Informe de Rendimiento: Comparación de Bucles, Multiplicación de Matrices Clásica y por Bloques

Pablo Luis Carazas Barrios

19 de marzo de 2025

1. Introducción

Este informe presenta un análisis comparativo entre:

- Dos bucles anidados que realizan la misma operación, pero en distinto orden de recorrido (capítulo 2, pág. 22 del libro).
- La multiplicación de matrices clásica (tres bucles).
- La multiplicación de matrices por bloques (seis bucles).

Para evaluar el rendimiento de cada enfoque, se realizaron mediciones de *cache misses* y tiempos de ejecución con distintas herramientas (`valgrind` + `kcachegrind`) y distintos tamaños de matriz y de bloque.

2. Comparación de los Dos Bucles Anidados

2.1. Descripción de los bucles

Se tienen dos variantes que realizan la misma suma de productos $y[i] += A[i][j] \times x[j]$, pero difieren en si primero se itera sobre i y luego sobre j, o viceversa. Aunque la cantidad de operaciones es la misma, el orden en que se recorre la matriz A influye en la localidad de datos, debido al almacenamiento *row-major* en C/C++.

2.2. Resultados experimentales

La siguiente tabla muestra un ejemplo de los *cache misses* obtenidos con distintos tamaños de matriz:

Tamaño Matriz	Bucle 1	Bucle 2
10	0	0
100	9.29	8.14
250	38.56	39.03
500	49.27	71.00
1000	89.70	98.74

Cuadro 1: Ejemplo de medición de *cache misses* (o tiempo) para dos órdenes de bucles.

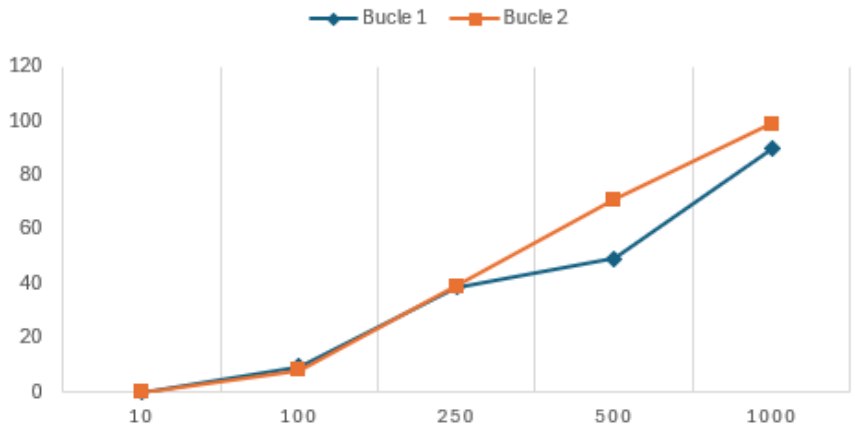

Figura 1: Comparación entre *cache misses* en bucles con distinto orden de índices.

2.3. Análisis

Los datos muestran un comportamiento diferente dependiendo del orden en que se recorre la matriz A. Al incrementarse el tamaño de la matriz, el *bucle 1* (donde el índice de filas `i` es el externo y el de columnas `j` es el interno) tiende a tener menos *cache misses* que el *bucle 2*, pero la magnitud de la diferencia varía según el tamaño de `MAX`. Existen varias razones para ello:

- **Almacenamiento *row-major***: En C/C++, las matrices se almacenan por filas, es decir, las posiciones contiguas en memoria corresponden a elementos `A[i][0]`, `A[i][1]`, ... Cuando el bucle externo avanza por `i` y el interno por `j`, se accede a elementos que están más próximos en memoria, reduciendo la probabilidad de fallos de caché.
- **Localidad espacial**: Al recorrer fila por fila (`i` fijo y `j` de 0 a MAX-1), los elementos requeridos para el siguiente paso de la iteración suelen estar en la misma línea de caché o en líneas próximas. En cambio, si se invierte

el orden (`j` externo, `i` interno), se "salta" entre filas distintas, forzando un mayor número de cargas en caché para traer datos no contiguos.

