INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

ING. SISTEMAS COMPUTACIONALES

DESARROLLO DE SISTEMAS DISTRIBUIDOS

PINEDA GUERRERO CARLOS

TAREA #2

USO EFICIENTE DE LA MEMORIA CACHÉ

FECHA DE REALIZACIÓN: 10/03/2021

FECHA DE ENTREGA: 12/03/2021

GRUPO: 4CM3

ELABORO:

PÉREZ FEDERICO JOSÉ JOEL

INTRODUCCIÓN

La caché acelera el acceso a los datos que presentan localidad espacial y/o localidad temporal. Sin embargo, los algoritmos no siempre están diseñados para acceder a los datos de manera que se privilegie el acceso a la memoria en forma secuencial o su acceso a la memoria en forma dispersa.

El programa MultiplicaMatriz.java realiza la multiplicación de dos matrices cuadradas A y B utilizando el algoritmo estándar (renglón por columna) en este caso las matrices tienen un tamaño de 1000 por 1000.

Como Java almacena las matrices en la memoria como renglones el acceso a la matriz B, que en este programa es por columnas, es muy ineficiente si las matrices son muy grandes, ya que cada vez que se accede a un elemento de la matriz B se transfiere una línea de caché completa de la memoria RAM a la cache.

MultiplicaMatriz.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();

        //Inicializacion de matrices
        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;
            }
        }

        //Multiplicar Matrices
        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");
    }

}
```

Si modificamos el algoritmo de multiplicación de matrices de manera que incrementemos la localidad espacial haciendo que el acceso a la matriz B sea por renglones como se presenta en el programa MultiplicaMatriz2.java; Lo que se hizo fue intercambiar los índices que se usaron para acceder a los elementos de la matriz B la cual previamente se ha transpuesto ya que es necesario transponer la matriz B para que el algoritmo siga calculando el producto de las matrices.

MultiplicaMatriz2.java

```
public class MultiplicaMatriz2 {
    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();

        //Inicializacion de matrices A y B
        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;
            }
        }

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

        //Multiplicar Matrices
        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");
    }

}
```

De esta manera obtenemos un acceso más eficiente a los elementos de la matriz B debido a que ahora se leen los elementos debe en forma secuencial lo cual aumenta la localidad espacial y temporal de los datos.

RESULTADOS

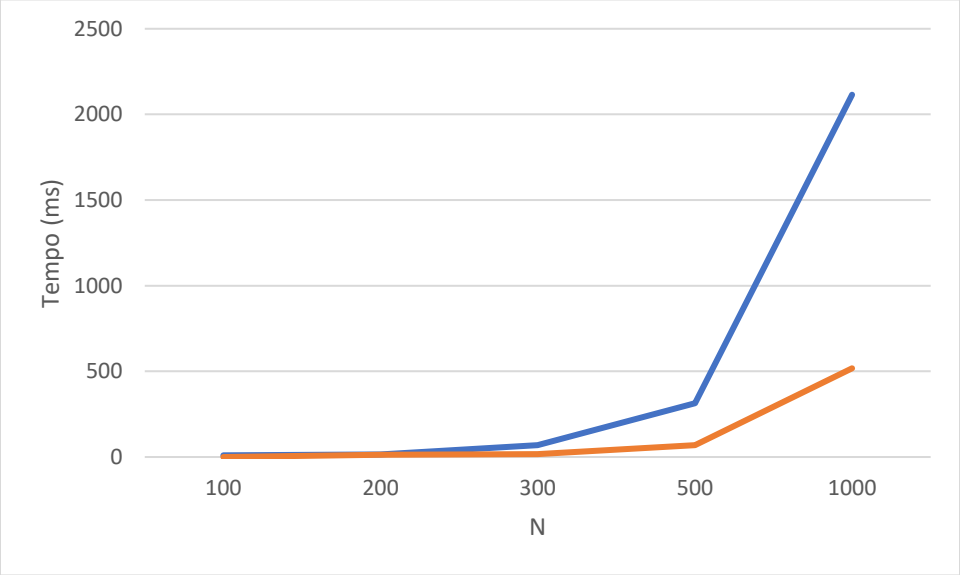
Al ejecutar ambos programas para diferentes tamaños de las matrices observamos una diferencia en tiempo de ejecución como observamos en la Tabla 1, ya que el algoritmo que accede a ambas matrices por renglones (MultiplicaMatriz.java) es mucho más eficiente ya que en este algoritmo la localidad espacial y la localidad temporal de los datos es mayor debido a que ambas matrices, A y B, son accedidas por renglones tal como se almacenan en la memoria por Java.

Tabla 1. *Tiempos de Ejecución*

N	Tiempo (ms) MultiplicaMatriz	Tiempo (ms) MultiplicaMatriz2
100	10	0
200	16	14
300	69	17
500	314	69
1000	2115	518

A continuación, se presenta una grafica con ambos resultados para una mejor comparativa. La línea azul representa los tiempos del primer programa y la línea naranja los tiempos del segundo programa con acceso eficiente a memoria.

Figura 1. *Grafica de dispersión con líneas de los tiempos de ejecución del programa MultiplicaMatriz y MultiplicaMatriz2*



De la Figura 1 podemos observar como el tiempo se reduce sustancialmente en el segundo programa al momento de incrementar el tamaño de la matriz.

Los datos del ordenador en el cual se ejecutaros los programas son los siguientes:

Marca: Microsoft

Procesador: Intel(R) Core(TM) i7-7660U CPU @ 2.50GHz 2.50 GHz

RAM instalada: 8.00 GB

Tipo de sistema: Sistema operativo de 64 bits, procesador x64

CONCLUSIONES

Como se pudo observar con los programas MultiplicaMatriz.java y MultiplicaMatriz2.java el no considerar como accedemos a memoria a la hora de hacer un programa afecta en el tiempo de ejecución del mismo. En este caso el problema a resolver fue la multiplicación de dos matrices y observé como al ir aumentando el tamaño de la matriz el tiempo de ejecución se prolongó demasiado, caso contrario cuando como es que java accede a información en memoria, se puede mejorar la eficiencia del algoritmo considerando que el tiempo de recuperación de la información es más rápido si tiene localidad espacial y/o temporal, como se observó en la Tabla 1.