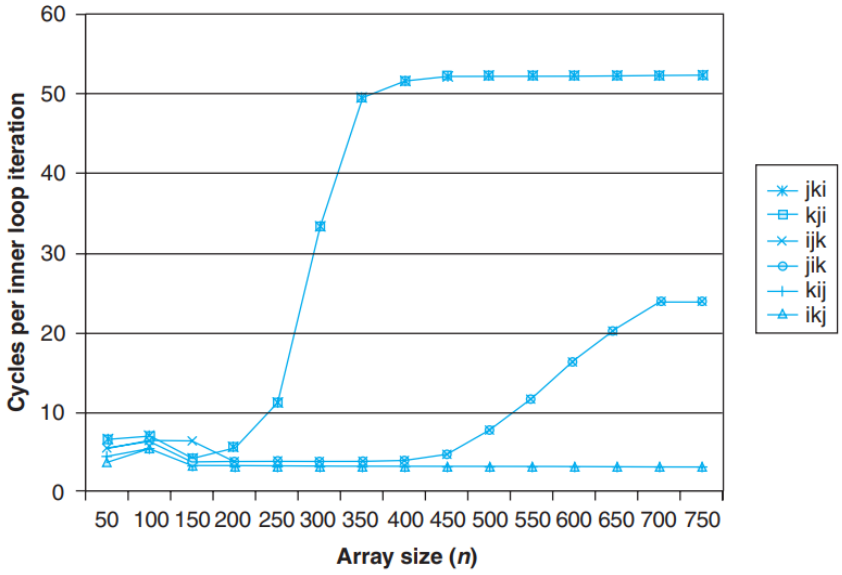

Fig. Gráfico de interacción del libro

Antes de continuar con el análisis de varianza, vamos a resumir los datos que hemos obtenido. La tabla siguiente muestra los tiempos promedio de ejecución, en segundos, para diferentes tamaños de matrices y versiones de algoritmos. Estos tiempos se obtuvieron utilizando datos de tipo Float:

n	**ver1**	**n**	**ver3**	**n**	**ver4**	**n**	**ver5**
350	0.1531	350	0.1651	350	0.1396	350	0.1739
450	0.3379	450	0.3681	450	0.2946	450	0.3898
550	0.6426	550	0.6812	550	0.5386	550	0.7079
750	1.6618	750	1.7935	750	1.361	750	2.9309
1000	4.0197	1000	4.1784	1000	3.2217	1000	11.2945
1250	10.5139	1250	11.818	1250	6.3267	1250	24.3527
1500	19.5549	1500	27.7836	1500	10.9162	1500	47.0271

Tabla. Regresión tipo Float

La siguiente tabla presenta los tiempos promedio de ejecución (en segundos) para distintos tamaños de matrices y variantes de algoritmos utilizados con datos de tipo double:

n	ver1	n	ver3	n	ver4	n	ver5
350	0.1556666667	350	0.1777272727	350	0.1398888889	350	0.1842727273
450	0.3542	450	0.3964	450	0.2954	450	0.401
550	0.72375	550	0.7492222222	550	0.5388	550	1.308888889
750	1.7172	750	2.2302	750	1.392636364	750	4.948111111
1000	4.561272727	1000	7.2139	1000	3.3362	1000	13.20444444
1250	11.29922222	1250	23.94036364	1250	6.8247	1250	30.4725
1500	21.22511111	1500	32.73011111	1500	16.70154545	1500	45.88

Tabla. Regresión tipo Double

Luego del análisis de la gráficas y la regresión, seguimos con el ANOVA multivariado para ver la relación entre los tres factores que determinamos anteriormente.

Factores Analizados:

- **C(versión)**: Se refiere a las diferentes versiones del algoritmo de multiplicación de matrices.

- **C(n)**: Se refiere al tamaño de las matrices utilizadas en el experimento.

- **C(type)**: Se refiere al tipo de datos (float vs. double) utilizados en las matrices.

H0: No hay diferencias significativas en el tiempo de ejecución promedio explicadas por los factores (Tipo algoritmo, Tamaño matriz, Tipo de dato y sus respectivas interacciones)

H1: Hay diferencias significativas en el tiempo de ejecución promedio explicadas por los factores (Tipo algoritmo, Tamaño matriz, Tipo de dato y sus respectivas interacciones)

========== TABLA ANOVA 3 factors ==========

	sum_sq	df	F	PR(>F)
C(version)	1.579452e+03	7.0	2.169596e+04	0.000000e+00
C(n)	1.019672e+03	6.0	1.634104e+04	0.000000e+00
C(type)	1.559152e-09	1.0	1.499197e-07	9.996912e-01
C(version):C(n)	3.129543e+02	42.0	7.164766e+02	2.245674e-226
C(version):C(type)	3.979456e+03	7.0	5.466333e+04	0.000000e+00
C(type):C(n)	-6.673922e-14	6.0	-1.069548e-12	1.000000e+00
C(version):C(type):C(n)	1.951636e+03	42.0	4.468069e+03	0.000000e+00
Residual	5.241555e+00	504.0	NaN	NaN

Primero, podemos comparar la variabilidad explicada por los factores con el valor residual para asegurarnos de que los datos se recolectaron con precisión y de que la influencia de otros factores secundarios no fue un problema. En este caso, la suma de cuadrados total es aproximadamente 8843, lo que es considerablemente mayor que 5.2. Por lo tanto, podemos tener confianza en que los datos se recolectaron correctamente.

Interpretación de los valores de la tabla:

Factor 1: Versión:

- La suma de cuadrados (sum_sq) indica que la versión del algoritmo explica una cantidad significativa de la variabilidad en los datos (1.579452e+03).

- Con 7 grados de libertad (df), el valor F (2.169596e+04) es muy alto, lo que sugiere que la versión del algoritmo tiene un efecto considerable en los resultados.

- El valor p (0.000000e+00) es menor al alpha, lo que indica que la versión del algoritmo afecta de forma significativa el tiempo promedio de ejecución.

2. **Factor 2: Tamaño de Matrices**:

- La suma de cuadrados (sum_sq) muestra que el tamaño de las matrices también tiene un impacto importante en los resultados (1.019672e+03).

- Con 6 grados de libertad (df), el valor F (1.634104e+04) es muy alto, indicando una influencia significativa del tamaño de las matrices.

- El valor p (0.000000e+00) es menor al alpha, lo que indica que el tamaño de la matriz afecta de forma significativa el tiempo promedio de ejecución.

3. **Factor Tipo de Dato** :

- La suma de cuadrados es cercana a cero (1.559152e-09), lo que sugiere que el tipo de datos de la matriz (float vs. double) no tiene un impacto significativo en los resultados.

- Con 1 grado de libertad (df), el valor F (1.499197e-07) es muy cercano a 0, indicando que no hay una influencia significativa en el tipo de dato.

- El valor p (9.996912e-01) es mayor al alpha, lo que indica que el tipo de dato NO afecta de forma significativa el tiempo promedio de ejecución.

Interacciones:

- **Versión y tamaño de la matriz:**

El valor p (2.245674e-226) es menor al alpha, lo que indica que la interacción entre la versión y el tamaño de la matriz afecta de forma significativa el tiempo promedio de ejecución.

- **Versión y tipo de dato**:

El valor p (0.000000e+00) es menor al alpha, lo que indica que la interacción entre la versión y el tipo de dato afecta de forma significativa el tiempo promedio de ejecución.

- **Tipo y tamaño de la matriz:**

El valor p (1.000000e+00) es mayor al alpha, lo que indica que el tipo de dato y el tamaño de la matriz NO afecta de forma significativa el tiempo promedio de ejecución.

- **Versión, tipo de dato, y tamaño de matriz.**

El valor p (0.000000e+00) es menor al alpha, lo que indica que la interacción entre la versión, el tipo de dato y el tamaño de la matriz afecta de forma significativa el tiempo promedio de ejecución.

Residual:

La suma de cuadrados residual es de 5.241555e+00 con 504 grados de libertad. Este término residual es relativamente bajo en comparación con las sumas de cuadrados de los factores principales y sus interacciones, lo que sugiere que el modelo estadístico explica bien la mayoría de la variabilidad en los datos.

Conclusión ANOVA:

El análisis ANOVA muestra que tanto la versión del algoritmo como el tamaño de las matrices son factores críticos que afectan significativamente el rendimiento de los algoritmos de multiplicación de matrices. Las interacciones entre estos factores también son importantes, especialmente la interacción entre versiones del algoritmo y tamaños de matrices. Por otro lado, el tipo de datos de la matriz no tiene un impacto significativo en el rendimiento. El término residual pequeño sugiere que el modelo estadístico utilizado es adecuado para explicar la variabilidad en los datos, proporcionando una base sólida para futuras mejoras y optimizaciones en el proyecto.

Gráficas de regresión:

En esta sección, procederemos a presentar las gráficas correspondientes al análisis de regresión. Estas gráficas permitirán visualizar la relación entre los diferentes factores del experimento y el rendimiento de los algoritmos de multiplicación de matrices. El análisis de regresión nos ayudará a identificar tendencias y patrones significativos que no son fácilmente observables a simple

vista. A continuación, se detallarán las gráficas y los resultados obtenidos para cada combinación de factores considerados en el experimento.

Graficas con tipo de dato double:

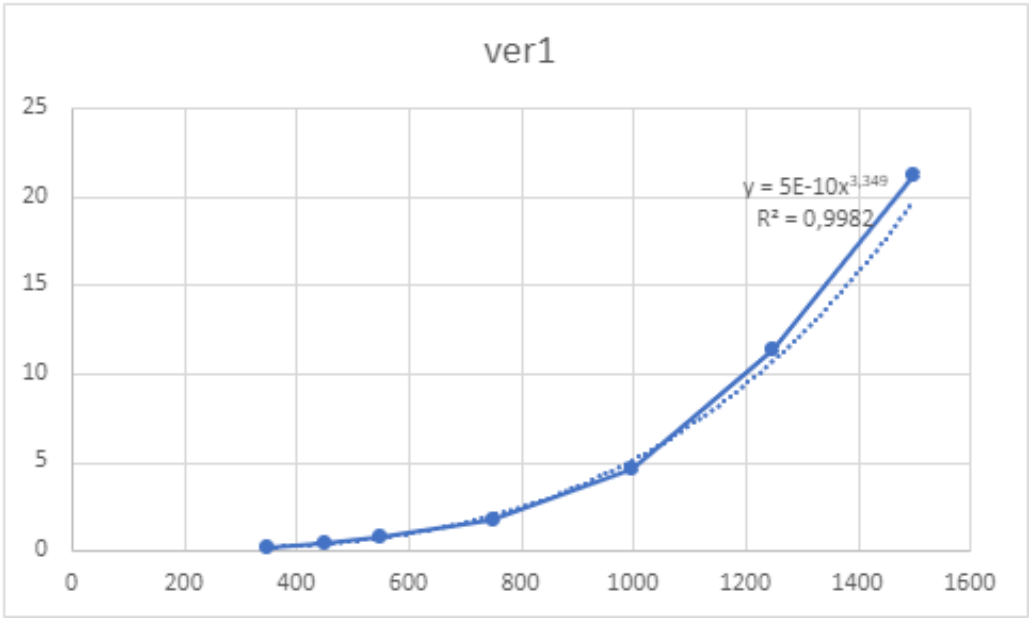

Fig. Gráfica regresión ver1 (Excel hoja RegresionDouble)

