



Ejercicios de memoria caché (segunda parte)

Ejercicio 1

Sea el siguiente fragmento de código en lenguaje C:

```
double A[256][256];  
double B[256][256]  
  
void transponer(double ** dest, double ** orig) {  
    for (int i=0;i<256;++i) {  
        for (int j=0;j<256;++j) {  
            dest[j][i] = orig[i][j]  
        }  
    }  
}
```

Recuerde que en el lenguaje C, las matrices se almacenan por filas. Es decir que el orden de almacenamiento es: A[0][0], A[0][1], A[0][2], ..., A[0][255], A[1][0], A[1][1], ...A[1][255], ...

Suponga que este fragmento de código se ejecuta en un procesador con una caché L1 totalmente asociativa de 16 KB con política LRU y tamaño de línea de 64 bytes. Asuma que un fallo de caché requiere 16 ciclos (al igual que una lectura adelantada o *prefetch*). Asuma también que la caché L2 siempre produce aciertos y que puede procesar una petición cada dos ciclos. Cada iteración del bucle interno requiere cuatro ciclos si el dato está presente en la caché L1. La caché tiene una política de asignación en escritura (las escrituras también traen bloques a la caché) con captación en escritura en caso de fallos de escritura. Ignore el coste de las post-escritura a niveles inferiores.

Observe que el código que se propone hace que se recorra la matriz destino en un orden problemático (aunque es un orden ideal para la matriz origen). No obstante si se intercambian los bucles el problema sería el inverso. Para solventar este problema es necesario realizar un acceso por bloques

Ejercicio 1.1

¿Cuál sería el tamaño mínimo de la caché para aprovechar el acceso por bloques?

Como la matriz contiene valores de tipo double, cada elemento requiere 8 bytes. Al tratarse de líneas de 64 bytes, el número de elementos que caben en una línea es 8 valores. Al realizarse acceso por bloques al traer a la caché un elemento se traen 8 de esa fila. Por tanto, el tamaño mínimo sería usando bloques de 8 x 8.



Se necesitarían almacenar en la caché 64 elementos de la matriz origen y 64 elementos de la matriz destino. En total 128 elementos.

Tamaño = $128 \times 8 \text{ bytes} = 1 \text{ KB}$

Ejercicio 1.2

¿Qué relación hay entre el número relativo de fallos en la versión original y la versión de acceso por bloques para el caso de la caché de tamaño mínimo del ejercicio 1.1?

La versión original tendrá para la matriz origen un fallo por cada 8 accesos.

Sin embargo, para los accesos a la matriz destino, el acceso a (por ejemplo) `dest[0][1]`, se producirá solamente después de haber traído a la caché el primer bloque de todas las filas. Es decir, se habrá traído a memoria $256 \times 64 \text{ bytes}$ (16 KB). Por tanto, ya se habrán remplazado todas las líneas de la caché. Así que la matriz destino produce un 100% de fallos.

En total la versión original produce 9 fallos por cada 16 accesos.

$$m = 9/16 = 0.5625$$

Por otra parte, la versión de acceso por bloques necesita mantener en caché 8 bloques de la matriz origen y 8 bloques de la matriz destino. En total $16 \times 64 \text{ bytes}$ (1 KB), por lo que se producirán 16 fallos por cada 128 accesos.

$$m = 16 / 128 = 0.125$$

La mejora del acceso por bloques es por tanto:

$$\text{mejora} = 0.5625 / 0.125 = 4.5$$



Ejercicio 1.3

Escriba el código de la función para que realice la transposición con un tamaño de bloque dado por un valor B, para el tamaño de bloque.

```
void transponer(double ** dest, double ** orig, int B) {  
    for (int i=0; i<256; i+=B) {  
        for (int j=0; j<256; j+=B) {  
            for (int k=0; k<min(i+B,256); ++k) {  
                for (int l=0; l<min(j+B,256); ++l) {  
                    dest[j][i] = orig[i][j]  
                }  
            }  
        }  
    }  
}
```

Ejercicio 1.4

¿Cuál es el nivel mínimo de asociatividad necesario para que la caché L1 tengan el mejor rendimiento independientemente de las posiciones de las matrices en memoria?

Con una memoria caché de correspondencia directa podría ocurrir que un bloque de la matriz origen tenga asignada la misma posición en caché que otro de la matriz destino dando lugar a fallos de conflicto.

En una memoria caché asociativa de dos vías se soluciona este problema.

Si se añaden más de dos vías no se obtiene ninguna ganancia adicional para este problema

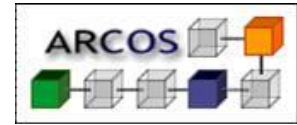
Ejercicio 1.5

Suponga que en el caso original, se decide realizar lecturas adelantadas de bloques. Determine el número máximo de lecturas adelantadas que podría haber en curso en un momento dado.

Se puede iniciar una lectura adelantada cada dos ciclos. Como la matriz destino siempre genera un fallo, habría que hacer una lectura adelantada por iteración. Además para la matriz origen habría que hacer una lectura adelantada cada 8 iteraciones. Esto es más de una lectura adelantada por iteración.

En el caso de una lectura adelantada por iteración, se inicia una lectura adelantada cada cuatro ciclos, por lo que habría como máximo cuatro lecturas adelantadas (16/4) en curso.

A esto hay que añadir que cada 8 iteraciones (32 ciclos) se requiere una lectura adelantada adicional.



Por tanto en el peor de los casos habría 5 lecturas adelantadas en curso.