Informe de Prácticas

Sesión 1

Computación de Altas Prestaciones

Máster en Ingeniería Informática

Universidad de Oviedo

Autor:

Rubén Martínez Ginzo, UO282651@uniovi.es

Abril 2025

Índice

Índice de figuras

Índice de tablas

1. Introducción

En esta sesión de prácticas de laboratorio se evalúan distintos esquemas de acceso a memoria aplicados a la multiplicación de matrices. Concretamente, se analiza el rendimiento de esta operación matemática utilizando tres estrategias de acceso diferentes:

- *Row-Major order*
- *Column-Major order*
- *Z-Order*

El objetivo principal es determinar cuál de estas estrategias ofrece un mejor desempeño en términos de tiempo de ejecución y eficiencia en el uso de memoria.

1.1. Desarrollo

Para el desarrollo de esta práctica, se han seguido las indicaciones recogidas en el guion de la sesión correspondiente. Se han implementado las tres estrategias de acceso en el código fuente, y se han llevado a cabo diversas pruebas para medir su rendimiento. Con el fin de comparar los tiempos de ejecución, se ha desarrollado un programa en *C* con fines de *benchmarking*, tal y como se detalla en el capítulo siguiente. Todo el código fuente se encuentra disponible de forma pública en el siguiente repositorio de GitHub, así como en el archivo *zip* asociado a esta entrega.

1.2. *Benchmarking*

Además del trabajo realizado durante la sesión de laboratorio, se ha desarrollado un programa en *C* que automatiza la medición de los tiempos de ejecución para las distintas estrategias de acceso a memoria.

Este programa, ubicado en el subdirectorio *benchmark*, permite especificar diferentes tamaños de matriz y un número fijo de iteraciones para cada experimento. Durante la ejecución, se realiza la multiplicación de matrices utilizando cada uno de los esquemas de acceso, registrando el tiempo requerido por cada estrategia. En el caso particular del algoritmo *zorder*, se evalúan automáticamente todos los tamaños de bloque posibles para cada tamaño de matriz, llevando a cabo la multiplicación con cada configuración. Cada multiplicación se repite tantas veces como iteraciones se hayan especificado, calculando el tiempo medio de ejecución de todas ellas. Esto permite reducir significativamente el ruido en las mediciones y obtener resultados más precisos.

Al finalizar, el programa devuelve un *log* en formato CSV con los resultados otenidos. Dichos tiempos de ejecución constituyen la base de los análisis y comparativas que se presentan en los capítulos siguientes.

2. Expectativas y análisis inicial

Antes de analizar los resultados obtenidos, se comentan las expectativas iniciales respecto al rendimiento de cada uno de los esquemas de acceso a memoria aplicados al producto de matrices.

2.1. *Row-Major order*

El esquema *Row-Major order* se basa la lectura de matrices por filas, lo que, traspolado al producto de matrices ($C = A \times B$) implica que la matriz resultado **C** se recorrerá de dicha forma. Esto es, para realizar el cálculo de cada elemento de la primera fila de la matriz resultado, se recorrerá la primera file de la matriz **A** y todas las columnas de la matriz **B**. Este proceso se repetirá para cada una de las filas de la matriz **C**.
Se espera que este esquema de acceso a memoria sea muy eficiente, ya que minimiza la cantidad de fallos en caché. Concretamente, para el cálculo de cada fila de la matriz **C**, se espera que se produzcan:

- 1 único fallo de caché para la matriz **C**, al acceder al primer elemento de su primera fila.
- 1 único fallo de caché para la matriz **A**, al acceder al primer elemento de su primera fila.
- $N \times N$ fallos de caché para la matriz **B**, ya que se accede columna a columna y cada columna tiene N elementos.

2.2. *Column-Major order*

El esquema *Column-Major order* se basa la lectura de matrices por columnas, lo que, traspolado al producto de matrices ($C = A \times B$) implica que la matriz resultado **C** se recorrerá de dicha forma. Esto es, para realizar el cálculo de cada elemento de la primera columna de la matriz resultado, se recorrerá la primera columna de la matriz **B** y todas las filas de la matriz **A**. Este proceso se repetirá para cada una de las columnas de la matriz **C**.
Se espera que este esquema de acceso a memoria sea menos eficiente que el anterior, ya que se producen más fallos de caché. Concretamente, para el cálculo de cada columna de la matriz **C**, se espera que se produzcan:

- N fallos de caché para la matriz **C**, ya que se accede a N elementos de su primera columna.
- N fallos de caché para la matriz **A**, ya que se accede a N veces a cada una de sus filas.
- $N \times N$ fallos de caché para la matriz **B**, ya que se accede N veces a los N elementos de su primera columna.