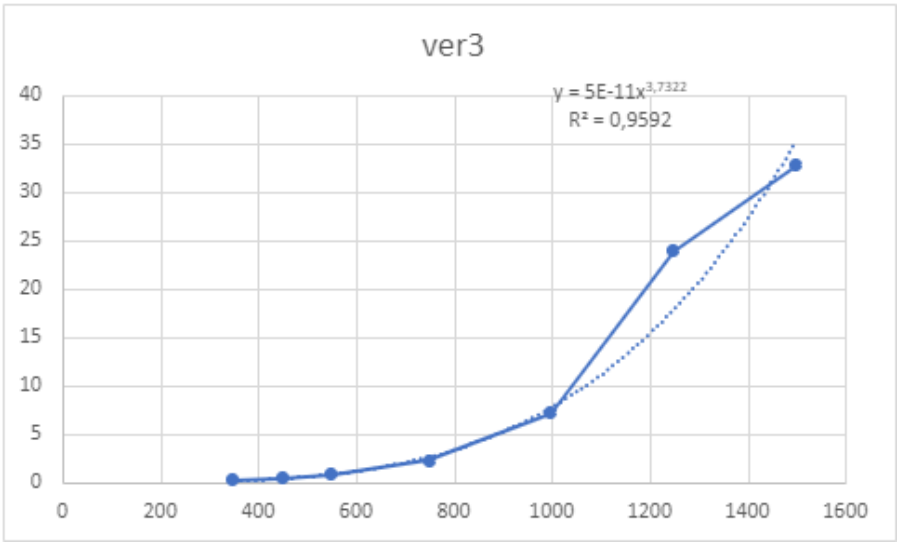

Fig. Gráfica regresión ver3 (Excel hoja RegresionDouble)

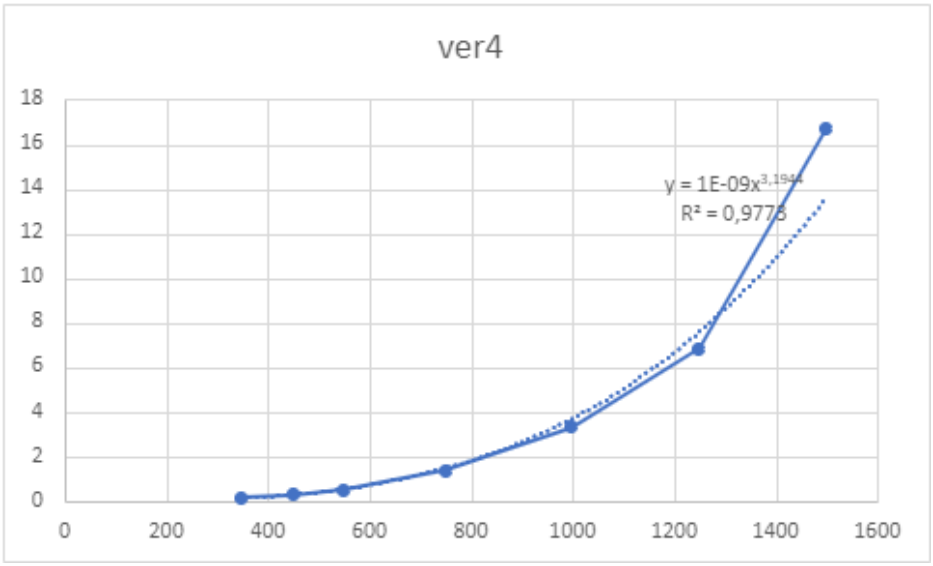

Fig. Gráfica regresión ver4 (Excel hoja RegresionDouble)

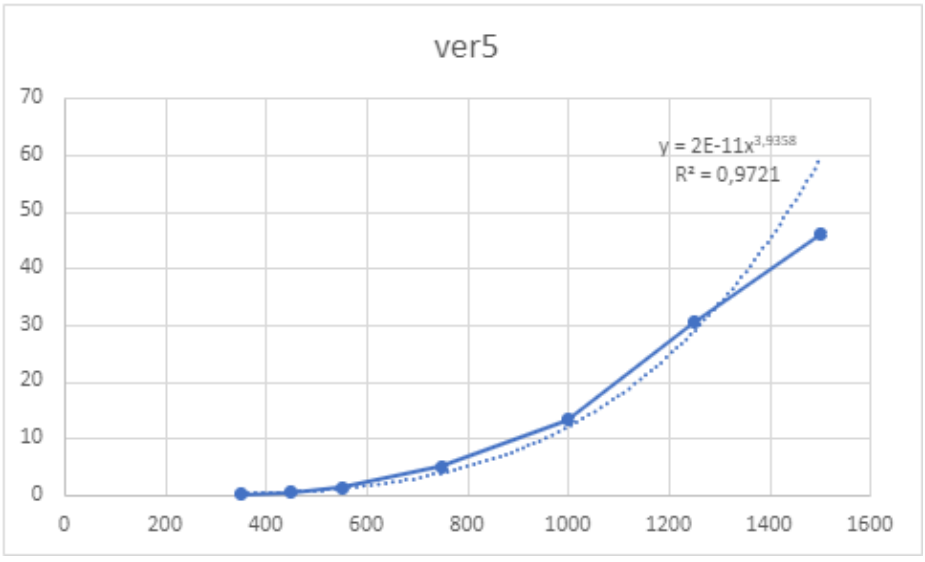

Fig. Gráfica regresión ver5 (Excel hoja RegresionDouble)

Graficas con tipo de dato float:

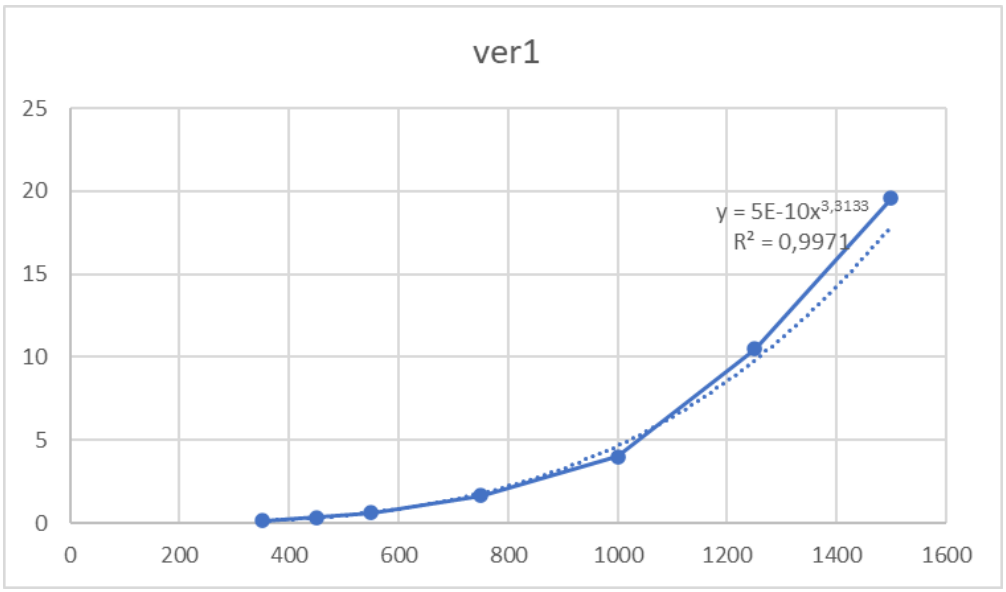

Fig. Gráfica regresión ver1 (Excel hoja RegresionFloat)

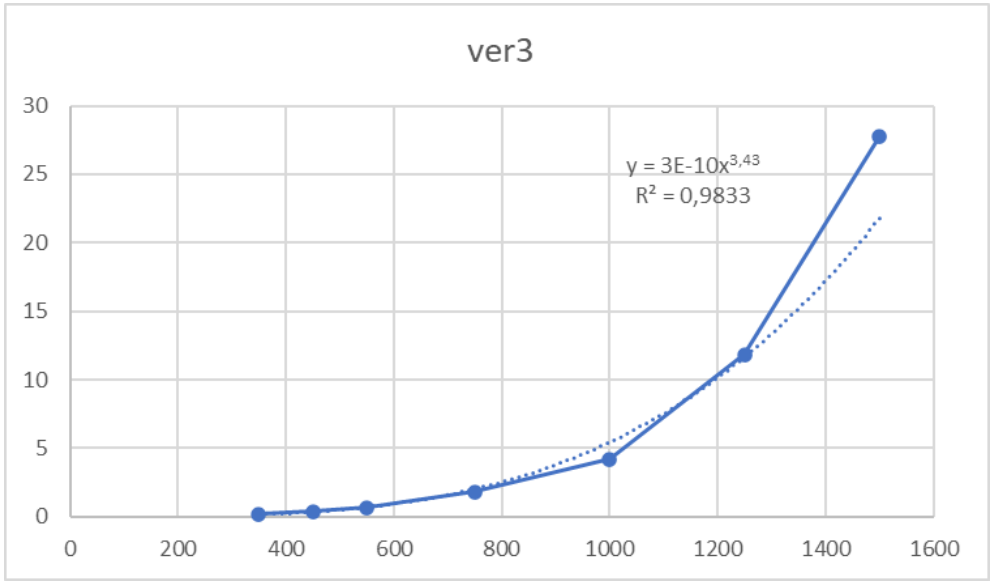

Fig. Gráfica regresión ver 3 (Excel hoja RegresionFloat)

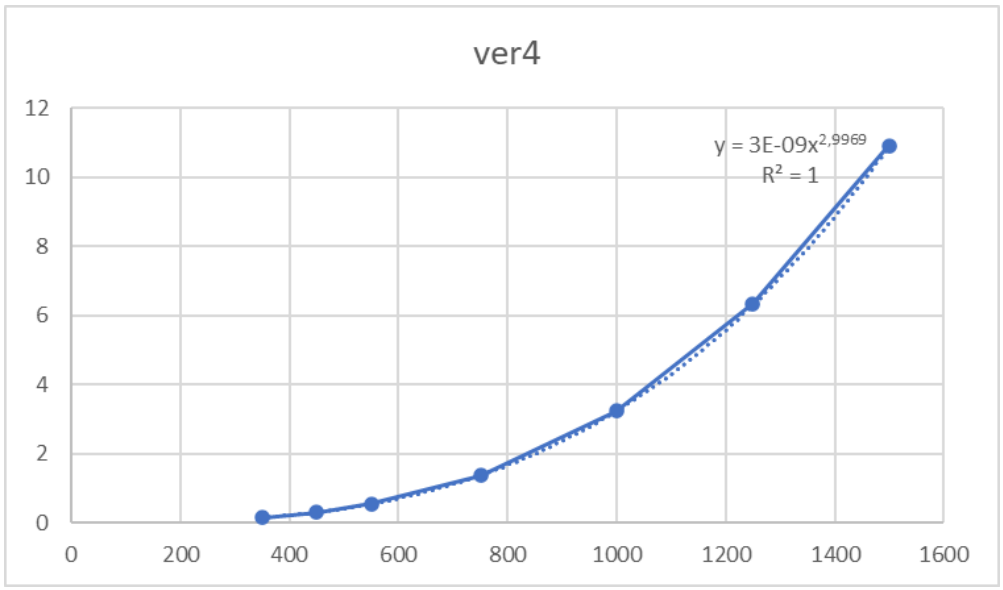

Fig. Gráfica regresión ver 4 (Excel hoja RegresionFloat)

