# Informe semana 8: Concebir y planificación de un Experimento de 4 factores 🡪Proyecto Final

**1. Introducción**

En este laboratorio, se diseñará un experimento de **cuatro factores** que incluirá diferentes versiones de los algoritmos, tamaño de matrices, tipo de dato y lenguaje de programación.

**Objetivos específicos:**

* Comparar los tiempos de ejecución de los 9 tratamientos y analizar su relación con la localidad espacial.
* Identificar y analizar datos atípicos en los resultados de ejecución.
* Diseñar un experimento con cuatro factores para evaluar rendimiento computacional.

## Diseño de Experimento de Cuatro Factores

## En esta sección, usted debe concebir un experimento factorial con el objetivo de evaluar el impacto de diversos factores en el rendimiento de la multiplicación de matrices. Se busca analizar cómo influyen la versión del algoritmo, el tamaño de la matriz, el tipo de dato y el lenguaje de programación en el tiempo de ejecución y la eficiencia computacional.

## 2.1 Definición del Experimento

El experimento del proyecto final se estructurará considerando **cuatro factores** con los siguientes niveles:

* **Factor 1: Versión del Algoritmo (6 niveles):** a, b, c, d, e, f  
  Se seleccionan seis versiones diferentes del algoritmo de multiplicación de matrices, cada una con una estrategia distinta de recorrido y acceso a memoria.
* **Factor 2: Tamaño del Arreglo N (Definir Niveles):**  
  Los tamaños de matriz serán definidos con base en la jerarquía de memoria caché del sistema. Se deberán establecer mínimo 12 niveles que permitan evaluar el impacto de la jerarquía de memoria en el desempeño. El sistema dónde se ejecutará el experimento será un computador de uno de los estudiantes.
* **Factor 3: Tipo de Dato (2 niveles):** float y double  
  Se analizará cómo el uso de diferentes tipos de datos afecta la precisión numérica y el tiempo de ejecución.
* **Factor 4: Lenguaje de Programación (3 niveles):** C++, Python, Java  
  Se evaluará el impacto del lenguaje de programación en el rendimiento computacional, teniendo en cuenta aspectos como la optimización de compiladores y la administración de memoria.

### 2.2 Matriz de Diseño Factorial

Para llevar a cabo el experimento, se debe construir una **matriz de diseño factorial completo** que contemple todas las combinaciones posibles de los niveles de los factores.

Se debe:

1. Justificar la elección de los niveles para cada factor, especialmente el factor tamaño del arreglo.
2. Diseñar la matriz de diseño factorial.
3. Generar una Matriz aleatorizada del diseño para evitar sesgos en la ejecución del experimento.

Para garantizar la precisión y confiabilidad de los resultados, debe implementar un esquema de repeticiones y réplicas en el experimento, ¿cuántas planea hacer?

**Entrega:** Presentar la matriz de diseño en una tabla bien estructurada, especificando los niveles seleccionados y la justificación de cada uno. Recuerde incluir la columna Order estándar y otra de orden de ejecución (aleatorizado).

### 2.3 Planificación de ejecución del Experimento

### Con lo que hemos aprendido y experimentado en estas primeras semanas en el laboratorio, ya somos conscientes que una buena planificación es importante para garantizar la validez y confiabilidad de los resultados. Se deben considerar los siguientes aspectos entre otros:

### 2.3.1 Preguntas Clave para la Planificación

### Antes de la ejecución, se recomienda responder las siguientes preguntas para definir adecuadamente los parámetros del experimento:

### Selección de Niveles:

### ¿Cómo se han definido los niveles del tamaño de la matriz N?

**R//:** Los niveles del tamaño de la matriz se han definido en función de la jerarquía de caché del procesador (L1, L2, L3) y su capacidad, como se observa en las tablas proporcionadas. Esto permite evaluar el impacto del tamaño de la matriz en el rendimiento del algoritmo. Además, se ha seguido la siguiente fórmula para definir los niveles:

El numerador será el valor en KB de cada nivel del caché multiplicado por el valor de un KB en bytes (1024), y el denominador será el número de matrices que usaremos por el tamaño en bytes del tipo de los datos que estarán dentro de nuestra matriz (por ejemplo, 4 bytes para float).

