



# Instituto Politécnico Nacional

## Escuela Superior de Cómputo

*Asignatura:*

Desarrollo de sistemas distribuidos

*Grupo:*4CM3

Tarea 2. Memoria cache

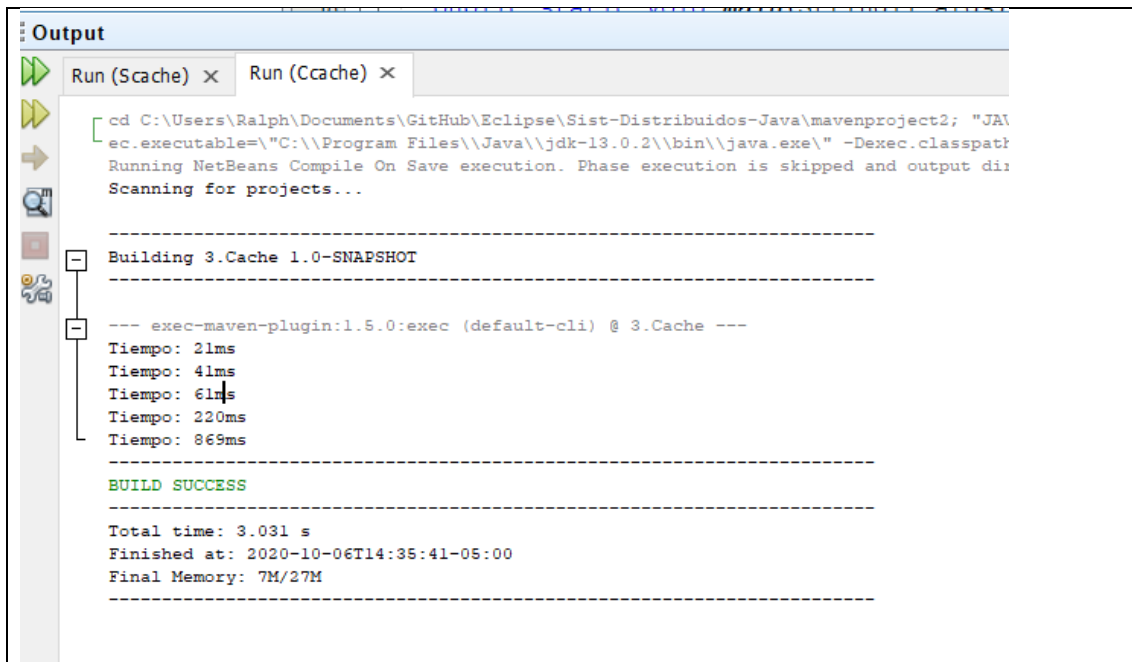
*Alumno:* Garcia Garcia Rafael

*Profesor:* Carlos Pineda Guerrero



## Capturas de pantalla de la tarea:

Lo primero a realizar es ejecutar nuestro programa para obtener los tiempos utilizando memoria caché o sin usarlo en mi caso decidí modificar un poco los programas para que el método *main* donde se realiza la multiplicación de matrices se sustituya un método llamado *multiplicar* y así crear un nuevo método *main* en donde haya un Array con los valores de prueba. Posterior a toda esta edición de código procedemos a ejecutarlos y compilarlos para tener en la salida de terminal los siguientes valores:



The screenshot shows the Eclipse IDE's Output window with the 'Run (Ccache)' tab selected. The terminal output displays the execution of a Maven build for '3.Cache 1.0-SNAPSHOT'. It lists five individual execution times: 21ms, 41ms, 61ms, 220ms, and 869ms, followed by a 'BUILD SUCCESS' message. The total time for the build is 3.031 seconds, finished at 2020-10-06T14:35:41-05:00, with a final memory usage of 7M/27M.

```
Output
Run (Scache) x Run (Ccache) x

cd C:\Users\Ralph\Documents\GitHub\Eclipse\Sist-Distribuidos-Java\mavenproject2; "JAV
ec.executable="C:\\Program Files\\Java\\jdk-13.0.2\\bin\\java.exe\" -Dexec.classpath
Running NetBeans Compile On Save execution. Phase execution is skipped and output di
Scanning for projects...

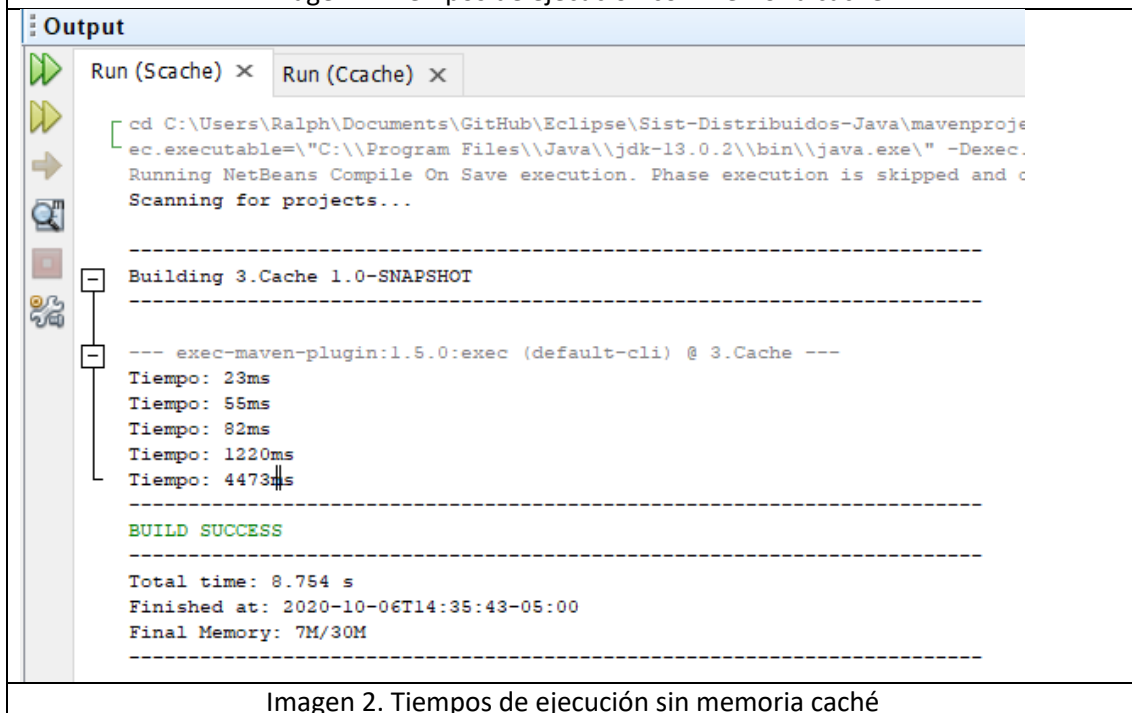
-----
Building 3.Cache 1.0-SNAPSHOT
-----

--- exec-maven-plugin:1.5.0:exec (default-cli) @ 3.Cache ---
Tiempo: 21ms
Tiempo: 41ms
Tiempo: 61ms
Tiempo: 220ms
Tiempo: 869ms

BUILD SUCCESS

Total time: 3.031 s
Finished at: 2020-10-06T14:35:41-05:00
Final Memory: 7M/27M
```