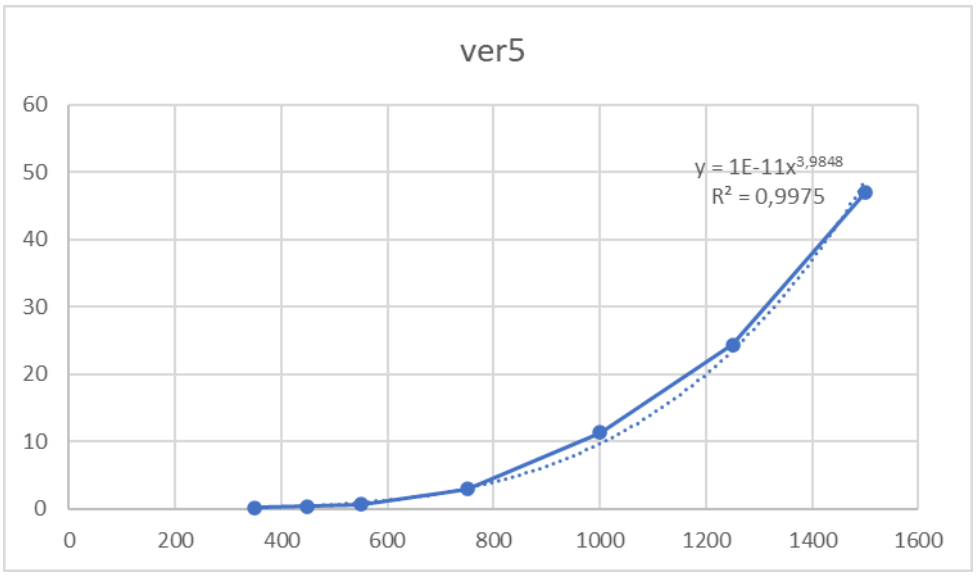

Fig. Gráfica regresión ver 5 (Excel hoja RegresionFloat)

Análisis para el grupo del LHW 12:

Ahora, para el segundo grupo de igual forma se va a investigar cómo interactúan los algoritmos seleccionados después del ANOVA (con los mismos factores). En los siguientes gráficos se mostrará la relación entre el tamaño de la matriz y el tiempo normalizado para cada algoritmo. Al igual que en el grupo anterior, en esta primera fase del análisis, los datos almacenados en las matrices son de tipo float de 32 bits y luego se usa el tipo double de 64 bits.

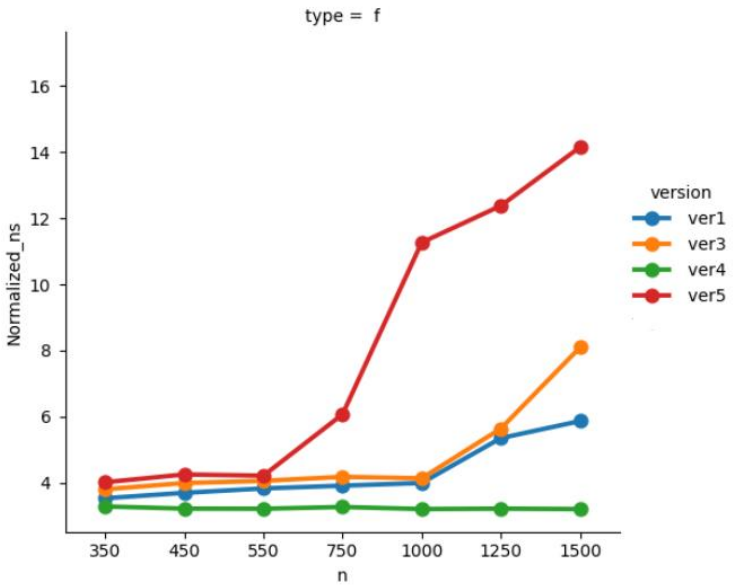

Fig. Gráfico interacción de matriz float (ReporteSem16_DavideFlamini_OscarGomez)

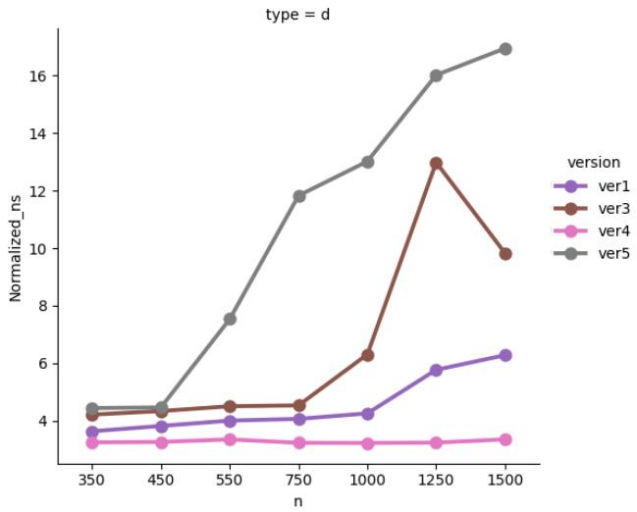

Fig. Gráfico de interacción matriz double (ReporteSem16_DavideFlamini_OscarGomez)

La siguiente tabla muestra los tiempos promedio de ejecución (en segundos) para diferentes tamaños de matrices y versiones de algoritmos usando para los datos de tipo Float:

n	**ver1**	**n**	**ver3**	**n**	**ver4**	**n**	**ver5**
350	0,1518	350	0,1634	350	0,1412	350	0,1725
450	0,3375	450	0,3645	450	0,294	450	0,3881
550	0,6383	550	0,6763	550	0,5361	550	0,7023
750	1,6539	750	1,7668	750	1,3824	750	2,5628
1000	4,0017	1000	4,1423	1000	3,2133	1000	11,2625
1250	10,4609	1250	11,0242	1250	6,2995	1250	24,1655
1500	19,8125	1500	27,3166	1500	10,8427	1500	47,7265

Tabla. RegresionFloat

La siguiente tabla muestra los tiempos promedio de ejecución (en segundos) para diferentes tamaños de matrices y versiones de algoritmos usando para los datos de tipo Double:

n	**ver1**	**n**	**ver3**	**n**	**ver4**	**n**	**ver5**
350	0,15633333	350	0,17718182	350	0,13977778	350	0,18745455
450	0,3547	450	0,396	450	0,2983	450	0,4016
550	0,72316667	550	0,75055556	550	0,5591	550	1,25422222
750	1,7385	750	2,2199	750	1,39863636	750	4,99211111
1000	4,36672727	1000	6,9761	1000	3,3395	1000	13,0045556
1250	11,2817778	1250	24,1781818	1250	6,8282	1250	30,6555

1500	21,2078889	1500	33,2173333	1500	18,2878182	1500	48,8936364

Tabla. RegresionDouble

Gráficas de regresión:

Se presentan las gráficas de regresión:

Graficas con tipo de dato double:

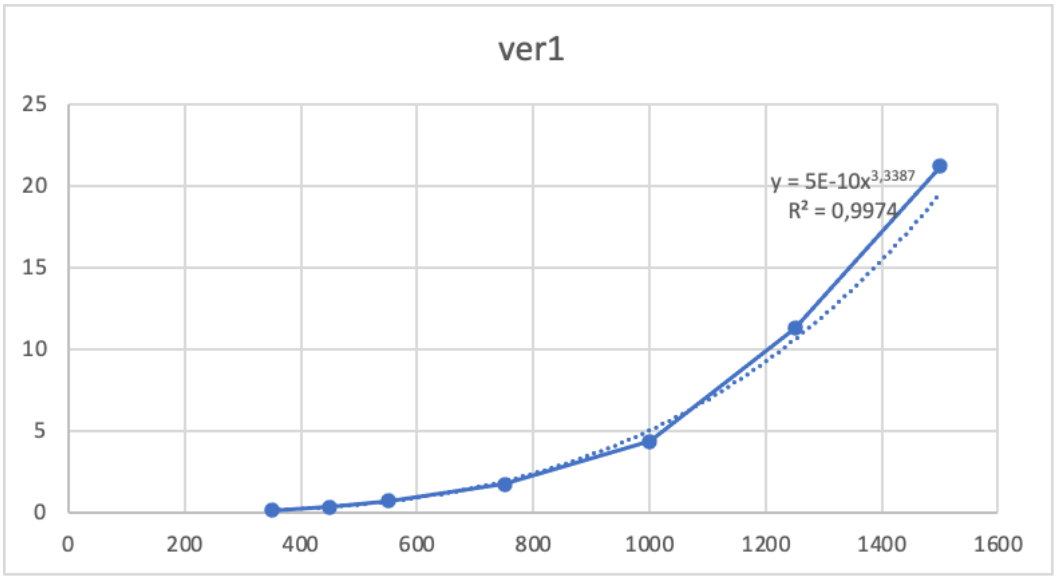

Fig. Gráfica regresión ver1 (Excel hoja RegresionDouble)

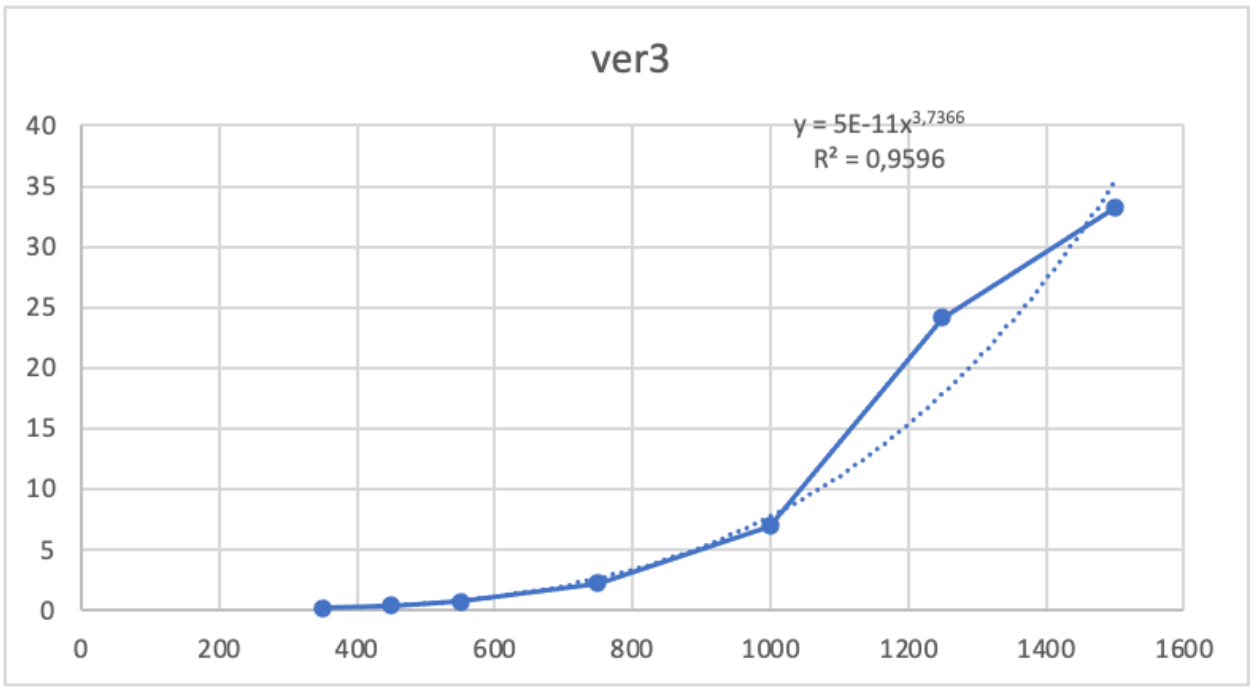

Fig. Gráfica regresión ver3 (Excel hoja RegresionDouble)