2.3. *Z-Order*

El esquema *Z-Order* se basa en la división recursiva de las matrices en submatrices de tamaño $2^n \times 2^n$ y su posterior recorrido en orden Z. Traspolado al producto de matrices ($C = A \times B$), implica que la matriz resultado **C** se recorrerá en orden Z para cada una de las submatrices en las que se haya dividido.
Este esquema permite una mejor utilización de la memoria caché en casos donde las matrices a multiplicar son de grandes dimensiones, ya que se accede a los elementos de forma más local. Sin embargo, su rendimiento puede verse afectado por el tamaño de bloque utilizado. Se espera, por tanto, que sea el esquema más eficiente en casos de matrices de grandes dimensiones. Predecir empíricamente el número de fallos de caché en el uso de este esquema es algo muy complejo, por lo que su estudio se limita a la próxima experimentación y comparación de resultados.

3. *Row-Major order* vs *Column-Major order*

En esta sección se comparan los dos métodos de recorrido de matrices, *Row-Major order* y *Column-Major order*, en cuanto a su rendimiento en el producto matricial. Para ello, se ha realizado dicha operación para los siguientes tamaños de matrices: {2, 4, 8, 16, 32, 64, 128, 256, 398, 512, 636, 774, 892, 1024, 1152, 1280, 1408, 1536}. Cada producto se ha realizado un total de 32 veces y se ha medido el tiempo medio de ejecución. Los resultados obtenidos se presentan en la Figura 1.

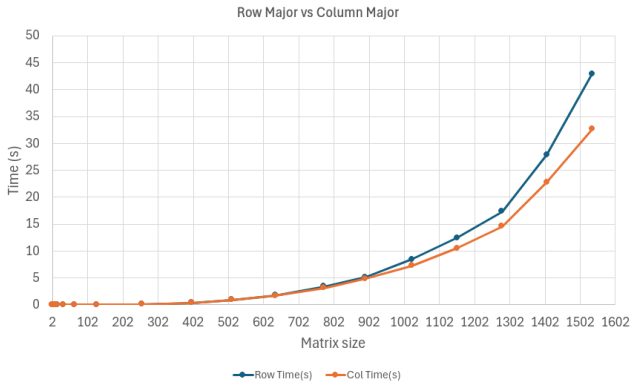

Figura 1: Comparación de tiempos de ejecución para *Row-Major order* y *Column-Major order*

Sorprendentemente, el recorrido por columnas es más rápido que el recorrido por filas. Esta diferencia comienza a ser notable a partir de un tamaño de matriz relativamente grande, aproximadamente de 1024×1024. Dicha diferencia crece a medida que aumenta el tamaño de la matriz, alcanzando unos $10s$ de diferencia en el mayor tamaño de matriz probado (1536×1536). Para representar la diferencia de rendimiento entre ambos métodos, se ha calculado la aceleración de *Row-Major order* respecto a *Column-Major order* para cada tamaño de matriz. Dicha aceleración se ha graficado en la Figura 2.

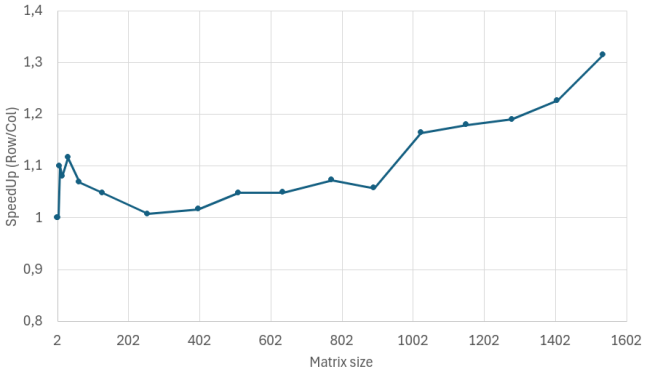

Figura 2: Aceleración de *Row-Major order* respecto a *Column-Major order*

Como puede observarse, *Column-Major order* es más rápido que *Row-Major order* para todos los tamaños de matriz probados, siendo notablemente mejor conforme aumenta el tamaño de la matriz. Esto se ve reflejado en la tendencia creciente de la aceleración graficada.

4. *Z-Order*

En esta sección se estudia el rendimiento del esquema de acceso a memoria *Z-Order* en cuanto al producto de matrices. Para ello, se ha realizado dicha operación para los siguientes tamaños de matrices: {2, 4, 8, 16, 32, 64, 128, 256, 398, 512, 636, 774, 892, 1024, 1152, 1280, 1408, 1536}. Para cada tamaño de matriz, se ha probado con todos los tamaños de bloque posibles, incluyendo el tamaño de bloque 1 y el del propio tamaño de la matriz. Cada producto se ha realizado un total de 32 veces y se ha medido el tiempo medio de ejecución.
Como los resultados para cada tamaño de matriz dependen del tamaño de bloque, las gráficas a mostrar son las resultantes de las ejecuciones de tamaños de matriz específicos, donde se pueden observar los tiempos de ejecución para cada tamaño de bloque. En la Figura 3 se muestran los resultados obtenidos para el tamaño de matriz 1024×1024.

