

---

# PRÁCTICA 7: LOCALIDAD Y *Cache Blocking*

---

Arquitectura e Ingeniería de Computadores (3º curso)  
E.T.S. de Ingeniería Informática (ETSINF)  
Dpto. de Informática de Sistemas y Computadores (DISCA)

## Objetivos:

- Entender cómo los patrones de acceso a memoria de un programa afectan a sus prestaciones.
- Modificar los patrones de acceso a memoria para mejorar el rendimiento.
- Implementar la técnica *cache blocking*.

## Multiplicación de matrices

### Antecedentes

Las matrices son estructuras de datos bidimensionales en las que cada elemento se accede a través de dos índices. Para multiplicar dos matrices, se pueden usar 3 simples bucles anidados. Asumiendo que las matrices A, B, y C son todas de tamaño  $n \times n$  y se encuentran almacenadas linealmente en memoria, un posible algoritmo de multiplicación ( $C = A \times B$ ) sería el siguiente:

```
for ( int i = 0; i < n; i++ )  
    for ( int j = 0; j < n; j++ )  
        for ( int k = 0; k < n; k++ )  
            C [i + j*n] + = A [k + j*n] * B [i + k*n];
```

La multiplicación de matrices se usa en muchos algoritmos de álgebra lineal y su eficiencia es fundamental para muchas aplicaciones, como los gráficos en videojuegos o en el entrenamiento y la inferencia en redes neuronales.

Si se consideran las iteraciones del bucle interno del algoritmo previo (índice k), se observa que:

- Se accede a componentes consecutivas (*stride* 1) de la matriz A.
- Se accede con *stride*  $n$  a las componentes de la matriz B.
- Se accede a la misma componente de la matriz C (*stride* 0).

Nótese que para calcular correctamente la multiplicación de matrices, el orden de los bucles no importa. Sin embargo, el orden de estos bucles sí es importante para el rendimiento. Las memorias *cache* funcionan mejor (con más aciertos y menos fallos) cuando los accesos a memoria exhiben una mayor localidad espacial y temporal. La optimización del patrón de acceso a memoria de un programa es esencial para obtener unas buenas prestaciones.


## Ejercicios


1. Estudiar cómo escala el algoritmo básico de multiplicación de matrices con el tamaño de la matriz.

Inspecciona el fichero `matrix1.c`, el cual contiene una implementación de la multiplicación de matrices. Esta implementación se puede compilar con la orden:

```
$ gcc -O3 -o matrix1 matrix1.c
```

Ejecuta `matrix1` y observa los resultados. Cópalos en algún lugar (por ejemplo, una hoja de cálculo) para no tener que ejecutar el programa repetidas veces.


 Piensa en el algoritmo básico de multiplicación de matrices. ¿Es relativamente bueno o malo en términos de localidad? ¿Por qué?


 ¿Por qué disminuyen las prestaciones para valores grandes de  $n$ ? (pista: ¿De qué depende el tiempo que permanece en la cache un bloque?)

 Para obtener información sobre la CPU (modelo, tamaño de cache, etc.) utiliza la orden `lscpu`.

2. Realizar varias implementaciones del algoritmo de multiplicación de matrices variando el orden de los bucles.

Implementa en el fichero `matrix2.c` los 6 posibles algoritmos de multiplicación de matrices que se pueden realizar variando el orden de los bucles. Compila y ejecuta `matrix2`.

 ¿Cuál de los 6 diferentes ordenamientos proporciona unas mayores prestaciones? ¿Cuál o cuales proporcionan las peores?

 ¿Proporcionan los índices con los que se accede a cada una de las matrices en el bucle interno alguna orientación para seleccionar el ordenamiento con mayores prestaciones?

## Transposición de matrices

### Antecedentes

A veces, deseamos intercambiar filas y columnas de una matriz. Esta operación se llama *transposición* y una implementación eficiente puede ser muy útil cuando se realizan operaciones de álgebra lineal complejas. La transpuesta de la matriz  $A$  se denota como  $A^T$ . La siguiente figura ilustra un ejemplo de transposición de los elementos de una matriz 5x5:

1	2	3	4	5		1	6	11	16	21
6	7	8	9	10		2	7	12	17	22
11	12	13	14	15	⇒	3	8	13	18	23
16	17	18	19	20		4	9	14	19	24
21	22	23	24	25		5	10	15	20	25

El código básico para realizar la transposición de matrices es el siguiente:

```
for( i = 0; i < n; i++ )
    for( j = 0; j < n; j++ )
        dst[j + i*n] = src[i + j*n];
```

Tal como ocurre con la multiplicación de matrices, este código causa muchos fallos por la poca reutilización de los datos almacenados en cache.

Además de cambiar el orden de los bucles, se puede incrementar la reutilización de los datos implementando una técnica denominada *cache blocking*. La técnica *cache blocking* reduce la tasa de fallos de la cache mejorando la localidad temporal y espacial de los accesos a memoria. Cuando se aplica la técnica a la transposición, la matriz se divide en submatrices  $A_{ij}$ , y cada matriz se transpone por separado en su ubicación final en la matriz transpuesta, tal como se muestra en la siguiente figura:

$A_{11}$	$A_{12}$	$A_{13}$		$A_{11}^T$	$A_{21}^T$	$A_{31}^T$
$A_{21}$	$A_{22}$	$A_{23}$	⇒	$A_{12}^T$	$A_{22}^T$	$A_{32}^T$
$A_{31}$	$A_{32}$	$A_{33}$		$A_{13}^T$	$A_{23}^T$	$A_{33}^T$

Con esta técnica, se reduce significativamente el tamaño del conjunto de datos con los que el algoritmo trabaja en cada momento, lo que reduce el número de fallos y por tanto redanda en una mejora de las prestaciones.

## Ejercicios

1. Implementar la transposición de matrices utilizando la técnica de *cache blocking*.

El fichero `transpose.c` contiene dos funciones que realizan la transposición de matrices. La primera función implementa el algoritmo básico visto más arriba. La segunda función debe implementarse por el alumno.

Antes de realizar la implementación, razona las siguientes cuestiones:

- 🔖 ¿Cuántos bucles debe contener el algoritmo que utiliza *cache blocking*? ¿Cuál es el objetivo de los dos primeros bucles?
- 🔖 ¿Cuál debe ser el rango que recorre cada bucle? ¿En cuánto debe incrementarse cada uno de los índices de los bucles en cada iteración?

Una vez respondidas ambas cuestiones, realiza la implementación y compila y ejecuta el fichero `transpose.c` para comprobar su correcto funcionamiento y cuantificar la mejora de prestaciones.

## **(Opcional) Mejorar las prestaciones de la multiplicación de matrices utilizando cache blocking**

La técnica de cache blocking también puede aplicarse a la multiplicación de matrices. Propón una implementación, coméntala con tu profesor y desarróllala en el fichero `matrix3.c`. Posteriormente comprueba la mejora de prestaciones obtenida con respecto a `matrix1`.