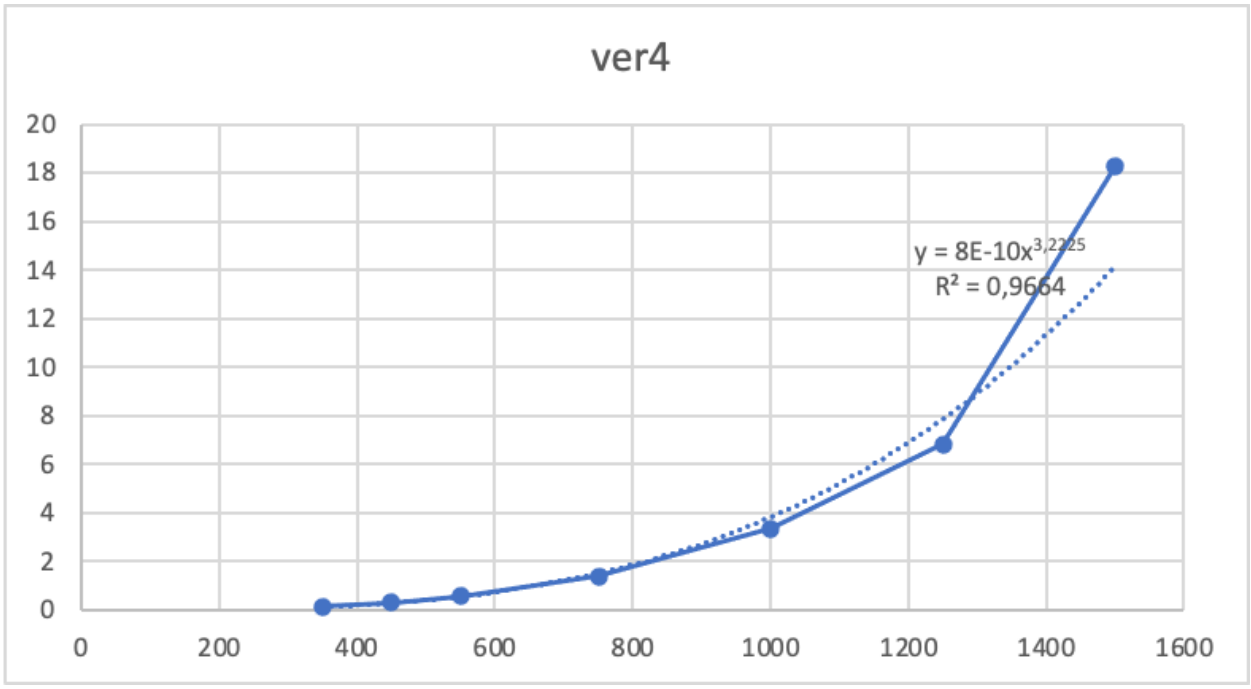

Fig. Gráfica regresión ver4 (Excel hoja RegresionDouble)

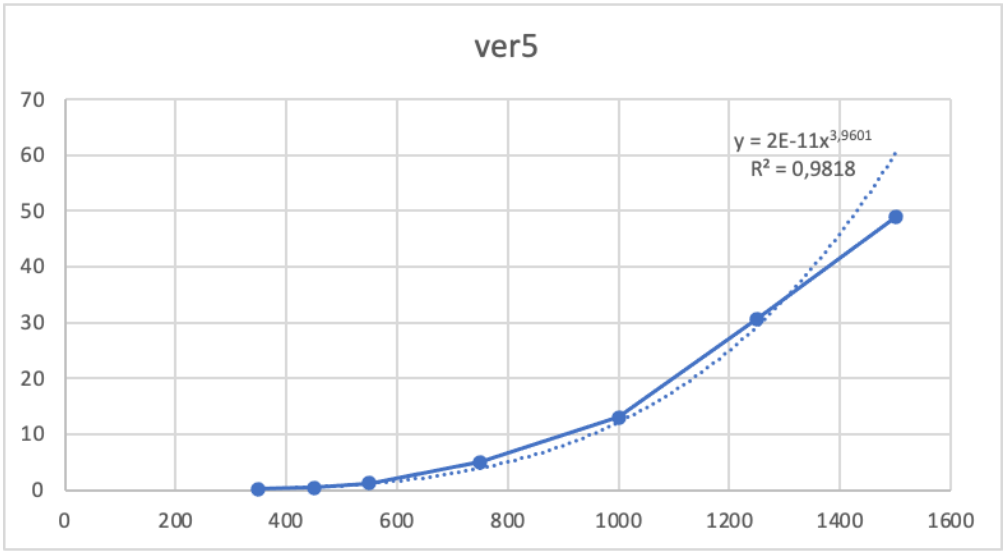

Fig. Gráfica regresión ver5 (Excel hoja RegresionDouble)

Graficas con tipo de dato float:

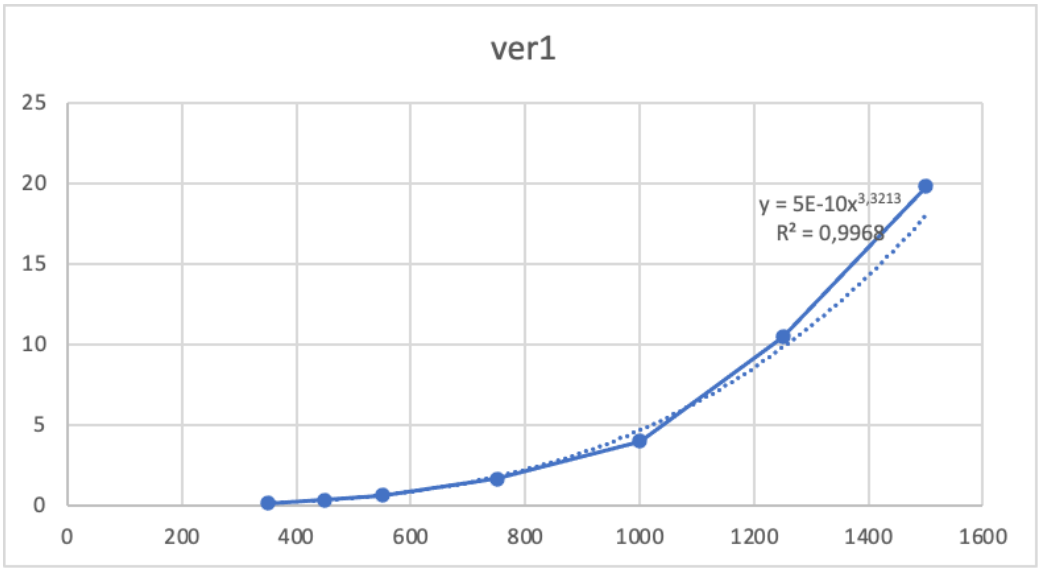

Fig. Gráfica regresión ver1 (Excel hoja RegresionFloat)

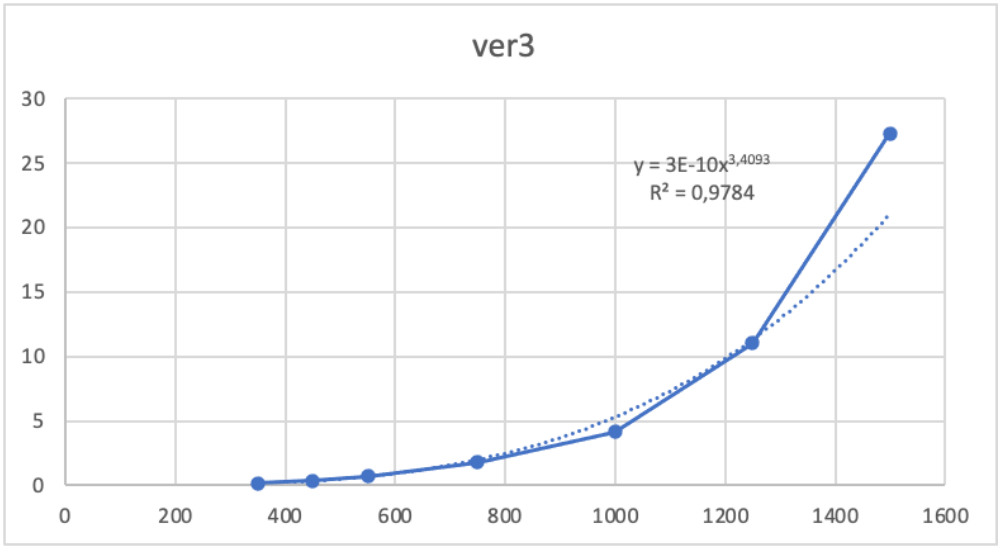

Fig. Gráfica regresión ver 3 (Excel hoja RegresionFloat)

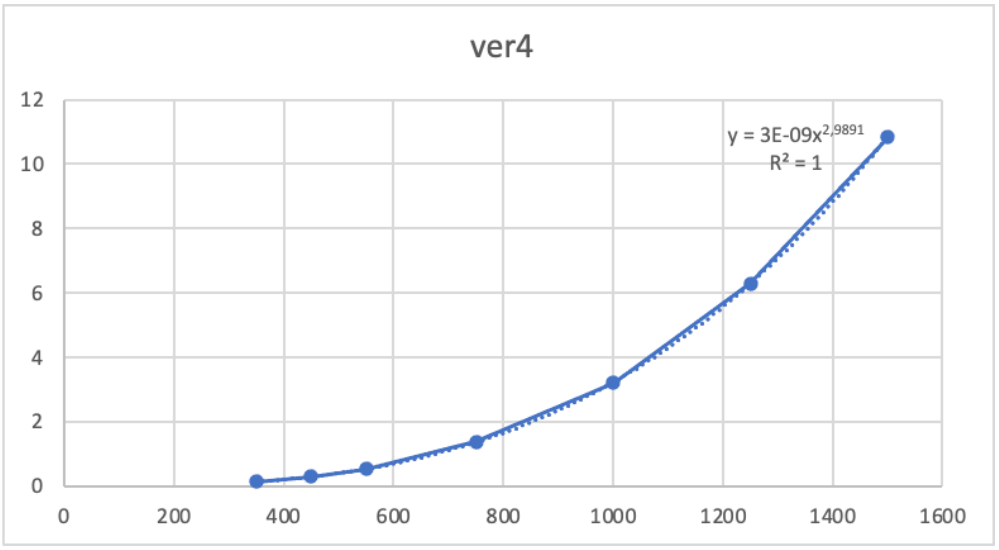

Fig. Gráfica regresión ver 4 (Excel hoja RegresionFloat)

