# Instituto Politécnico Nacional.

# Escuela Superior De Cómputo.


**Materia:**

**Desarrollo de Sistemas Distribuidos.**

**Tema:**

**Uso eficiente de la memoria cache.**

**(Tarea 02)**

**Profesor:**

**Carlos Pineda Guerrero.**

**Alumno:**

**Mario Alberto Miranda Sandoval.**

**Grupo:**

**4CM5**

# Objetivo.

Observar el uso de la memoria cache a través de dos programas con diferente implementación, pero misma salida, observando como hacer uso eficiente de la memoria cache.

# Datos.

Marca: Acer.

Modelo: Aspire A515-51

Tamaño de la cache:

- L1-D cache: 32 Kb x 2.
- L1-I cache: 32 Kb x 2.
- L2 cache: 256 Kb x 2.
- L3 cache: 3 Mb

Tamaño de la RAM: 8 Gb.

# Código.

**MatrizMultiplica.java**

```java
public class MultiplicaMatriz {
  static int N = 1000;
  static int[][] A = new int[N][N];
  static int[][] B = new int[N][N];
  static int[][] C = new int[N][N];

  public static void main(String[] args) {
    long t1 = System.currentTimeMillis();

    for (int i = 0; i < N; i++)
      for (int j = 0; j < N; j++) {
        A[i][j] = 2 * i - j;
        B[i][j] = i + 2 * j;
        C[i][j] = 0;
      }

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

    long t2 = System.currentTimeMillis();
    System.out.println("Tiempo: " + (t2 - t1) + "ms");
  }
}
```

Para este código se puede observar que se hace una operación muy costosa como lo es la multiplicación de matrices, este código hace la operación de una manera normal sin considerar la memoria local o espacial, el código funciona de una manera muy simple, se crean las

matrices de tamaño N, antes de empezar a rellenar las matrices se toma una marca de tiempo, posteriormente las matrices se rellenan de acuerdo con los bucles, destacando la inicialización de todos los elementos de la matriz C (matriz resultado) a 0.

Posteriormente en tres bucles se procede a hacer el algoritmo de la multiplicación, al final se toma otra marca de tiempo, se restan y se muestra el tiempo de ejecución.

**MultiplicaMatriz_2.java**

```java
public class MultiplicaMatriz_2 {
  static int N = 1000;
  static int[][] A = new int[N][N];
  static int[][] B = new int[N][N];
  static int[][] C = new int[N][N];

  public static void main(String[] args)
  {
    long t1 = System.currentTimeMillis();

    for (int i = 0; i < N; i++)
      for (int j = 0; j < N; j++)
      {
        A[i][j] = 2 * i - j;
        B[i][j] = i + 2 * j;
        C[i][j] = 0;
      }

    for (int i = 0; i < N; i++)
      for (int j = 0; j < i; j++) {
        int x = B[i][j];
        B[i][j] = B[j][i];
        B[j][i] = x;
      }

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

    long t2 = System.currentTimeMillis();
    System.out.println("Tiempo: " + (t2 - t1) + "ms");
  }
}
```

Para la segunda versión del código se tiene casi los mismos pasos, solo que hay un cambio que va acerca de transponer la matriz B a modo que se lea de manera secuencial los renglones de esa matriz. Al igual que el programa anterior se toman dos marcas de tiempo antes y después de la ejecución para después desplegarse en pantalla.

## Comparación.

De manera interesante se tiene que el segundo programa es más rápido que el primero, esto a simple vista puede ser no tan claro ya que

si observamos el código podemos darnos cuenta de que ambos tienen una complejidad máxima de $O(n^3)$, mientras que el primer programa tiene un solo bloque con complejidad $O(n^2)$ el segundo tiene dos, pero ¿cómo es posible que el segundo bloque sea más eficiente que el primero teniendo más instrucciones?, la respuesta es sencilla, esto se debe al número de veces que se accede a la memoria cache, ya que Java almacena las matrices como renglones, entonces en el primer programa cuando se ejecuta la multiplicación va accediendo a los elementos por columna, es decir el primer elemento de la primer columna, posteriormente el primer elementos de la segunda columna así sucesivamente hasta llegar al primer elemento de la columna n, el problema con esto es que como ya se ha dicho al meter a la memoria cache renglones donde solo se accede un elemento para después sacarse y volver a meter otro renglón genera un aumento en el número de operaciones de la memoria cache y un desperdicio enorme, mientras que en el segundo programa al transponer la matriz, da la flexibilidad que al momento de hacer la multiplicación, se recorran todos los elementos de un renglón de manera secuencial, generando que la memoria cache tenga un menor número de operaciones que en el programa anterior, esto se puede apreciar en la siguiente tabla y gráfica.

| N | Matriz Multiplica (ms) | Matriz Multiplica 2 (ms) |
|---|---|---|
| 100 | 15 | 14 |
| 200 | 34 | 33 |
| 300 | 109 | 69 |
| 500 | 437 | 282 |
| 1000 | 4319 | 2213 |