### ¿La selección de los lenguajes de programación propuestos, son válidos para usted?

**R//:** Sí, los lenguajes seleccionados (C++, Python y Java) son válidos, ya que representan diferentes paradigmas y niveles de abstracción, lo que permite analizar cómo cada lenguaje interactúa con la jerarquía de memoria. C++ es un lenguaje de bajo nivel que permite un control fino sobre la gestión de memoria y optimizaciones específicas, como el uso directo de punteros y la alineación de datos para aprovechar la caché. Python, es un lenguaje de alto nivel con un modelo interpretado y una gestión de memoria automática (a través del recolector de basura), lo que puede introducir variabilidad en el acceso a la memoria y afectar la localidad espacial. Java, con su máquina virtual (JVM) y optimizaciones mediante el compilador JIT (Just-In-Time), ofrece un punto intermedio, combinando abstracción con cierto grado de optimización automática.

### ¿Tiene alguna validez técnica comparar dichos lenguajes?

**R//:** Sí, comparar C++, Python y Java es técnicamente válido, ya que cada uno tiene modelos distintos de gestión de memoria y optimización (C++ con control manual, Python con overhead interpretado, Java con JVM y JIT), lo que afecta el uso de la caché y el tiempo de ejecución. Esto permite evaluar su impacto en el rendimiento computacional de forma justa al estandarizar las implementaciones.

### Control de Factores Secundarios:

### ¿Existen factores externos que puedan afectar la medición del tiempo de ejecución?

### R//: Sí existen. Algunos son:

* Procesos en segundo plano en el sistema operativo: Las aplicaciones o servicios ejecutándose simultáneamente (como actualizaciones automáticas, antivirus o procesos del sistema) pueden competir por recursos de CPU y memoria, afectando la consistencia de los tiempos medidos.
* Variaciones en la temperatura del procesador (thermal throttling): Un aumento en la temperatura del CPU debido a cargas prolongadas podría activar mecanismos de reducción de frecuencia (throttling) para evitar sobrecalentamiento, lo que alteraría el rendimiento y, por ende, los tiempos de ejecución.
* Diferencias en la implementación del compilador o intérprete: Las versiones o configuraciones específicas de los compiladores (e.g., GCC para C++, HotSpot para Java) o intérpretes (e.g., CPython para Python) pueden introducir variaciones en las optimizaciones aplicadas, afectando el acceso a la memoria y la velocidad de ejecución.
* Carga de la memoria caché por otros procesos: Si otros programas acceden a la memoria principal o caché antes o durante las corridas, podrían desplazar datos relevantes del experimento, reduciendo la localidad espacial y aumentando los tiempos de ejecución.
* Fluctuaciones en el voltaje o la fuente de energía: En sistemas no estabilizados, variaciones en el suministro eléctrico podrían influir en el rendimiento del procesador, especialmente en computadoras portátiles sin batería completamente cargada.

### ¿Cómo se garantizará que el entorno de ejecución sea constante?

* Ejecutar los experimentos en un entorno aislado: Se realizarán las pruebas en modo consola o en un sistema operativo configurado en modo mínimo (e.g., arrancando en modo seguro con solo servicios esenciales), cerrando todas las aplicaciones no necesarias y desactivando procesos en segundo plano como actualizaciones automáticas o notificaciones del sistema. Esto reduce la competencia por recursos de CPU y memoria.
* Fijar la frecuencia del procesador: Se configurará el modo de energía del computador en "Alto rendimiento" y, si es posible, se deshabilitarán tecnologías de ajuste dinámico como Intel SpeedStep o AMD Cool’n’Quiet, utilizando herramientas del sistema operativo o la BIOS. Esto asegura que la frecuencia del CPU permanezca constante durante todas las corridas, evitando fluctuaciones por throttling o políticas de ahorro de energía.
* Usar el mismo compilador o versión del intérprete para cada lenguaje: Se estandarizarán las herramientas de compilación e interpretación, por ejemplo, utilizando GCC 12.x para C++, CPython 3.11 para Python y OpenJDK 17 para Java, con configuraciones idénticas (e.g., nivel de optimización -O2 para C++). Esto garantiza que las diferencias en los tiempos de ejecución provengan de las características del lenguaje y no de variaciones en las herramientas.
* Control de temperatura: De ser posible se ejecutarán las pruebas en un ambiente con ventilación adecuada y temperatura estable, evitando picos térmicos que activen throttling. Si es necesario, se pausarán las corridas entre bloques para estabilizar las condiciones térmicas.