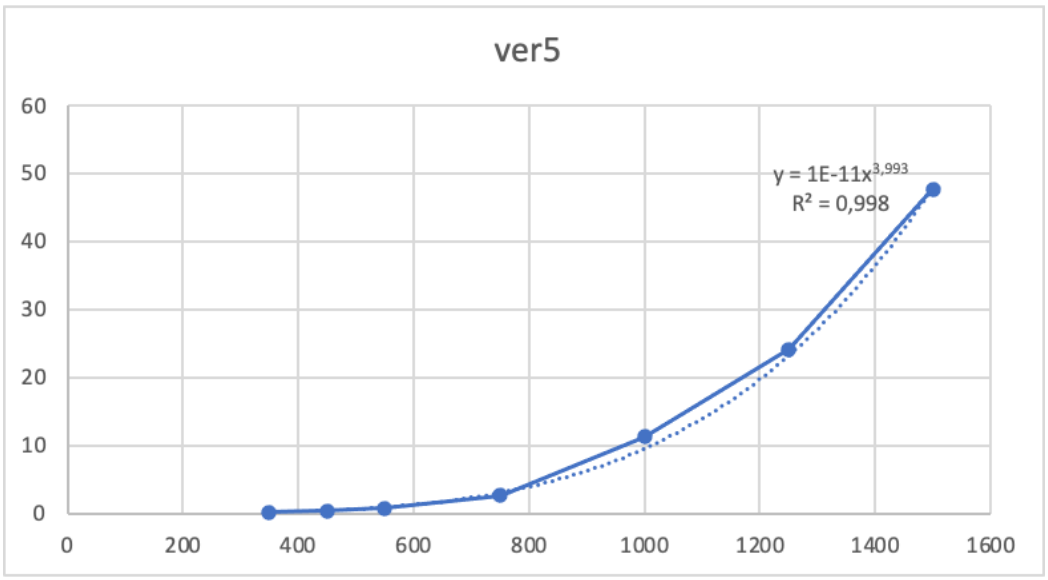

Fig. Gráfica regresión ver 5 (Excel hoja RegresionFloat)

Fórmula del modelo lineal general para 3 factores:

$$\alpha F1 + \beta F2 + \gamma F3 + residual + AF1F2 + BF1F2 + CF1F2 + DF1F2F3$$

Interpretación de ANOVA

```
===========          TABLA ANOVA 3 factors         ================
                              sum_sq     df             F         PR(>F)
C(version)                1.541050e+03    7.0  2.136827e+03  1.993405e-314
C(n)                      1.915191e+03    6.0  3.098213e+03  4.940656e-324
C(type)                  -1.136266e-12    1.0 -1.102886e-11   1.000000e+00
C(version):C(n)           3.517053e+02   42.0  8.127932e+01   1.761295e-62
C(version):C(type)        4.152040e+03    7.0  5.757235e+03   0.000000e+00
C(type):C(n)             -1.101385e-11    6.0 -1.781716e-11   1.000000e+00
C(version):C(type):C(n)   2.139607e+03   42.0  4.944647e+02   0.000000e+00
Residual                  5.192542e+01  504.0           NaN            NaN

Valor p para C(version): 1.993404714e-314
Valor p para C(n): 5e-324
Valor p para type(n): 1.0
alpha: 0.05
```

Factores Analizados:

Son los mismos que analizó el otro grupo, es decir, la versión del algoritmo, el tamaño de la matriz y el tipo de dato.

Interpretación Detallada:

1. **C(version)**:

- La suma de cuadrados (1.541050e+03) y el valor F (2.136827e+03) son altos, indicando que la versión del algoritmo tiene un impacto significativo en los tiempos de ejecución.

- El valor p extremadamente bajo (1.993405e-314) sugiere que las diferencias entre versiones son altamente significativas.

2. **C(n)**:

- La suma de cuadrados (1.915191e+03) y el valor F (3.098213e+03) también son altos, lo que muestra que el tamaño de las matrices influye significativamente en el rendimiento.

- El valor p extremadamente bajo (4.940656e-324) confirma la importancia de este factor.

3. **C(type)**:

- La suma de cuadrados es prácticamente cero (-1.136266e-12), y el valor F es negativo e insignificante (-1.102886e-11), sugiriendo que el tipo de datos (float vs. double) no tiene un impacto significativo.

- El valor p de 1.000000e+00 indica que no hay diferencia significativa debida a este factor.

4. **Interacciones**:

- **C(version):C(n)** y **C(version):C(type)** muestran interacciones significativas con altos valores de F y muy bajos valores p, indicando que las combinaciones específicas de versiones y tamaños de matrices afectan el rendimiento.

- **C(type):C(n)** no es significativa, lo que sugiere que la interacción entre el tipo de datos y el tamaño de las matrices no afecta el rendimiento.

5. **C(version):C(type):C(n)**:

- La interacción entre los tres factores es significativa, con una suma de cuadrados de 2.139607e+03 y un valor F de 4.944647e+02, confirmando la complejidad del impacto combinado de los tres factores en el rendimiento.

Residual:

- La suma de cuadrados residual es de 5.192542e+01 con 504 grados de libertad. Este término residual es relativamente bajo en comparación con las sumas de cuadrados de los factores principales y sus interacciones, lo que sugiere que el modelo estadístico explica bien la mayoría de la variabilidad en los datos.

- Un residual pequeño indica que la variabilidad no explicada por el modelo es mínima, lo cual es deseable y sugiere que los factores seleccionados (versiones del algoritmo, tamaños de matrices, y sus interacciones) capturan adecuadamente las variaciones en los tiempos de ejecución.

Hipótesis:

- La hipótesis nula $((H_0))$ para cada factor y su interacción es que no hay diferencias significativas en los resultados debido a ese factor o interacción.

- Los valores p extremadamente bajos para **C(version)**, **C(n)**, y sus interacciones implican que rechazamos la hipótesis nula para estos factores, confirmando que tienen un impacto significativo en el rendimiento del algoritmo de multiplicación de matrices.

- El valor p alto para **C(type)** y su interacción con **C(n)** implica que no rechazamos la hipótesis nula para estos factores, sugiriendo que no tienen un impacto significativo.

Conclusión ANOVA:

El análisis ANOVA muestra que tanto la versión del algoritmo como el tamaño de las matrices son factores críticos que afectan significativamente el rendimiento de los algoritmos de multiplicación de matrices. Las interacciones entre estos factores también son importantes, especialmente la interacción entre versiones del algoritmo y tamaños de matrices. Por otro lado, el tipo de datos de la matriz no tiene un impacto significativo en el rendimiento. El término residual pequeño sugiere que el modelo estadístico utilizado es adecuado para explicar la variabilidad en los datos, proporcionando una base sólida para futuras mejoras y optimizaciones en el proyecto.

Interpretación de los resultados obtenidos por los dos grupos:

La teoría de la jerarquía de memoria y el análisis del tiempo de acceso a la memoria promedio (AMAT, por sus siglas en inglés) son fundamentales para entender porqué factores como la versión del algoritmo, el tamaño de las matrices y el tipo de datos impactan significativamente en el rendimiento de los algoritmos de multiplicación de matrices.

La jerarquía de memoria en los sistemas de computación se compone de varios niveles, cada uno con diferente capacidad, velocidad y costo. Los niveles típicos incluyen:

1. **Registros**: Pequeños y extremadamente rápidos, ubicados en el procesador.

2. **Caché L1, L2, L3**: Memoria caché de varios niveles, más grande y ligeramente más lenta que los registros.

3. **Memoria principal (RAM)**: Mucho más grande que la caché, pero más lenta.

4. **Memoria secundaria (disco duro, SSD)**: Mucho más grande y mucho más lenta que la RAM.

Tiempo de Acceso a la Memoria Promedio (AMAT)

El AMAT se calcula como:

$$[\text{AMAT} = \text{Hit Time} + \text{Miss Rate} \times \text{Miss Penalty}]$$

Donde:

- **Hit Time**: Tiempo necesario para acceder a un dato en el nivel más alto de la jerarquía de memoria (por ejemplo, caché).

- **Miss Rate**: Tasa de fallos de caché, es decir, la fracción de accesos que no se encuentran en la caché y requieren acceso a un nivel inferior de la jerarquía.

- **Miss Penalty**: Tiempo adicional necesario para acceder al dato desde el nivel inferior cuando ocurre un fallo de caché.

Factores y su Impacto en el Rendimiento

1. **Versión del Algoritmo**:

   - Diferentes versiones de los algoritmos de multiplicación de matrices pueden tener variaciones significativas en su acceso a la memoria. Algoritmos optimizados para la localidad de caché pueden tener menores tasas de fallos de caché (baja **Miss Rate**) y, por tanto, menor AMAT.

   - Por ejemplo, algoritmos como el algoritmo de Bloques o el algoritmo Strassen pueden ser diseñados para minimizar accesos a la memoria principal, aprovechando mejor la caché.

2. **Tamaño de las Matrices**:

   - El tamaño de las matrices afecta directamente la cantidad de datos que deben ser procesados y almacenados en memoria. Para matrices grandes, es más probable que los datos no quepan en la caché, resultando en una mayor **Miss Rate** y, por tanto, un mayor AMAT.

- Matrices pequeñas pueden caber completamente en la caché, resultando en un menor AMAT debido a menos fallos de caché.

3. **Tipo de Datos (float vs. double)**:

- Los tipos de datos también afectan el número de elementos que pueden caber en la caché. Datos de tipo double ocupan el doble de espacio que datos de tipo float, lo que puede aumentar la **Miss Rate**.

- Sin embargo, en el análisis ANOVA proporcionado, el tipo de datos no resultó ser significativo, lo cual puede indicar que en este caso específico, las diferencias en el tamaño de los datos no son lo suficientemente grandes como para afectar notablemente el rendimiento.

Justificación de los Resultados a la Luz de la Teoría

- **C(version)**: La versión del algoritmo tiene un impacto significativo porque diferentes implementaciones pueden optimizar la localidad de datos y reducir la **Miss Rate**, mejorando así el AMAT.

- **C(n)**: El tamaño de las matrices es crucial ya que matrices más grandes pueden provocar un mayor número de fallos de caché, aumentando el AMAT. La optimización del tamaño de los bloques de matrices puede mejorar la localidad espacial y temporal.

- **Interacciones**: Las interacciones significativas entre la versión del algoritmo y el tamaño de las matrices indican que ciertas combinaciones de algoritmos y tamaños son más eficientes en términos de jerarquía de memoria, reduciendo el AMAT.

- **Residual**: Un residual pequeño sugiere que las principales fuentes de variabilidad (versiones de algoritmo y tamaños de matrices) están bien capturadas por el modelo, indicando que estos factores son los más influyentes en el rendimiento debido a sus efectos en la jerarquía de memoria.

La teoría de la jerarquía de memoria y el AMAT explican por qué las versiones del algoritmo y el tamaño de las matrices impactan significativamente en el rendimiento de la multiplicación de

matrices. Las versiones del algoritmo optimizadas para la localidad de caché y los tamaños de matrices que se ajustan mejor a la capacidad de la caché pueden reducir significativamente el AMAT, mejorando el rendimiento global.

COMPARACIÓN DE RESULTADOS

Comportamiento Observado en los Algoritmos

En los gráficos generados a partir de los experimentos con los algoritmos ijk (1), jik (3), jki (4) y kij (5), se observa un comportamiento similar entre los algoritmos hasta que el tamaño de la matriz n alcanza 550. A partir de este punto, se nota un aumento significativo en el tiempo de ejecución para la versión 5 (kij) del algoritmo a medida que n incrementa. Los algoritmos 1 (ijk), 3 (jik) y 4 (jki) mantienen un comportamiento constante dentro de este intervalo hasta que el tamaño de la matriz llega a 1000. En ese momento, se aprecia que los algoritmos 1 (ijk) y 3 (jik) también comienzan a experimentar un aumento significativo en el tiempo de ejecución, al igual que el algoritmo 5 (kij).