- **Tamaños pequeños (p.ej., 10)**: Cuando la matriz es muy pequeña, la totalidad de `A` (y los vectores `x` e `y`) puede caber en caché L1 o L2, por lo que apenas hay *cache misses*, y las diferencias entre uno u otro orden de bucles se vuelven insignificantes.
- **Tamaños grandes (p.ej., 250, 500, 1000)**: A medida que crece la matriz, la ventaja de acceder a elementos contiguos es más notoria, pues cada vez que se produce un *miss*, la penalización de traer datos de memoria principal es más costosa. De ahí que el *bucle 1* muestre un conteo menor (o un crecimiento más lento) de fallos de caché.
- **Misma complejidad, distinto acceso a memoria**: Aunque en ambos bucles se realizan $\mathcal{O}(n^2)$ operaciones, el orden del recorrido de `A` afecta directamente el patrón de acceso a memoria y, por tanto, el tiempo de espera del procesador para cargar datos a caché.

En consecuencia, el orden de los bucles puede marcar la diferencia entre un buen y un mal aprovechamiento de la jerarquía de memoria, a pesar de que los dos enfoques tengan la misma complejidad teórica ($O(n^2)$).

3. Multiplicación de Matrices Clásica

3.1. Descripción

La multiplicación de matrices clásica consta de tres bucles anidados que recorren índices i, j y k. Su complejidad es $O(n^3)$ en número de operaciones. La disposición en memoria y el orden de los bucles pueden afectar el uso de la caché, pero generalmente se conoce como la versión básica que sirve de referencia para comparaciones.

4. Multiplicación de Matrices por Bloques

4.1. Descripción de la técnica de bloques

La idea central es dividir cada matriz en submatrices ("bloques") de tamaño $b \times b$, de modo que se operen submatrices completas que puedan permanecer en la caché el mayor tiempo posible. Con ello se busca reducir el número de *cache misses* frente a la versión clásica.

4.2. Comparación entre distintos tamaños de bloque

La siguiente tabla muestra un ejemplo de mediciones de *cache misses* usando diferentes tamaños de bloque (16, 32, 64 y 128) para matrices de distintos tamaños:

Tamaño Matriz	Bloques 16	Bloques 32	Bloques 64	Bloques 128
10	0	0	0	0
100	30.49	24.45	23.28	23.28
250	81.68	79.67	75.13	98.64
500	88.03	91.03	89.53	99.73

Cuadro 2: Medición de *cache misses* o métrica de tiempo según distintos tamaños de bloque.

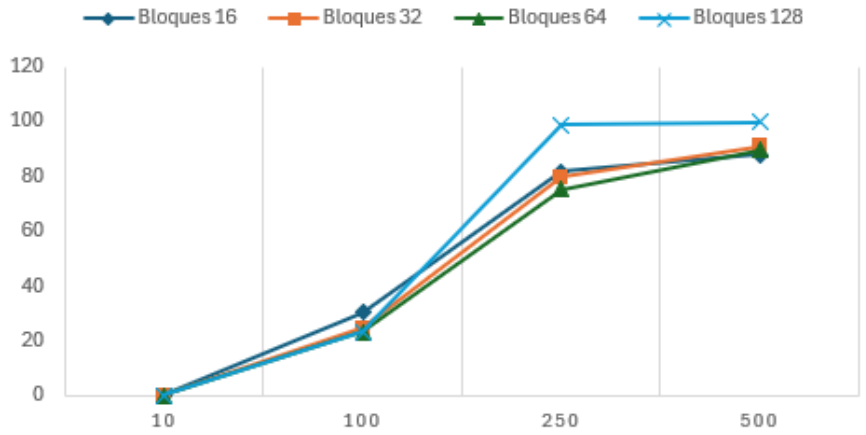

Figura 2: Comparación entre Cache Misses entre diferentes tamaño de bloques.

- Un tamaño de bloque demasiado grande (p.ej., 128) puede saturar la caché y empeorar el resultado.
- Un bloque demasiado pequeño (16) puede agregar sobrecarga extra en bucles sin explotar al máximo la localidad.
- En estos experimentos, los bloques de 32 o 64 ofrecen un equilibrio razonable.