### Repeticiones y Aleatorización:

### ¿Cuántas repeticiones se realizarán por cada combinación para reducir la variabilidad?

**R//:** Se realizarán 10 repeticiones distribuidas aleatoriamente por cada combinación de factores tal como se hizo en clase, esto para reducir la variabilidad en las mediciones del tiempo de ejecución, asegurando estimaciones confiables y suficiente potencia estadística para análisis como ANOVA, según los principios del capítulo introductorio (sección 1.15). El **número de tratamientos** por lenguaje se calcula como (versiones del algoritmo) × (tamaños de matriz) × (tipos de dato) = 6×12×2=144, ignorando los niveles del lenguaje, ya que cada lenguaje (C++, Python, Java) se evalúa por separado con la misma cantidad de tratamientos (144 tratamientos), dando 144 × 10 = 1440 corridas por lenguaje y 1440 × 3 = 4320 corridas totales, un número viable que equilibra precisión y esfuerzo práctico.

### ¿Va a realizar repeticiones o réplicas?

**R//:** El experimento realizará repeticiones, no réplicas, ya que se ejecutarán 10 corridas por cada combinación de factores bajo las mismas condiciones experimentales (mismo hardware y configuración), con el objetivo de reducir la variabilidad en los tiempos de ejecución y estimar el error experimental, lo que coincide con la definición de repetición dada en el Capítulo 1 DOE Terminología y Conceptos v1.3.4: Cuando se hacen unidades consecutivas sin cambiar los niveles de las variables de diseño entre unidades, estas unidades similares se llaman repeticiones. No se realizarán réplicas ya que las réplicas implicarían repetir el experimento completo en condiciones diferentes (e.g., otro equipo o día), según la distinción establecida en la sección 1.15 del capítulo introductorio.

### ¿Cómo se distribuirán aleatoriamente los tratamientos para evitar sesgos?

**R//:** Se generará un número aleatorio único para cada combinación de factores (1 a 144) utilizando un algoritmo de permutación aleatoria (en este caso se usó la función de generación de números aleatorios de Python random.shuffle) y se ejecutarán en ese orden aleatorio. Se generará un número aleatorio único para cada combinación de factores (1 a 144)

### Formato de Registro de Datos:

### ¿Cómo se almacenarán los tiempos de ejecución para cada tratamiento (combinación de factores)?

**R//:** Los tiempos de ejecución se almacenarán en un archivo CSV o base de datos (en este caso en excel), con columnas para cada factor (tipo de algoritmo, tamaño de matriz, tipo de dato, lenguaje), el tiempo medido y el tiempo normalizado con el cuál se generarán los reportes.

### ¿Se establecerán criterios para identificar valores atípicos?

**R//:** Sí, se establecerán criterios para identificar valores atípicos en los tiempos de ejecución usando métodos como el del rango intercuartílico (IQR): se calcularán los cuartiles Q1 y Q3 de las 10 repeticiones por tratamiento, considerando atípico cualquier valor por debajo de Q1−1.5×IQR o por encima de Q3+1.5×IQR, donde IQR=Q3−Q1; estos valores se analizarán para descartar errores experimentales (como interrupciones del sistema) y garantizar que el análisis posterior con pruebas estadísticas como ANOVA sea robusto, siguiendo las prácticas de manejo de datos de la sección 1.17 del capítulo introductorio.

### 2.3.2 Control de Factores Secundarios

### Es importante minimizar el impacto de factores secundario que puedan influir en el desempeño de los algoritmos.

### Indique al menos 4 estrategias que ha aprendido en clase o que se imagina debería seguir para minimizar la variabilidad de los factores.