Consideraciones sobre la Jerarquía de Memoria, Localidad y ILP

Dado que el tamaño de la matriz n es tan grande que una sola fila de la matriz no cabe en la caché L1, no es posible almacenar toda la matriz en L1. Esto tiene implicaciones importantes para la localidad espacial y temporal, el paralelismo a nivel de instrucción (ILP), y para el rendimiento de los algoritmos:

1. **Para $n < 550$** :

- Aunque no se puede almacenar una fila completa de la matriz en L1, los datos utilizados en las iteraciones del bucle interno aún pueden beneficiarse de la alta velocidad de acceso de L1. Esto se debe a que las sub-matrices más pequeñas y las porciones de las filas que se acceden consecutivamente pueden caber en L1.

- No se observa una diferencia significativa en el tiempo de ejecución entre los algoritmos en este rango porque las referencias de memoria tienen una alta localidad espacial y temporal.

- El paralelismo a nivel de instrucción (ILP) permite que múltiples instrucciones se ejecuten simultáneamente, mejorando la eficiencia y el rendimiento de los algoritmos en este rango.

2. **Para $550 \leq n < 1000$** :

- Los datos exceden la capacidad de la caché L1 y es probable que se comiencen a almacenar en la memoria caché L2, que tiene un tiempo de acceso mayor que L1.

- Este cambio provoca un incremento observado en el tiempo de ejecución, especialmente para la versión 5 (kij) del algoritmo. La versión 5 tiene un acceso más desfavorable a la memoria debido a su patrón de acceso que afecta negativamente la localidad espacial.

- Los algoritmos 1 (ijk), 3 (jik) y 4 (jki) aún mantienen un comportamiento más constante debido a patrones de acceso que benefician la localidad temporal y espacial en L2.

- El ILP sigue beneficiando el rendimiento, pero su impacto se ve reducido por los mayores tiempos de acceso a L2 en comparación con L1.

3. **Para $n \geq 1000$** :

- Los datos exceden la capacidad de la caché L2 y comienzan a almacenarse en la memoria caché L3, que tiene un tiempo de acceso aún mayor.

- Este comportamiento explica por qué los algoritmos 1 (ijk) y 3 (jik), además del algoritmo 5 (kij), experimentan un aumento significativo en el tiempo de ejecución en este punto.

- La memoria L3 tiene un tiempo de acceso considerablemente mayor y la eficiencia de los algoritmos empieza a depender críticamente de la localidad espacial y temporal en un nivel de caché más amplio.

- El ILP tiene menos impacto en este rango debido a los mayores tiempos de acceso de la memoria L3, aunque sigue contribuyendo a una mejora en la ejecución paralela de instrucciones.

Análisis Adicional para Matrices double

Para matrices que almacenan datos de tipo double (64 bits), se observa que la memoria caché L1 alcanza su capacidad máxima con un tamaño de matriz $n = 450$. Este comportamiento se debe a que los datos de tipo double ocupan el doble de espacio que los datos de tipo float. Siguiendo la misma lógica, a partir de $n = 750$, los datos exceden la capacidad máxima de la memoria caché L2.

- Para $n < 450$:

- Aunque una fila completa de la matriz no cabe en L1, las porciones de las filas que se acceden consecutivamente pueden caber, beneficiándose de la alta velocidad de acceso de L1.

- Los datos caben en L1, y se observa un rendimiento constante similar al de los datos de tipo float.

- El ILP permite una ejecución eficiente de las instrucciones, mejorando el rendimiento dentro de este rango.

- Para $450 \leq n < 750$:

- Los datos exceden L1 y se almacenan en L2. La localidad espacial y temporal empieza a jugar un papel más importante.

- El ILP sigue beneficiando el rendimiento, aunque su impacto se reduce debido al mayor tiempo de acceso de L2 en comparación con L1.

- Para $n \geq 750$:

- Los datos exceden L2 y se almacenan en L3, mostrando aumentos significativos en los tiempos de ejecución debido a los mayores tiempos de acceso de la caché L3.

- El ILP tiene menos impacto en este rango debido a los mayores tiempos de acceso de la memoria L3, aunque sigue contribuyendo a una mejora en la ejecución paralela de instrucciones.

Técnicas de Optimización Consideradas

Para mejorar el rendimiento de los algoritmos de multiplicación de matrices, se pueden emplear varias técnicas de optimización:

1. **Arquitectura Superscalar**:

- Los procesadores modernos que utilizan una arquitectura superescalar incorporan múltiples unidades de ejecución. Cada unidad puede ejecutar instrucciones de manera independiente.

- Un procesador superescalar puede obtener múltiples instrucciones a la vez, decodificarlas y asignarlas a diferentes unidades de ejecución si no hay dependencias de datos entre las instrucciones.

2. **Ejecución Fuera de Orden (Out-of-order Execution)**:

- En la ejecución tradicional en orden (in-order execution), las instrucciones se ejecutan en el orden en que aparecen en el programa.

- En contraste, la ejecución fuera de orden permite que el procesador ejecuta instrucciones tan pronto como se resuelven sus dependencias, independientemente de su orden original.

- Esta técnica ayuda a llenar los vacíos en la ejecución causados por dependencias de datos o pausas.

3. **Ejecución Especulativa (Speculative Execution)**:

- Los procesadores pueden usar la ejecución especulativa para predecir el resultado de ramas condicionales.

- El procesador ejecuta especulativamente las instrucciones que siguen a la rama antes de que se evalúe la condición de la rama.

- Si la predicción es correcta, la ejecución continúa sin interrupciones; de lo contrario, los resultados incorrectos se descartan.

4. **Procesadores de Emisión Múltiple (Multiple Issue Processors)**:

- Estos procesadores pueden emitir múltiples instrucciones a las unidades de ejecución simultáneamente.

- Aprovechan el ILP analizando instrucciones para oportunidades de ejecución paralela, permitiendo que se procesen más instrucciones en un solo ciclo.

Conclusiones de comparación:

Ambos grupos de análisis muestran patrones similares y resultados consistentes en cuanto al impacto de los factores evaluados en el tiempo de ejecución de los algoritmos de multiplicación de matrices. Para decidir cuál grupo de algoritmos de multiplicación de matrices es mejor, es importante notar que ambos presentan un comportamiento similar y parecen estar igualmente válidos. La selección entre estos grupos no es evidente debido a la similitud en su desempeño y eficiencia.

El análisis demuestra que la localidad espacial y temporal, junto con la jerarquía de la memoria caché y el paralelismo a nivel de instrucción (ILP), son factores determinantes en el rendimiento de los algoritmos de multiplicación de matrices. Las diferencias en los patrones de acceso a la memoria de cada algoritmo tienen un impacto significativo en su rendimiento, especialmente cuando el tamaño de las matrices supera la capacidad de los niveles más rápidos de la caché. El comportamiento observado en los algoritmos ijk (1), jik (3), jki (4) y kij (5) muestra cómo la capacidad de las diferentes memorias caché influye en el rendimiento a medida que aumenta el tamaño de la matriz, afectando particularmente aquellos algoritmos con patrones de acceso menos favorables a la localidad espacial. Además, el ILP contribuye significativamente a la eficiencia en la ejecución de las instrucciones, mejorando el rendimiento general de los algoritmos en diferentes rangos de tamaño de matriz.

Para optimizar aún más el rendimiento, se podrían considerar técnicas como la arquitectura superscalar, la ejecución fuera de orden, la ejecución especulativa y el uso de procesadores de emisión múltiple. Estas técnicas permiten mejorar la utilización del paralelismo a nivel de

instrucción y gestionar de manera más eficiente los accesos a memoria, lo que puede resultar en una ejecución más rápida y eficiente de los algoritmos de multiplicación de matrices.

Adicionalmente, las especulaciones sobre el uso de la caché se pueden sustentar por la información proporcionada en el experimento del libro "*Computer Systems A programmer 's Perspective*" en la página 627:

Algoritmo de multiplicación de matrices (clase)	**ijk (1) & jik (3) (AB)**	**jki (4) (AC)**	**kij (5) (BC)**
Cargas por iteración	2	2	2
Almacenamientos por iteración	0	1	1
Fallos de A por iteración	0.25	1.00	0.00
Fallos de B por iteración	1.00	0.00	0.25
Fallos de C por iteración	0.00	1.00	0.25
Total de fallos por iteración	1.25	2.00	0.50

Para los cuatro algoritmos de multiplicación de matrices: ijk (1), jik (3), jki (4) y kij (5). se pueden tomar en cuenta varias consideraciones importantes sobre el entorno y las características de las matrices y el sistema de caché:

- Hay una única caché con un tamaño de bloque de 32 bytes (B = 32).
- El tamaño de la matriz n es tan grande que una sola fila de la matriz no cabe en la caché L1.
- El compilador almacena variables locales en registros, por lo que las referencias a variables locales dentro de los bucles no requieren ninguna instrucción de carga o almacenamiento.
- Se considera el paralelismo a nivel de instrucción (ILP) para evaluar cómo el hardware puede ejecutar múltiples instrucciones simultáneamente.

A continuación, se presenta un análisis detallado de los algoritmos ijk (1), jik (3), jki (4) y kij (5), agrupados en clases de equivalencia según las matrices que se acceden en el bucle interno.

- **Clase AB (ijk (1) y jik (3))**: Escanean una fila de la matriz A con un paso de 1 en el bucle interno. Dado que cada bloque de caché contiene cuatro “doublewords” (un "doubleword" sería de 32 bits o sea 4 bytes), la tasa de fallos para A es de 0.25 fallos por iteración. Por otro lado, el bucle interno escanea una columna de B con un paso de n. Dado que n es grande, cada acceso a la matriz B resulta en un fallo, para un total de 1.25 fallos por iteración.

- **Clase AC (jki (4))**: Cada iteración realiza dos cargas y un almacenamiento (a diferencia de las rutinas de la Clase AB, que realizan dos cargas y ningún almacenamiento). Además, el bucle interno escanea las columnas de A y C con un paso de n. El resultado es un fallo en cada carga, para un total de dos fallos por iteración. Intercambiar los bucles ha disminuido la cantidad de localidad espacial en comparación con las rutinas de la Clase AB.

- **Clase BC (kij (5))**: Con dos cargas y un almacenamiento, requieren una operación de memoria más que las rutinas de la Clase AB. Por otro lado, dado que el bucle interno escanea tanto B como C fila por fila, con un patrón de acceso de paso-1, la tasa de fallos en cada matriz es solo de 0.25 fallos por iteración, para un total de 0.50 fallos por iteración.

Consideración del equipo como factor

Dado que ya se ha analizado exhaustivamente los resultados de cada equipo por separado, es importante examinar cómo se comportan los resultados al considerar ambos equipos de cómputo. Anteriormente, se determinó que uno de los factores secundarios que se minimizó fue el equipo utilizado para llevar a cabo el experimento. Esto se basó en la premisa de que todos los equipos contaban con características similares. Sin embargo, en la práctica real, factores como el tiempo, el uso del equipo y el rendimiento de la CPU pueden influir en los resultados. Aunque los equipos sean muy similares, pueden existir diferencias en su calidad de construcción u otros aspectos que puedan afectar el experimento. Esto llevó a considerar el conjunto de estados de los equipos como un factor principal.