5. Resultados Globales: Clásica vs. Bloques

5.1. *Cache misses* promedio

En la Tabla 3 se presenta un resumen de *cache misses* promedio al comparar la versión clásica y la por bloques.

Tamaño Matriz	Bloques (promedio)	Clásica
10	-	-
100	25.375	23.44
250	83.78	98.58
500	92.08	85.44

Cuadro 3: *Cache misses* promedio entre versión por bloques y versión clásica.

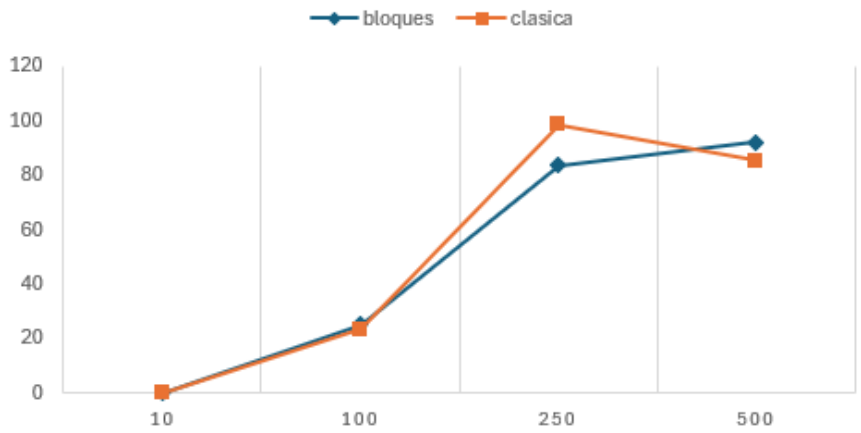

Figura 3: Comparación entre Cache Misses entre tipos de multiplicación.

Análisis de *cache misses* Tal como se muestra en la Tabla 3, la versión por bloques puede reducir la cantidad de *cache misses* (por ejemplo, en $n = 250$), mientras que en otros tamaños (100, 500) la versión clásica alcanza valores similares o incluso menores. Esto se debe principalmente a:

- **Concordancia con la caché**: Los tamaños de bloque elegidos pueden encajar mejor (o peor) en la jerarquía de memoria (L1, L2, L3). Cuando el bloque entra de forma óptima, se logra una buena localidad de datos, reduciendo *cache misses*.
- **Desajuste en ciertos tamaños**: Para matrices más grandes (500) o más pequeñas (100), el bloque puede no estar ajustado a la capacidad de la caché, lo que hace que la supuesta ventaja de la localidad se pierda. Además, el procesamiento de sub-bloques introduce más índices y lógica de control.
- **Ganancias variables según** n: En $n = 250$, la versión por bloques parece beneficiarse de la *localidad* conseguida con determinados bloques, mientras que en 500 la estructura de bloques no reduce tanto los accesos a memoria principal.

5.2. Tiempos de ejecución

La Tabla 4 compara los tiempos de ejecución (en ms) de la versión clásica y la versión por bloques con diferentes tamaños de bloque.

Tamaño Matriz	Clásica	B16	B32	B64	B128
10	2	3	2	2	2
100	306	353	455	349	370
250	5013	8511	5430	5407	5618
500	40562	44066	63405	60068	53859

Cuadro 4: Tiempos de ejecución en la versión clásica y la versión por bloques (B16, B32, B64, B128).

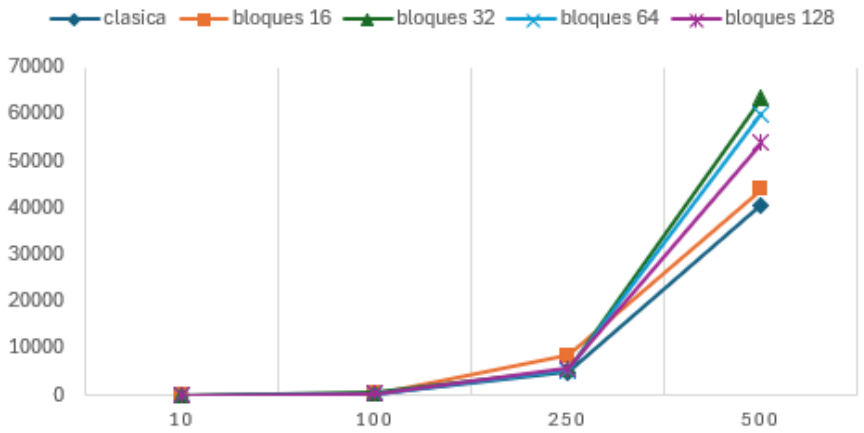

Figura 4: Comparación entre tiempos de ejecución (en ms) entre tipos de multiplicación y tamaño de bloques.