### R//: Los factores secundarios son variables externas no controladas directamente en el experimento, como procesos del sistema o temperatura del procesador, que pueden influir en el desempeño de los algoritmos y aumentar la variabilidad en los tiempos de ejecución, algunas estrategias que hemos aprendido a fin de minimizar la variabilidad de estos factores son:

### Ejecutar en modo terminal sin otras tareas para aislar el entorno

### Desactivar procesos en segundo plano innecesarios para liberar recursos

### Fijar el modo de energía en "Alto rendimiento" y desactivar ajustes dinámicos de frecuencia del procesador

### Realizar 10 repeticiones por tratamiento y promediar los resultados para reducir el efecto de mediciones atípicas

### Monitorear la temperatura del procesador, pausando si hay picos térmicos que puedan causar throttling.

### 2.4 Desarrollo de Versiones Equivalentes del Código en Distintos Lenguajes

### Como parte del experimento, es necesario que cada grupo cree las versiones del programa de multiplicación de matrices en los tres lenguajes de programación definidos: C++, Python y Java.

### Requisitos:

### El código en los tres lenguajes debe ser lo más equivalente posible, respetando la lógica del algoritmo y las estructuras utilizadas.

### Se deben implementar las seis versiones del algoritmo (a, b, c, d, e, f), que varían en el orden de los bucles y la forma de acceso a memoria.

### Importante:

### La versión más reciente del código C++ utiliza la librería chrono para medir tiempos de ejecución con mayor precisión.

### Se recomienda seguir este enfoque en las versiones de Python y Java, utilizando funciones de temporización de alta resolución en Python y en Java.

### Entrega: En el informe, cada grupo debe describir cómo se implementaron las versiones en los tres lenguajes.

### Y además explicar cómo planean ejecutar los experimentos en cada uno de ellos, en qué entorno de ejecución, teniendo muy en cuenta que el objetivo es evaluar el impacto de la cache del computador de uno de los miembros del grupo.

**Código en Python**

### En Python, no hay distinción real entre float y double (ambos son float de 64 bits), por lo que se tiene un único archivo .py, pero se añadió un argumento data\_type para elegir el tipo de dato, documentar la intención y mantener equivalencia con C++ y Java.

### Descripción de cómo se implementaron las versiones en Python

### La implementación en Python se llevó a cabo adaptando el código original en C++ para preservar la lógica y las estructuras del algoritmo de multiplicación de matrices, incluyendo las seis versiones (product\_mat\_a a product\_mat\_f) con sus respectivos órdenes de bucles (ijk, jik, jki, kji, kij, ikj) y patrones de acceso a memoria. En lugar de punteros y arreglos dinámicos, se utilizaron arreglos unidimensionales de numpy para representar las matrices en formato column-major, con índices calculados como i + j \* n, manteniendo la equivalencia con C++. Para soportar tanto float (32 bits) como double (64 bits), se emplearon los tipos np.float32 y np.float64, ya que el float nativo de Python es de 64 bits; las matrices A, B y C se inicializan con valores fijos de 2.0, 4.0 y 0.0, respectivamente, usando np.full y np.zeros según el tipo de dato seleccionado. La medición del tiempo se implementó con time.perf\_counter(), que proporciona alta resolución comparable a std::chrono en C++, reemplazando el clock() original. Los argumentos de línea de comandos se gestionan mediante sys.argv, y la salida se escribe en un archivo de texto (ReportS2\_LHW00.txt) utilizando open en modo append, con una columna que distingue entre float y double, asegurando que el código sea funcionalmente equivalente al de C++ y permita comparaciones directas.

### Explicación de cómo planeamos ejecutar los experimentos en Python

### Los experimentos en Python se ejecutarán en Visual Studio 2022, utilizando el soporte integrado para Python, en el computador de un miembro del grupo cuya caché será evaluada. El script se lanzará desde la terminal integrada con comandos como python script.py <n> <samples>, ejecutando cada una de las seis versiones del algoritmo (product\_mat\_a a product\_mat\_f) para ambos tipos de dato (float32 y float64), modificando manualmente la función llamada en el código entre corridas. Para garantizar mediciones consistentes y reflejar el impacto de la caché, se aislará el entorno cerrando aplicaciones innecesarias y se configurará el plan de energía del sistema en "Alto rendimiento" para fijar la frecuencia del procesador. Se probarán diferentes tamaños de matriz (n, e.g., 64, 128, 256, 512) y un número fijo de repeticiones (samples, e.g., 5), registrando los tiempos en el archivo de salida para analizar cómo el orden de los bucles y el tipo de dato afectan la localidad espacial y el rendimiento de la caché en la arquitectura x64 del equipo seleccionado.