Para incorporar el equipo como un factor, se utilizaron los datos previamente utilizados para comparar el comportamiento del tiempo normalizado, excluyendo el "***type***" que ya había sido descartado por el ANOVA realizado anteriormente. Posteriormente, se llevó a cabo un muestreo aleatorio equitativo entre ambos conjuntos de datos para evitar sesgos en los análisis. Estas selecciones se combinaron en una sola tabla, considerando si los datos provenían del equipo **LHW12** o **LHW05**, y esta información se registró en una nueva columna denominada "***sourc***e". Luego, esta tabla se utilizó en el ANOVA como un nuevo factor adicional al "***n***" y "***version***".

Las hipótesis el factor "*source*" en el ANOVA multifactor son:

H0: El factor source no tiene un efecto significativo sobre Normalized_ns.

H1: El factor source tiene un efecto significativo sobre Normalized_ns.

Los siguientes datos se pueden encontrar en comparacion_entre_pcs.ipynb

	sum_sq	df	F	PR(>F)
C(n)	1275.720207	6.0	33707.430962	0.000000e+00
C(version)	1846.327457	3.0	97568.346005	0.000000e+00
C(source)	0.490058	1.0	77.690650	1.960472e-17
C(n):C(version)	1376.268158	18.0	12121.378882	0.000000e+00
C(n):C(source)	0.954999	6.0	25.233259	3.151634e-26
C(version):C(source)	0.467503	3.0	24.704966	6.286321e-15
C(n):C(version):C(source)	3.209884	18.0	28.270816	7.974357e-65
Residual	3.179136	504.0	NaN	NaN

Fig. Resultados del análisis ANOVA con "*source*"

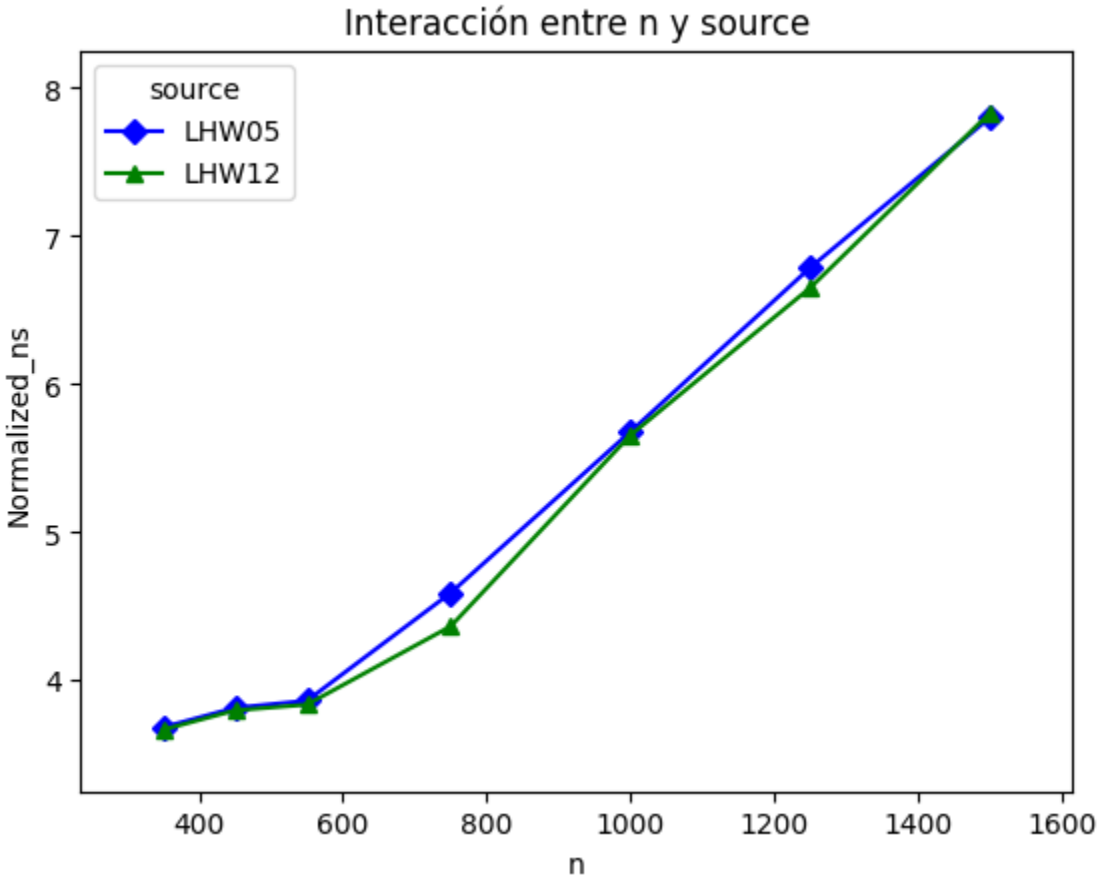

Fig. Gráfica del factor "n" en ambos equipos

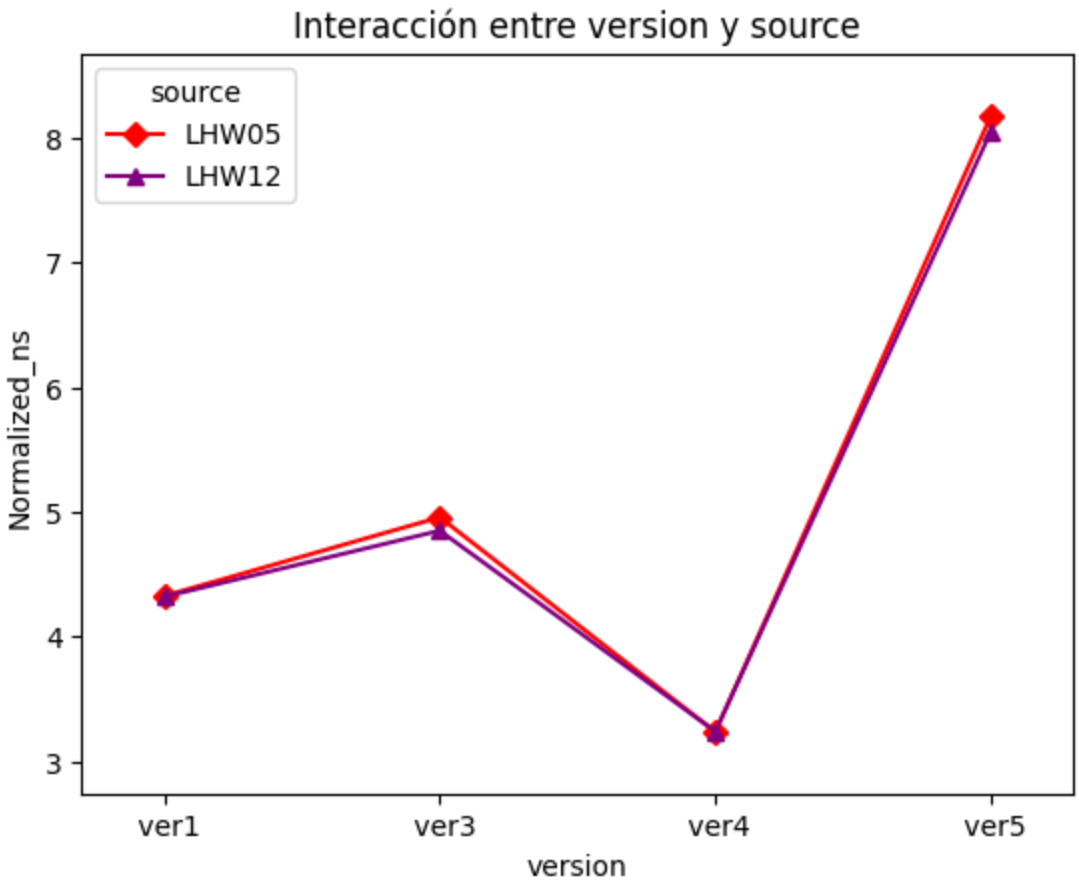

Fig. Gráfica del factor “version” en ambos equipos

Como se puede observar a partir de los valores de “***PR(>F)***” en la variable “***C(source)***”, el valor es considerablemente menor que el nivel de significancia (***alfa***) establecido para el experimento (***0.05***). Por lo tanto, rechazamos la hipótesis nula (***H0***) y aceptamos que “***source***” tiene un impacto significativo en el comportamiento del tiempo normalizado de los algoritmos. Además, tanto en “***C(version):C(source)***” como en “***C(n):C(source)***”, también se rechaza la hipótesis nula, lo que sugiere que “**source**” interactúa tanto con “***version***” como con “**n**”. Esto implica que estos tres factores tienen una influencia significativa en el tiempo.

Propuesta de mejora de algoritmo:

Para mejorar el rendimiento del algoritmo con peor desempeño, que es el número 5, se ha propuesto una estrategia de multiplicación por bloques que utiliza el mismo recorrido de este

algoritmo. Además, se ha estudiado el impacto de dividir estos algoritmos en bloques de diferentes tamaños. La propuesta del algoritmo es la siguiente:

```
for (jj = 0; jj < n; jj += BLOCK_SIZE) {
    for (kk = 0; kk < n; kk += BLOCK_SIZE) {
        for (j = jj; j < jj + BLOCK_SIZE && j < n; j++) {
            for (k = kk; k < kk + BLOCK_SIZE && k < n; k++) {
                for (i = 0; i < n; i++) {
                    C[i + j * n] += A[i + k * n] * B[k + j * n];
                }
            }
        }
    }
}
```

Nota: El código fuente de este algoritmo de mejora se encuentra en el archivo "matrix_mul.c".

Para evaluar el impacto de la estrategia de multiplicación de matrices por bloques, se consideró el tamaño de la matriz como un factor adicional y se realizó un ANOVA de dos vías para medir el impacto del tamaño de bloque con el que el algoritmo trabajará. En este contexto, se plantearon las siguientes hipótesis:

H0: El factor block_size no tiene un efecto significativo sobre Normalized_ns en el algoritmo de multiplicación de matrices por bloques.

H1: El factor block_size tiene un efecto significativo sobre Normalized_ns en el algoritmo de multiplicación de matrices por bloques.

Este ANOVA se realizó utilizando la siguiente matriz de diseño, en la cual los dos factores considerados fueron N y el tamaño de bloque. Es importante destacar que los niveles para el factor N (tamaño de la matriz) son los mismos que se han estado utilizando en todo el experimento desde que se introdujo el factor de tamaño de la matriz. Además, en este ANOVA se eliminó el factor del tipo de dato debido a que en el ANOVA anterior se encontró que este no es significativo. Como se mencionó en el marco teórico, los niveles del factor tamaño de bloque

seleccionados se justifican en el hecho de que se procuró no superar la capacidad de la caché más grande, que es la L3.

