MEMORIA PRACTICA 2 DE TÉCNICAS DE OPTIMIZACIÓN
Realizado por Jaime Lorenzo Sánchez

Ejercicio 1. Realizar una implementación y prueba del siguiente código, donde las dimensiones de las matrices son 128,128,128 para A y B, y 64,64,64,64 para C y D; N = 64

```
for (i = 0; i < N; i++)
        for (j = 0;  j < N; j++)
                for (k = 0; k < N; k++)
                        for (l = 0; l < N; l++)
                                B[k][l][i] = A[i][k][j] + C[l][j][k][i] * D[k][j][l][i];
```

Para la realización de la práctica, se ha utilizado una máquina virtual y un equipo con la siguiente información:

Procesador del equipo	Intel Core i7-8565U
Entorno de desarrollo	Ubuntu 20.04.3
Herramienta de desarrollo	C++
Compilador empleado	G++ version 9.3.0

Características de la memoria caché L1

data	4x32 KB (4 bloques de 32 KB)
line size	64 B
Associativity	8-way set

Calculamos el número de líneas por vía:

$$\frac{32\,KB}{8} = \frac{4\,KB}{via} \rightarrow \frac{\frac{4\,KB}{via}}{\frac{64\,B}{via}} = \frac{64\,lineas}{via}$$

Ejercicio 2. Obtener los tiempos de ejecución del programa descrito con O0 y O2

	Opcion -O0	Opción -O2
Vector (i,j,k,l)	232.23 ms	218.37 ms

Ejercicio 3. Aplicar el algoritmo de permutación de bucles visto en clase al desarrollo realizado y obtener los tiempos de ejecución del programa con O0 y O2

$$CacheTurns(X, I_k) = \frac{(U_k - L_k + 1) \cdot Stride(X,I_k) \cdot SetStride(X,I_k)}{C \cdot W}$$

- Uk: Limite superior del bucle
  - Lk : Limite inferior del bucle
- L: Número de bloques por conjunto o asociatividad → L = 8
- C: Número de conjuntos o sets → C = 64
- W: Número de palabras por bloque → w = 16
- SetStride (X,Ik): Distancia en sets entre los bloques que contienen la referencia a X realizadas por iteraciones consecutivas del índice Ik

$$Stride(X, I_k) = f_k \cdot I_k^{I+1} \cdot \prod_{i=1}^{k-1} D_i - f_k \cdot I_k^{I} \cdot \prod_{i=1}^{k-1} D_i$$

$$SetStride(X, I_k) = \left\lceil \frac{Stride(X,I_k)}{W} \right\rceil \bmod C$$

CALCULAMOS EL VALOR DE FA PARA CADA MATRIZ				
FA(A)	$128^2 * i + 128 * k + j$			
FA(B)	$128^2 * k + 128 * l + i$			
FA(C)	$64^3 * l + 64^2 * j + 64 * k + i$			
FA(D)	$64^3 * k + 64^2 * j + 64 * l + i$			
CALCULAMOS EL STRIDE DE CADA MATRIZ				
	x = i	X = j	X = k	X = l
Stride (A,x)	128^2	1	128	0
Stride (B,x)	1	0	128^2	128
Stride (C,x)	1	64^2	64	64^3
Stride (D,x)	1	64^2	64^3	64
CALCULAMOS EL SETSTRIDE DE CADA MATRIZ				
	x=i	x=j	x=k	x=l
SetStride(A,x)	0(64)	1	8	0(64)
SetStride(B,x)	1	0(64)	0(64)	8
SetStride (C,x)	1	0 (64)	4	0 (64)
SetStride (D,x)	1	0 (64)	0 (64)	4
CALCULAMOS CACHETURNS PARA CADA MATRIZ				

	x=i	x=j	x=k	x=l
CacheTurns(A,x)	65560	0.064	65	0
CacheTurns(B,x)	0.064	0	65560	65
CacheTurns(C,x)	0.064	16640	16.25	1064960
CacheTurns(D,x)	0.064	16640	1064960	16.25
CALCULAMOS VALOR VACHETURNS PARA CADA INDICE Ik				
	CacheTurns(i)	CacheTurns(j)	CacheTurns(k)	CacheTurns(l)
	65560.192	33280.064	1130601.26	1065041.25

Una vez calculado, situamos los bucles en orden decreciente en función del valor CacheTurns. Una vez ordenados los bucles, realizamos el estudio de los tiempos de ejecución para el vector (k,l,i,j).

El código a implementar es el siguiente:

```
for(int k=0; k<64; k++){
    for(int l=0; l<64; l++){
        for(int i=0; i<64; i++){
            for(int j=0; j<64; j++){
                B[k][l][i] =+ A[i][k][j] + C[l][j][k][i] * D[k][j][l][i];
            }
        }
    }
}
```

Tras la ejecución del programa, hemos obtenido los siguientes tiempos de ejecución:

	Opcion -O0	Opción -O2
Vector (k,l,i,j)	142.338 ms	62.025 ms

Ejercicio 4. Realizar una justificación exhaustiva de los resultados obtenidos.

Para realizar una justificación del los resultados obtenidos, primero debemos realizar un estudio de los tiempos de ejecución del algoritmo original con el algoritmo de permutación de bucles

	Opcion -O0	Opción -O2
Vector (i,j,k,l)	232.23 ms	218.37 ms
Vector (k,l,i,j)	142.338 ms	62.025 ms

El algoritmo de permutación de bucles explota la localidad espacial utilizando la probabilidad de reemplazo de los bloques de la caché. Al aplicar esta permutación con el bucle con menor CacheTurns para cada indice Ik, se producen menos fallos de caché y se reduce drásticamente el tiempo de ejecución.