### Código en Java

**Descripción de cómo se implementaron las versiones en Java**

La implementación en Java se realizó traduciendo la lógica del código C++ en dos archivos separados: MatrixProductFloat.java para float (32 bits) y MatrixProductDouble.java para double (64 bits), preservando las seis versiones del algoritmo (productMatA a productMatF) con sus órdenes de bucles (ijk, jik, jki, kji, kij, ikj) y accesos a memoria en formato column-major, utilizando arreglos unidimensionales con índices i + j \* n. Las matrices A, B y C se inicializan con valores 2.0, 4.0 y 0.0 mediante Arrays.fill, especificando 2.0f para float en el primer archivo y 2.0 para double en el segundo, asegurando compatibilidad con los tipos de dato nativos de Java. La medición del tiempo se implementó con System.nanoTime(), ofreciendo alta precisión similar a std::chrono en C++, en lugar de clock() del original. Los argumentos se leen desde args, y la salida se escribe en ReportS2\_LHW00.txt con FileWriter en modo append, incluyendo una columna typeData que indica float o double según el archivo ejecutado. La gestión de memoria es automática en Java, eliminando la necesidad de malloc y free, garantizando equivalencia funcional con el código C++.

**Explicación de cómo planeamos ejecutar los experimentos en Java**

Los experimentos en Java se ejecutarán en Visual Studio 2022 con soporte para Java (o en un IDE como IntelliJ IDEA si es necesario), en el computador de un miembro del grupo cuya caché será evaluada. Cada archivo (MatrixProductFloat.java y MatrixProductDouble.java) se compilará y ejecutará por separado desde la terminal integrada con comandos como java MatrixProductFloat <n> <samples> y java MatrixProductDouble <n> <samples> para cada una de las seis versiones del algoritmo, modificando manualmente la función llamada en el código (productMatA a productMatF) entre corridas. Para aislar el impacto de la caché, se cerrarán aplicaciones innecesarias y se configurará el sistema en modo "Alto rendimiento" para estabilizar la frecuencia del procesador, probando diferentes tamaños de matriz (n, e.g., 64, 128, 256, 512) y un número fijo de repeticiones (samples, e.g., 5). Los resultados se registrarán en el archivo de salida, permitiendo analizar cómo el orden de los bucles y el tipo de dato (float o double) afectan la localidad espacial y el rendimiento de la caché en la arquitectura x64 del equipo.

### Código en C++

**Descripción de cómo se implementaron las versiones en C++**

La implementación en C++ para double se desarrolló modificando el código original, que usaba float, reemplazando todas las instancias de float por double en las definiciones de las seis funciones (ProductMat\_a a ProductMat\_f), así como en las variables locales y los punteros de las matrices A, B y C, preservando los órdenes de bucles (ijk, jik, jki, kji, kij, ikj) y los accesos a memoria en formato column-major con índices i + j \* n. Las matrices se inicializan dinámicamente con malloc usando sizeof(double) para reflejar el mayor tamaño del tipo de dato (64 bits), y los valores iniciales se mantienen en 2.0, 4.0 y 0.0 sin necesidad de conversión explícita, ya que double los soporta nativamente. La medición del tiempo se mantuvo con clock(), aunque cumple con el estándar básico del código original (nota: podría actualizarse a chrono para mayor precisión como se sugirió, pero se conservó la estructura inicial para consistencia). La salida se ajustó para indicar double en la columna typeData del archivo ReportS2\_LHW00.txt, utilizando fprintf\_s y printf, asegurando que la implementación sea equivalente a la versión con float pero con mayor precisión numérica.

**Explicación de cómo planeamos ejecutar los experimentos en C++**

Los experimentos en C++ se ejecutarán en Visual Studio 2022 en el computador de un miembro del grupo, compilando y ejecutando el programa desde la terminal integrada con comandos como <programa>.exe <n> <samples> para cada una de las seis versiones del algoritmo (ProductMat\_a a ProductMat\_f), ajustando manualmente la función invocada en el código fuente entre corridas. Para garantizar que las mediciones reflejen el impacto de la caché, se minimizarán las aplicaciones en ejecución y se configurará el sistema en modo "Alto rendimiento" para fijar la frecuencia del procesador, utilizando diferentes tamaños de matriz (n, e.g., 64, 128, 256, 512) y un número fijo de repeticiones (samples, e.g., 5). Los resultados se registrarán en ReportS2\_LHW00.txt, permitiendo analizar cómo el orden de los bucles y el uso de double afectan la localidad espacial y el rendimiento de la caché en la arquitectura x64 del equipo seleccionado.