Imagen 1. Tiempos de ejecución con memoria caché



The screenshot shows the Eclipse IDE's Output window with the 'Run (Ccache)' tab selected. The terminal output displays the execution of a Maven build for '3.Cache 1.0-SNAPSHOT'. It lists five individual execution times: 23ms, 55ms, 82ms, 1220ms, and 4473ms, followed by a 'BUILD SUCCESS' message. The total time for the build is 8.754 seconds, finished at 2020-10-06T14:35:43-05:00, with a final memory usage of 7M/30M.

```
Output
Run (Scache) x Run (Ccache) x

cd C:\Users\Ralph\Documents\GitHub\Eclipse\Sist-Distribuidos-Java\mavenproje
ec.executable="C:\\Program Files\\Java\\jdk-13.0.2\\bin\\java.exe\" -Dexec.
Running NetBeans Compile On Save execution. Phase execution is skipped and c
Scanning for projects...

-----
Building 3.Cache 1.0-SNAPSHOT
-----

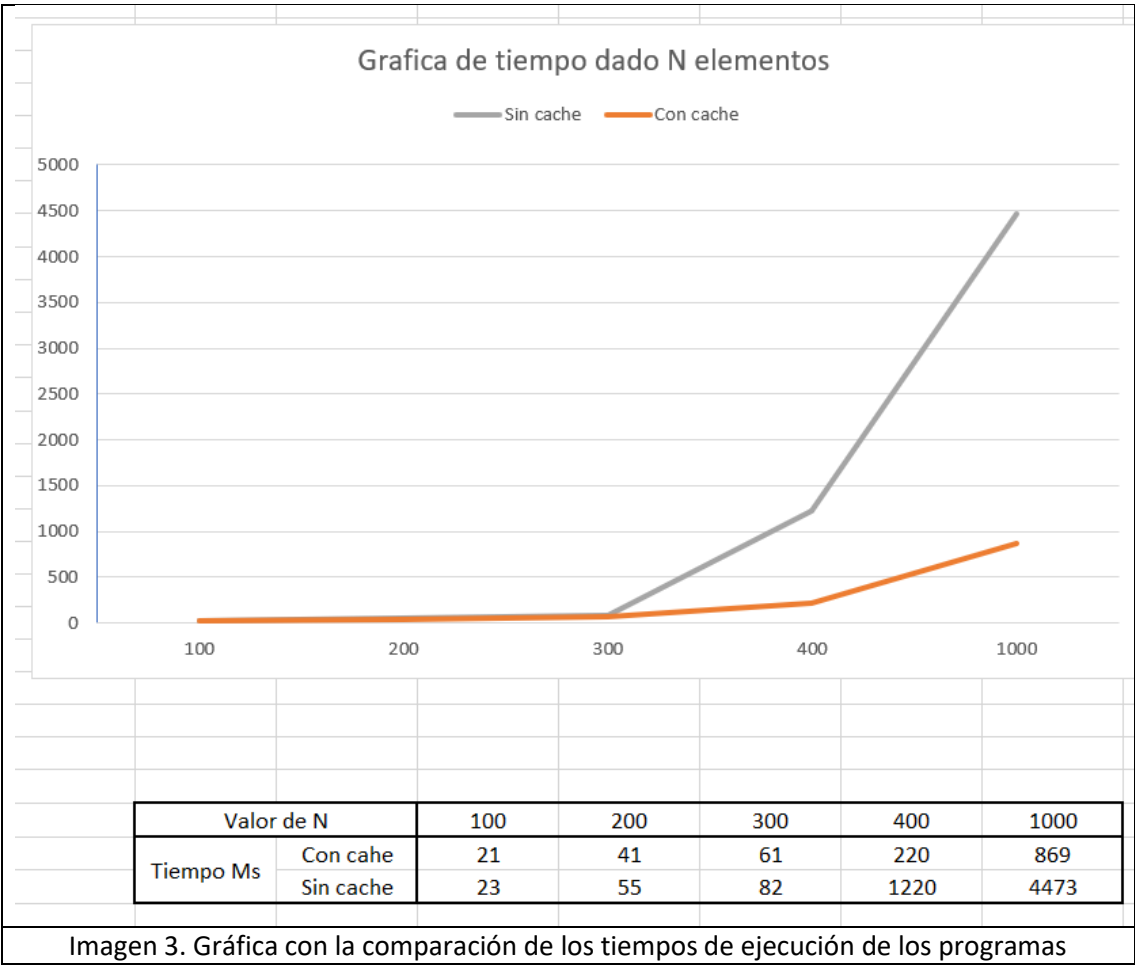
--- exec-maven-plugin:1.5.0:exec (default-cli) @ 3.Cache ---
Tiempo: 23ms
Tiempo: 55ms
Tiempo: 82ms
Tiempo: 1220ms
Tiempo: 4473ms

BUILD SUCCESS

Total time: 8.754 s
Finished at: 2020-10-06T14:35:43-05:00
Final Memory: 7M/30M
```

Imagen 2. Tiempos de ejecución sin memoria caché

Observando las terminales de Netbeans podemos ver una enorme reducción de tiempo en valores grandes ya que el acceso de memoria es mucho más rápido. Graficando estos valores en Exel podemos ver los siguientes resultados:



Como observamos en la gráfica el uso de la memoria caché es de vital importancia para el manejo de grandes cantidades de datos, además de como ingenieros poder entender entre los conceptos de localidad espacial y temporal ya que este tipo de acceso en la memoria puede ayudarnos a reducir tiempos de ejecución en nuestros programas.

