MEMORIA CACHE: EJEMPLO PRACTICO

I. EXPLICACION DEL PROBLEMA

Se pide analizar el funcionamiento de los siguientes dos algoritmos con respecto al uso y acceso de la memoria caché:

```
double A[MAX][MAX], x[MAX], y[MAX];
. . .
/* Initialize A and x, assign y = 0 */
. . .
/* First pair of loops */
for (i = 0; i < MAX; i++)
   for (j = 0; j < MAX; j++)
      y[i] += A[i][j]*x[j];
. . .
/* Assign y = 0 */
. . .
/* Second pair of loops */
for (j = 0; j < MAX; j++)
   for (i = 0; i < MAX; i++)
      y[i] += A[i][j]*x[j];
```

Cache Line	Elements of A			
0	A[0][0]	A[0][1]	A[0][2]	A[0][3]
1	A[1][0]	A[1][1]	A[1][2]	A[1][3]
2	A[2][0]	A[2][1]	A[2][2]	A[2][3]
3	A[3][0]	A[3][1]	A[3][2]	A[3][3]

II. ANALISIS DEL FUNCIONAMIENTO

La utilización de la memoria caché se relaciona con los "cache miss", que corresponden a la no coincidencia cuando se consulta por un valor en la línea de caché.

Primer Algoritmo

El recorrido de las consultas del arreglo es por columnas (j) de una misma fila (i), lo que genera que el valor consultado coincida en la misma línea de caché en toda la fila.
Cuando la consulta es en una línea siguiente, se genera un cache miss, por lo que la cantidad de cache miss que se incurrirán será la división de la cantidad de elementos de A entre los elementos continuos almacenados en la línea de caché.

$$\#cacheMiss = \frac{\#elementosA}{\#elementosLineaCache}$$

A[0][0] A[0][1] A[0][2] A[0][3]
consultas continuas en misma línea

Segundo Algoritmo

El recorrido de las consultas del arreglo es por elementos de una misma fila (i), lo que genera que el valor consultado no coincida con la línea de caché.
Se genera un cache miss con cada elemento nuevo que se consulta porque no corresponde con la línea de caché, por lo que la cantidad de cache miss que se incurrirán será la cantidad de elementos de A.

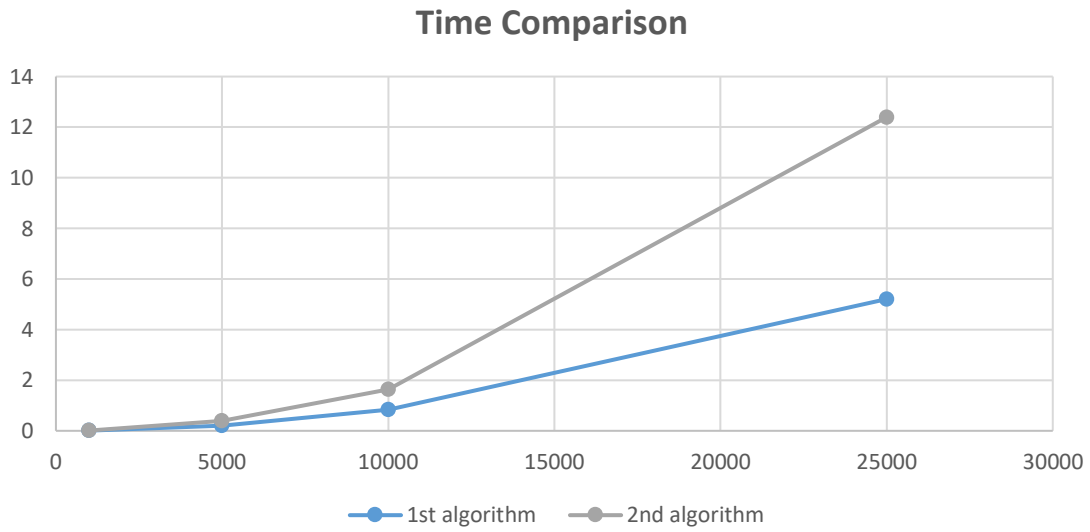
$\#cacheMiss =$ #elementosA

A[0][0]
A[1][0]
A[2][0]
A[3][0]
consultas por columna

III. RESULTADOS DEL EXPERIMENTO

Se implementó un código en C++ para comparar el desempeño de ambos algoritmos con diferentes tamaños de arreglo. El tiempo fue medido en segundos obteniendo los siguientes resultados:

	Time took (s)	
Array Size	1st algorithm	2nd algorithm
1000	0.008	0.01
5000	0.21	0.396
10000	0.834	1.631
25000	5.205	12.395



IV. CONCLUSIONES

Al haber evaluado una dimensión de 25 000 datos para el tamaño del arreglo, se evidencia que el primer algoritmo tiene un mejor uso de la memoria caché y menor tiempo de ejecución con respecto al segundo algoritmo.

Esto debido a que las consultas y acceso a valores del arreglo coinciden en mayor parte