Orden	**n**	**bloque**
0	350	128
1	750	32
2	350	8192
3	350	8
4	1000	16
5	1000	8192
6	450	128
7	1000	64
8	1000	128
9	750	512
10	750	128
11	1500	512
12	550	512
13	1250	16
14	450	16
15	1250	1024
16	1500	128
17	550	8
18	1500	2048
19	550	16
20	1250	128

21	1250	512
22	550	256
23	550	8192
24	450	32
25	1500	64
26	1250	256
27	450	512
28	350	4096
29	750	64
30	450	1024
31	550	1024
32	750	1024
33	750	256
34	1500	256
35	550	32
36	550	2048
37	1500	1024
38	1250	64
39	350	64
40	1250	4096
41	1000	32
42	1500	8
43	1250	2048
44	750	4096
45	350	512

46	1000	256
47	450	2048
48	1500	4096
49	1250	8
50	350	256
51	350	1024
52	350	32
53	450	8
54	550	128
55	550	4096
56	750	8192
57	750	8

Tabla. Matriz de diseño algoritmo por bloques

	sum_sq	df	F	PR(>F)
C(n)	0.110402	6.0	35.185000	8.667923e-27
C(block_size)	0.008328	10.0	1.592444	1.343806e-01
C(n):C(block_size)	0.065087	60.0	2.074305	1.294118e-05

```
Valor p para C(n): 8.667923174566661e-27
Valor p para C(block_size): 0.13438062574643636
Valor p para la interacción: 1.2941178923287617e-05
alpha: 0.05
```

Fig. ANOVA para la Multiplicación De Matrices Por Bloques.

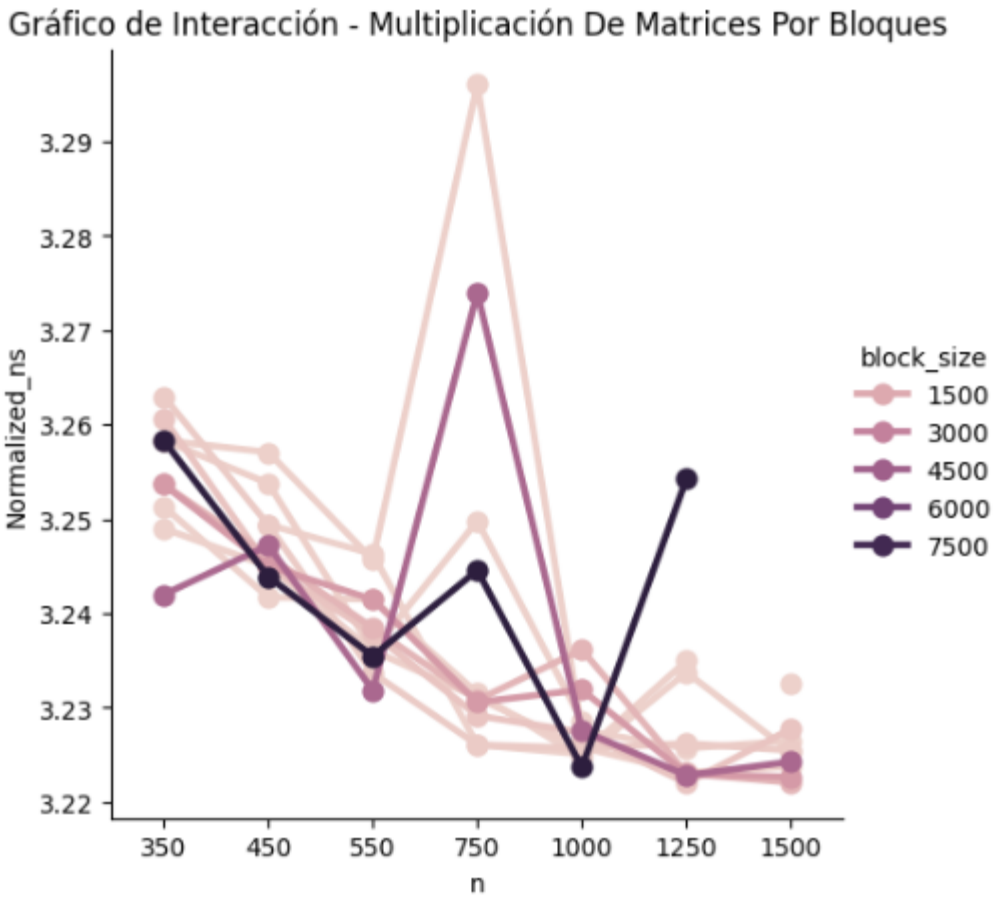

Fig. Gráfico del impacto del tamaño de bloque en el rendimiento del algoritmo que utiliza la estrategia de multiplicación por bloques.

Nota: Los datos del ANOVA y el gráfico se encuentran en el cuaderno de Colab denominado "ANOVA_Multiplication_By_Blocks.ipynb". Este cuaderno utiliza los datos extraídos del archivo Excel denominado "ReporteMultiplicaciónPorBloques" y de la hoja "MultiplicaciónPorBloques".

Como se puede observar, el valor p para la variable 'block_size' es mayor que el alfa, que es de 0.05. Por lo tanto, se acepta la hipótesis nula (H0), lo que indica que el factor 'block_size' no tiene un efecto significativo sobre 'Normalized_ns' en el algoritmo de multiplicación de matrices por bloques. Por otra parte, se observa que para la interacción entre 'C(n):C(block_size)', sí se rechaza la hipótesis nula, ya que el valor p es mucho menor que el alfa. Esto sugiere que 'block_size' interactúa con el tamaño 'n' de la matriz elegida. Esto tiene sentido, ya que un

tamaño de matriz más grande puede manejarse mejor a tamaños de bloques más pequeños. Cuando el tamaño de la matriz es mayor, el uso de tamaños de bloque pequeños que permitan una compactación eficiente de la información en los tamaños de bloque utilizados por la memoria caché conduce a un mejor rendimiento en términos de aprovechamiento de la memoria y reducción de los accesos a la memoria principal.

Por otra parte, es importante destacar que este algoritmo con la estrategia de multiplicación por bloques sí mejoró el rendimiento del algoritmo 5. Esto es evidente, ya que el tiempo normalizado, que es nuestra variable de comparación, es considerablemente menor para los tamaños grandes que en el algoritmo 5. Por ejemplo, estos son algunos de los tiempos normalizados para el algoritmo 5 obtenidos por el grupo 05:

version	type	sample	n	time(s)	Normalized(ns)
ver5	f	5	350	0,175	4,0816
ver5	f	5	450	0,387	4,2469
ver5	f	5	550	0,71	4,2675
ver5	f	5	750	2,86	6,7793
ver5	f	4	1000	11,227	11,227
ver5	f	8	1250	24,266	12,4242
ver5	f	6	1500	46,794	13,8649

Tabla. Tiempos de ejecución del algoritmo 5.

En cada caso, observamos que el tiempo normalizado es mayor a 4 nanosegundos, y aumenta a medida que se incrementa el tamaño de las matrices n. En contraste, como se muestra en la *gráfica del impacto del tamaño de bloque en el rendimiento del algoritmo que utiliza la estrategia de multiplicación por bloques*, el tiempo normalizado para el algoritmo de multiplicación por bloques se mantiene estable entre 3 y 4 nanosegundos. Esto sugiere que esta estrategia favorece la reducción de fallos de caché y hace que la operación de multiplicación de matrices concluya más rápidamente, a pesar de utilizar la misma forma de recorrido que el algoritmo 5.

CONCLUSIONES

El presente estudio ha evaluado el impacto de la localidad de caché en el rendimiento de la multiplicación de matrices, considerando seis versiones distintas de algoritmos. Los resultados obtenidos evidencian la importancia crítica de la localidad de caché para optimizar el

rendimiento. Algoritmos diseñados para aprovechar la localidad espacial y temporal, como el algoritmo de Bloques, muestran menores tasas de fallos de caché (Miss Rate) y, por consiguiente, un menor Tiempo Medio de Acceso a Memoria (AMAT). En contraste, versiones de algoritmos que no maximizan la utilización de la caché, como la versión 5 kij, experimentan un aumento significativo en el tiempo de ejecución a medida que incrementa el tamaño de la matriz.

El tamaño de la matriz es otro factor determinante en el rendimiento, ya que influye directamente en la capacidad de las cachés L1, L2 y L3 para almacenar datos. Para matrices pequeñas ($n < 550$), los datos se almacenan sin problema en la caché L1, beneficiándose de su alta velocidad de acceso. Sin embargo, a medida que las matrices crecen ($550 \leq n < 1000$), los datos comienzan a almacenarse en la caché L2, lo que resulta en tiempos de acceso mayores. Este incremento es aún más pronunciado para matrices grandes ($n \geq 1000$), donde los datos exceden L2 y se empiezan a guardar en la memoria L3, llevando a un aumento significativo en el tiempo de ejecución.

La interacción entre la versión del algoritmo y el tamaño de las matrices es particularmente significativa. Ciertas combinaciones de algoritmos y tamaños de matrices demuestran ser más eficientes en términos de jerarquía de memoria. Por ejemplo, las versiones 1 ijk y 3 jik mantienen un rendimiento más constante hasta cierto punto, mientras que la versión 5 kij muestra una degradación notable en el rendimiento conforme aumenta el tamaño de la matriz. Este comportamiento se explica por los patrones de acceso a la memoria que afectan la localidad espacial y temporal de cada versión del algoritmo.

El tipo de datos también puede influir en la capacidad de las cachés, ya que los datos de tipo double ocupan más espacio, alcanzando los límites de la caché más rápidamente. Sin embargo, el análisis ANOVA no muestra una significancia estadística del tipo de datos en el rendimiento, posiblemente debido a la escala de los experimentos realizados. Esto sugiere que, en el contexto

específico de este estudio, las diferencias en el tamaño de los datos no son lo suficientemente grandes como para afectar notablemente el rendimiento.

Para mitigar el impacto negativo de la localidad de caché, se implementó la multiplicación por bloques en el algoritmo con peor rendimiento (kij). Esta optimización mostró mejoras significativas, dado que los bloques más pequeños posibilitan una mejor utilización de la caché, incluso para matrices más grandes. Esto se debe a que distribuyen la información necesaria para las operaciones de multiplicación de matrices en la memoria caché, lo que permite aprovechar la localidad espacial y temporal de los datos en la jerarquía de memoria.

Finalmente, es importante mencionar que técnicas adicionales de optimización, como la arquitectura superscalar, la ejecución fuera de orden, la ejecución especulativa y los procesadores de emisión múltiple, pueden contribuir a mejorar aún más el rendimiento de la multiplicación de matrices. Estas técnicas explotan el paralelismo a nivel de instrucción (ILP), permitiendo una ejecución más eficiente y rápida de los algoritmos.

En resumen, la optimización de los algoritmos de multiplicación de matrices para mejorar la localidad de caché, junto con el uso de técnicas avanzadas de procesamiento, puede reducir significativamente el AMAT, mejorando el rendimiento global de estos algoritmos. Los hallazgos de este estudio proporcionan una base sólida para futuras investigaciones y optimizaciones en el ámbito de la computación de alto rendimiento.

REFERENCIAS

Randal E. Bryant, David R. O'Hallaron (2015) . Computer Systems: A Programmer's Perspective, Edición 3er edición, Pearson, ISBN: 978-0134092669

Sand Software Sound. (n.d.). Memory hierarchy and access time. Recuperado de http://sandsoftwaresound.net/raspberry-pi/raspberry-pi-gen-1/memory-hierarchy/

Montgomery, D. C. (2019). Design and analysis of experiments (10.ª ed.). John Wiley & Sons.