Ahora expliquemos cada uno de los programas en las siguientes partes:

Programa sin memoria caché:

```
1. public class Scache
2. {
3.     static int N = 1000;
4.     static int[][] A = new int[N][N];
5.     static int[][] B = new int[N][N];
6.     static int[][] C = new int[N][N];
7.
8.     static int multiplicar(int N){
9.         long t1 = System.currentTimeMillis();
10.
11.         // inicializa las matrices A y B
12.
13.         for (int i = 0; i < N; i++)
```

```

14.     for (int j = 0; j < N; j++)
15.     {
16.         A[i][j] = 2 * i - j;
17.         B[i][j] = i + 2 * j;
18.         C[i][j] = 0;
19.     }
20.
21.     // multiplica la matriz A y la matriz B, el resultado queda en la matriz C
22.
23.     for (int i = 0; i < N; i++)
24.         for (int j = 0; j < N; j++)
25.             for (int k = 0; k < N; k++)
26.                 C[i][j] += A[i][k] * B[k][j];
27.
28.     long t2 = System.currentTimeMillis();
29.     System.out.println("Tiempo: " + (t2 - t1) + "ms");
30.     return N;
31. }
32.
33. public static void main(String[] args){
34.     int[] x = new int []{100, 200, 300, 500,1000};
35.     for (int i=0;i!=5;i++){
36.         multiplicar(x[i]);
37.     }
38. }
39. }

```

En la primera parte del programa inicializamos nuestras matrices en los atributos de la clase principal, además de que se sustituye el método **main** por el método **multiplicar** para que podamos ingresar los valores con un array y tener los resultados de los tiempos en una sola ejecución del programa, en el método multiplicar estamos multiplicando las matrices sin hacer uso de la memoria caché agarrando los elementos de la matriz de manera que no se utiliza localidad espacial.

## Memoria con caché:

```

1.  /**
2.   *
3.   * @author Ralph
4.   */
5.  public class Ccache {
6.
7.      static int N = 1000;
8.      static int[][] A = new int[N][N];
9.      static int[][] B = new int[N][N];
10.     static int[][] C = new int[N][N];
11.
12.     public static int multiplicar(int N)
13.     {
14.         long t1 = System.currentTimeMillis();
15.
16.         // inicializa las matrices A y B
17.
18.         for (int i = 0; i < N; i++)
19.             for (int j = 0; j < N; j++)
20.             {
21.                 A[i][j] = 2 * i - j;
22.                 B[i][j] = i + 2 * j;
23.                 C[i][j] = 0;
24.             }

```


```

25.
26.     // transpone la matriz B, la matriz traspuesta queda en B
27.
28.     for (int i = 0; i < N; i++)
29.         for (int j = 0; j < i; j++)
30.             {
31.                 int x = B[i][j];
32.                 B[i][j] = B[j][i];
33.                 B[j][i] = x;
34.             }
35.
36.     // multiplica la matriz A y la matriz B, el resultado queda en la matriz C
37.
38.     // notar que los indices de la matriz B se han intercambiado
39.
40.     for (int i = 0; i < N; i++)
41.         for (int j = 0; j < N; j++)
42.             for (int k = 0; k < N; k++)
43.                 C[i][j] += A[i][k] * B[j][k];
44.
45.     long t2 = System.currentTimeMillis();
46.     System.out.println("Tiempo: " + (t2 - t1) + "ms");
47.     return N;
48. }
49.
50. public static void main(String[] args){
51.     int[] x = new int []{100, 200, 300, 500,1000};
52.     for (int i=0;i!=5;i++){
53.         multiplicar(x[i]);
54.     }
55. }


```

Como en el programa anterior sustituimos el método **main** por el método **multiplicar** para que podamos ingresar los valores a medir en un arreglo de enteros, ahora en el método multiplicar primero transponemos la matriz B para que los valores que existen tengan un acceso de memoria por localidad espacial, esto como podemos ver en las salidas de la terminal reduce bastante el tiempo de ejecución de nuestro programa.


Valores del sistema de ejecución:




**CPU**  
58% 2.97 GHz




**Memoria**  
9.1/15.9 GB (57%)



**Disco 0 (C:)**  
5%



**Wi-Fi**  
Wi-Fi  
E: 360 R: 600 Kbps



**GPU 0**  
Intel(R) HD Graphics ...  
9%