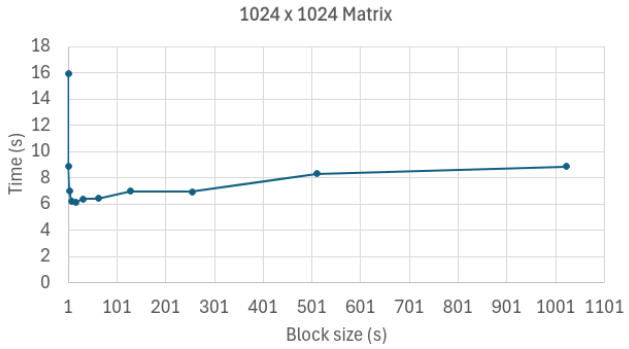

Figura 3: Tiempos de ejecución de *Z-Order* para el tamaño de matriz 1024 × 1024

Se observa a simple vista una tendencia que se repite en todos los tamaños de matriz probados:

- El tamaño de bloque 1 es el que más tiempo de ejecución requiere.
- Los tamaños de bloque relativamente pequeños (4, 8, 16 y 32 en este caso) son los que mejores resultados ofrecen.
- A partir de cierto valor de tamaño de bloque, el tiempo de ejecución comienza a aumentar.

Esta tendencia se repite de forma similar para todos los tamaños de matriz probados. Véase en la Figura 4 los resultados obtenidos para el tamaño de matriz 1280 × 1280 y en la Figura 5 los resultados obtenidos para el tamaño de matriz 1536 × 1536.

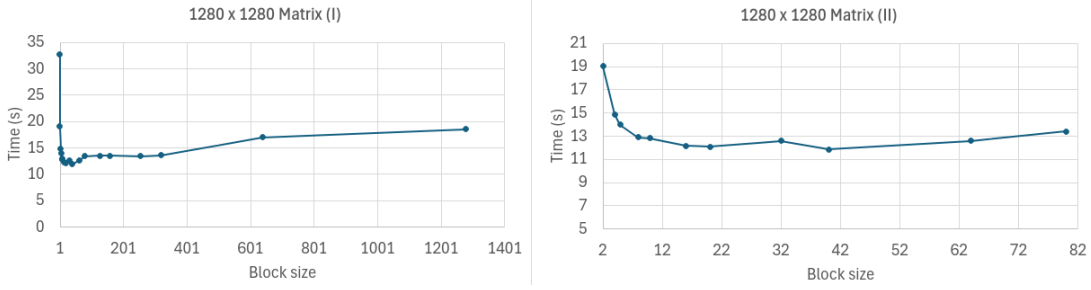

Figura 4: Tiempos de ejecución de *Z-Order* para el tamaño de matriz 1280 × 1280

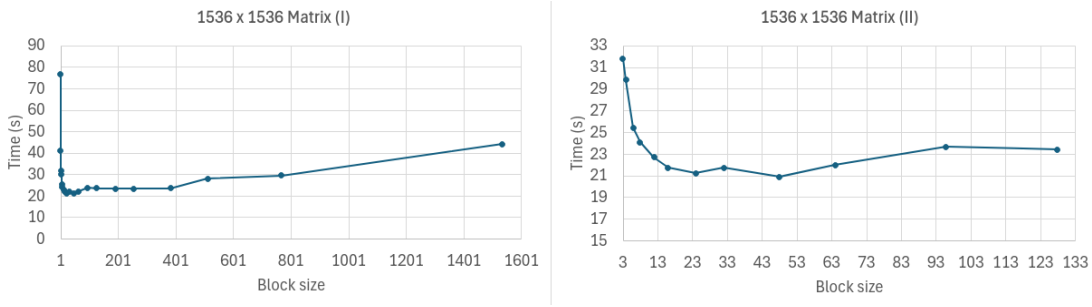

Figura 5: Tiempos de ejecución de *Z-Order* para el tamaño de matriz 1536×1536

Por lo tanto, se puede concluir que el uso de *Z-Order* es muy eficiente si se utilizan tamaños de bloque óptimos, en concreto, tamaños de bloque relativamente pequeños (valores entre 8 y 64 válidos para el tamaño de matriz específico). Sin embargo, el uso de tamaños de bloque demasiado pequeños o demasiado grandes puede resultar en un rendimiento muy deficiente.