### 2.5 Automatización de ejecuciones.

### Dada la gran cantidad de combinaciones de factores que se evaluarán en este experimento (432 tratamientos únicos, cada uno con 10 repeticiones, sumando un total de 4320 ejecuciones), la automatización del proceso es fundamental para garantizar la eficiencia, reducir la posibilidad de errores manuales y asegurar la reproducibilidad de los resultados.

### ¿Cómo están ejecutando múltiples combinaciones de factores?

### Para gestionar la ejecución de las múltiples combinaciones de factores, se ha optado por un enfoque centralizado y automatizado. En lugar de modificar manualmente el código fuente para cada variación o ejecutar cada combinación de forma aislada, el sistema se ha diseñado para ser controlado por parámetros externos.

### Desarrollo de un Script Maestro y Parametrización de los Programas:

### Se realiza un Script Maestro (preferentemente en Python, debido a su flexibilidad para la manipulación de archivos, ejecución de procesos y manejo de cadenas de texto) que se encarga de orquestar la ejecución de cada una de las versiones de cada algoritmo bajo todas las condiciones experimentales definidas.

### El sistema de automatización opera mediante dos componentes clave. Primero, los programas de multiplicación de matrices en C++, Python y Java se han parametrizado para aceptar argumentos por línea de comandos, tales como la versión del algoritmo, el tamaño de la matriz (N), el número de repeticiones y, opcionalmente, el tipo de dato. Esta modificación permite que cada programa configure internamente la ejecución específica solicitada, realice las mediciones de tiempo correspondientes y emita el resultado (tiempo de ejecución) de manera estandarizada para su fácil captura.

### El segundo componente es un Script Maestro que orquesta todo el experimento. Este script lee una matriz de diseño experimental aleatorizada, que detalla todas las combinaciones de factores. Luego, para cada tratamiento, construye dinámicamente el comando necesario para ejecutar el programa pertinente (C++, Python o Java) con los parámetros adecuados. Finalmente, el script ejecuta estos comandos, captura los tiempos de ejecución resultantes y los registra de forma estructurada en un archivo global, consolidando todas las observaciones del experimento.

### 

### 2.6 Matriz de diseño experimental.

### La matriz de diseño experimental completa que fundamenta este estudio es una representación tabular de todas las combinaciones de factores que se evaluarán, incluyendo las repeticiones planificadas y el orden aleatorio de ejecución.

### Esta matriz es el resultado del diseño factorial completo 6×N×2×3, donde N representa los niveles específicos definidos para el tamaño de la matriz. Dada la definición de un mínimo de 12 niveles para N, la matriz base contiene al menos 6×12×2×3=432 combinaciones únicas de los cuatro factores (Versión del Algoritmo, Tamaño N, Tipo de Dato, Lenguaje de Programación).

### Para cada una de estas combinaciones únicas, se han planificado 10 repeticiones, lo que resulta en una matriz final con un mínimo de 432×10=4320 filas, cada fila representando una ejecución individual del experimento.

### La matriz incluirá las siguientes columnas principales:

### Order Standard: Un identificador secuencial para cada una de las combinaciones únicas de factores (1 a ≥432).

### Algoritmo: Especifica la versión del algoritmo de multiplicación de matrices (a, b, c, d, e, f).

### Tamaño N: Indica el tamaño de la matriz cuadrada N×N. Esta columna contendrá los ≥12 valores numéricos específicos definidos para el experimento, justificados en función de la jerarquía de caché del sistema objetivo.Tipo Dato: Detalla el tipo de dato utilizado en las matrices (float o double).

### Lenguaje: Identifica el lenguaje de programación de la implementación (C++, Python, o Java).

### Repeticion: Indica el número de repetición para esa combinación de factores específica (1 a 10).

### Order Ejecucion: Un número único (1 a ≥4320) que define el orden completamente aleatorio en el que se llevarán a cabo las ejecuciones experimentales para mitigar sesgos.

### 

### 