Análisis de los tiempos de ejecución

- **Reducción de *cache misses* vs. overhead de bucles adicionales**: Aunque la multiplicación por bloques puede ofrecer mejor localidad de datos, también introduce una sobrecarga de control (bucles adicionales y cálculo de límites en cada sub-bloque). Por eso, en muchos casos, la versión clásica sigue resultando más rápida, a pesar de no siempre tener la menor cifra de *cache misses*.

- **Escenarios favorables a la versión por bloques**: En el caso de $n = 250$, cuando se elige un bloque de tamaño intermedio (por ejemplo, 32 o 64), se observa una reducción notable de *cache misses*. Sin embargo, el tiempo total no siempre desciende de modo proporcional, precisamente

por la lógica extra. Aun así, la versión por bloques se acerca al rendimiento de la clásica en ciertos escenarios.

- **Procesadores modernos y prefetching**: Algunos procesadores son muy eficientes en traer datos a la caché anticipadamente (*prefetch*), de modo que si la versión clásica tiene un patrón de acceso predecible, logra enmascarar parte del coste de los accesos a memoria. Esto puede explicar por qué la versión clásica sigue siendo muy competitiva incluso cuando la versión por bloques reduce los *cache misses*.
- **Dependencia del hardware**: Resultados distintos pueden encontrarse en otros procesadores con diferentes tamaños de L1, L2 o L3. Aquellas arquitecturas con cachés más grandes o mejor optimización de prefetch podrían sacar mayor provecho a la versión por bloques.

En conclusión, si bien la versión por bloques demuestra ser capaz de disminuir los *cache misses* en ciertos tamaños de matriz y bloques, esa ventaja no siempre se traduce en un menor tiempo de ejecución por la sobrecarga de bucles y otros factores (prefetch, jerarquía de memoria, vectorización, etc.).

6. Uso de Herramientas: Valgrind y KCachegrind

Para cuantificar los *cache misses* se utilizó:

- `valgrind --tool=callgrind`
  Permite recolectar información sobre el uso de memoria y contadores de instrucciones.
- `kcachegrind`
  Para visualizar y analizar los resultados, identificando cuántos accesos e invocaciones se realizan y cuántos fallos de caché tienen lugar.

7. Conclusiones Generales

- **Bucles Anidados:** Recorrer la matriz con el índice de fila externo e índice de columna interno (i–j) suele ser más eficiente para tamaños grandes en memoria *row-major*.
- **Versión Clásica vs. Bloques:**
  - La *multiplicación clásica* (tres bucles) es sencilla y, en estos experimentos, frecuentemente más rápida en la mayoría de los tamaños probados.
  - La *multiplicación por bloques* (seis bucles) a veces reduce los *cache misses*, pero puede mostrar sobrecarga en el control de sub-bloques, que en ciertos tamaños compensa o anula la ganancia en localidad.

- Existen casos intermedios (p.ej., matrices de tamaño 250 y bloques de 32 o 64) donde la versión por bloques se aproxima e incluso puede superar en *cache misses* a la clásica; no obstante, el tiempo final puede no ser mejor.

- **Tamaño de Bloque:** No hay un bloque universalmente óptimo. Bloques muy grandes (128) pueden degradar la ventaja de la localidad de caché, mientras que bloques muy pequeños (16) añaden demasiados bucles. Un equilibrio (32–64) suele ser más favorable.

Referencias:

- Apuntes del Libro (Cap. 2, pág. 22) para la comparación de bucles anidados.
- Documentación de `valgrind` (`callgrind`) y `kcachegrind`.

7.1. Repositorio

https://github.com/Pablito290/Paralela