Para terminar con este estudio, se analizará el rendimiento de *Z-Order* respecto al de los esquemas de acceso a memoria *Row-Major order* y *Column-Major order*. Concretamente se comparará, para cada tamaño de matriz, el tiempo de ejecución de *Row-Major order* y *Column-Major order* con el de *Z-Order* utilizando el tamaño de bloque más óptimo en cada caso. Los resultados se muestran en la Tabla 1. Se han omitido los tamaños de matriz pequeños ya que no son representativos.

Matrix size	*Row-Major order* (s)	*Column-Major order* (s)	*Z-Order* (s)
[...]	[...]	[...]	[...]
256×256	0,09026	0,089583	0,089992
398×398	0,394595	0, 388266	**0,375128**
512×512	0,946308	0,902864	**0,747350**
636×636	1,784823	1,702211	**1,447440**
774×774	3,397629	3,167861	**2,548269**
892×892	5,104598	4,827320	**4,665994**
1024×1024	8,473015	7,278059	**6,071009**
1152×1152	12,413447	10,524069	**8,527498**
1280×1280	17,331966	14,561772	**11,818383**
1408×1408	27,91348	22,76128	**15,968882**
1536×1536	42,912048	32,64931	**20,887361**

Tabla 1: Tiempos de ejecución de *Row-Major order*, *Column-Major order* y mejor *Z-Order*

A simple vista se observa que el rendimiento de *Z-Order* es muy similar en tamaños de matrices medianos y mucho mejor en tamaños de matrices grandes. Concretamente, su rendimiento mejora respecto a *Row-Major order* y *Column-Major order* a medida que el tamaño de matriz aumenta.

5. Conclusiones

En esta sección se presentan las conclusiones obtenidas durante el desarrollo de la práctica y a partir de los resultados obtenidos.

5.1. Curiosidades

5.1.1. *Row-Major order* vs *Column-Major order*

En un principio, se esperaba que el esquema de acceso a memoria *Row-Major order* fuese más eficiente que *Column-Major order* debido a la forma en la que se almacenan los datos en memoria. Sin embargo, los resultados obtenidos muestran lo contrario. Se ha analizado y revisado el código y la implementación de ambos esquemas por si existiese algún error, pero no se ha encontrado nada que lo justifique. Incluso se han utilizado herramientas de análisis de rendimiento como `perf` y, de forma contraria a lo analizado de forma empírica en la Sección 2, *Row-Major order* produce más fallos de caché que *Column-Major order*.

La única explicación *lógica* a este suceso es la mayor reutilización temporal que se produce en el esquema *Column-Major order* respecto a *Row-Major order*. En concreto, esto se debe a que, durante el cálculo de cada columna de la matriz resultado C, se accede repetidamente a una misma columna de la matriz B. Dado que la matriz B está almacenada en memoria en orden por filas, acceder a una columna implica realizar saltos entre elementos que no son contiguos en memoria. Sin embargo, como esa misma columna se reutiliza varias veces seguidas dentro del mismo cálculo, los datos una vez cargados en caché permanecen disponibles durante varias iteraciones.

A pesar de esta explicación, no se puede asegurar con total certeza que este fenómeno sea la causa principal de la diferencia de rendimiento entre ambos esquemas, ya que en la práctica intervienen muchos factores de bajo nivel que pueden influir significativamente en los resultados.

5.1.2. Tamaños de bloque en *Z-Order*

Como se comentó en la Sección 4, para cada tamaño de matriz se ha probado con todos los tamaños de bloque posibles, incluyendo el tamaño de bloque 1 y el del propio tamaño de la matriz. Esto se ha realizado a propósito para comprobar qué sucede en cada caso.

La base del algoritmo *Z-Order* es la división recursiva de las matrices en submatrices por lo que el uso de un tamaño de bloque 1 no tiene sentido, ya que no se produce dicha división. Esto se traduce en un rendimiento muy deficiente y se ha comprobado con los resultados. El uso de un tamaño de bloque igual al del tamaño de la matriz tampoco tiene sentido a nivel lógico, pero su rendimiento, si bien no es el mejor, es mucho más óptimo que en el caso anterior.

5.2. Dificultades

Durante el desarrollo e implementación de la práctica no se han encontrado dificultades significativas. Sí que es cierto que el hecho de haber realizado un programa "extra" para realizar *benchmarks* de forma más simple y rápida ha supuesto un esfuerzo adicional, pero ha simplificado mucho el proceso de toma de datos. La mayor dificultad ha sido la de averiguar y analizar por qué *Row-Major order* es más lento que *Column-Major order*, ya que no se esperaba este resultado. La conclusión respecto a este fenómeno ha sido la comentada en la Subsubsección 5.1.1.