

Tarea 2: Uso eficiente de la memoria cache

Pulido Bejarano Raymundo

6 de Octubre del 2020

1 Código fuente de los programas

1.1 Con N=1000

```
class MultiplicacionMatriz {
    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();
        //inicializamos las 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;
            }
        }

        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");
    }
}
```

Código sin optimizacion de uso de Cache y con N= 1000

```

class MultiplicacionMatriz_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();
        //inicializamos las 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;
            }
        }

        // transpone la matriz B, la matriz traspuesta queda
        en B

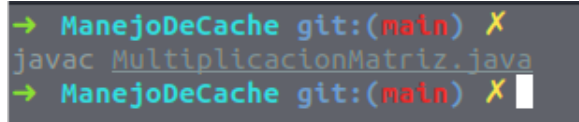
        for (int i = 0; i < N; i++)
            for (int j = 0; j < i; j++){
                int x = B[i][j];
                B[i][j] = B[j][i];
                B[j][i] = x;
            }

        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");
    }
}

```

Código con optimizacion de uso de Cache y con N= 1000



```

→ ManejoDeCache git:(main) X
javac MultiplicacionMatriz.java
→ ManejoDeCache git:(main) X

```

Figure 1: Compilación N= 1000, sin optimizacion

```

→ ManejoDeCache git:(main) X
javac MultiplicacionMatriz_2.java
→ ManejoDeCache git:(main) X

```

Figure 2: Compilación N= 1000, con optimizacion

```

→ ManejoDeCache git:(main) X
java MultiplicacionMatriz
Tiempo: 2024ms
→ ManejoDeCache git:(main) X

```

Figure 3: Ejecución N= 1000, sin optimizacion

```

→ ManejoDeCache git:(main) X
java MultiplicacionMatriz_2
Tiempo: 485ms
→ ManejoDeCache git:(main) X

```

Figure 4: Ejecución N= 1000, con optimizacion

En este programa se ve la implementacion de la multiplicacion de matrices de 1000 x 1000, usando una optimizacion de cache y una sin el, se puede notar que hay una correspondencia de 1 a 1/4 en el tiempo, es decir 2024ms a 485ms

1.2 Con N=500

```

class MultiplicacionMatriz {
    static int N = 500;
    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();
        //inicializamos las 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;
            }
        }
    }
}

```

```

        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");
    }
}

```

Código sin optimización de uso de Cache y con $N=500$

```

class MultiplicacionMatriz_2 {
    static int N = 500;
    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();
        //inicializamos las 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;
            }
        }

        // transpone la matriz B, la matriz traspuesta queda
        // en B

        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");
    }
}

```

Código con optimizacion de uso de Cache y con N= 500

```

→ ManejoDeCache git:(main) X
javac MultiplicacionMatriz.java
→ ManejoDeCache git:(main) X

```

Figure 5: Compilación N= 500, sin optimizacion

```

→ ManejoDeCache git:(main) X
javac MultiplicacionMatriz_2.java
→ ManejoDeCache git:(main) X

```

Figure 6: Compilación N= 500, con optimizacion

```

→ ManejoDeCache git:(main) X
java MultiplicacionMatriz
Tiempo: 176ms
→ ManejoDeCache git:(main) X

```

Figure 7: Ejecución N= 500, sin optimizacion

```

→ ManejoDeCache git:(main) X
java MultiplicacionMatriz_2
Tiempo: 79ms
→ ManejoDeCache git:(main) X

```

Figure 8: Ejecución N= 500, con optimizacion

En este programa se ve la implementacion de la multiplicacion de matrices de 500 x 500, usando una optimizacion de cache y una sin el, se puede notar que hay una correspondencia de 1 a 1/2 en el tiempo, es decir 176ms a 79ms

1.3 Con N=300

```

class MultiplicacionMatriz {
    static int N = 300;
    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();
        //inicializamos las 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;
            }
        }

        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");
    }
}

```

Código sin optimizacion de uso de Cache y con N= 300

```

class MultiplicacionMatriz_2 {
    static int N = 300;
    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();
        //inicializamos las 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;
            }
        }
    }
}

```

```

        // transpone la matriz B, la matriz traspuesta queda
        // en B

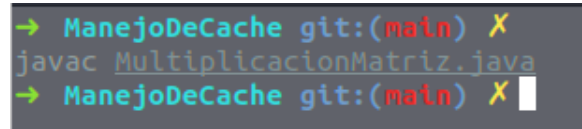
        for (int i = 0; i < N; i++)
            for (int j = 0; j < i; j++){
                int x = B[i][j];
                B[i][j] = B[j][i];
                B[j][i] = x;
            }

        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");
    }
}

```

Código con optimización de uso de Cache y con $N=300$

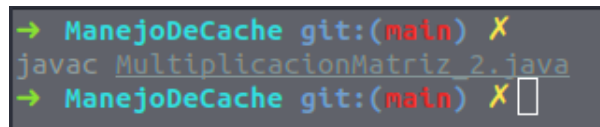


```

→ ManejoDeCache git:(main) X
javac MultiplicacionMatriz.java
→ ManejoDeCache git:(main) X

```

Figure 9: Compilación $N=300$, sin optimización



```

→ ManejoDeCache git:(main) X
javac MultiplicacionMatriz_2.java
→ ManejoDeCache git:(main) X

```

Figure 10: Compilación $N=300$, con optimización

```

→ ManejoDeCache git:(main) X
java MultiplicacionMatriz
Tiempo: 48ms
→ ManejoDeCache git:(main) X

```

Figure 11: Ejecución N= 300, sin optimizacion

```

→ ManejoDeCache git:(main) X
java MultiplicacionMatriz_2
Tiempo: 30ms
→ ManejoDeCache git:(main) X

```

Figure 12: Ejecución N=300, con optimizacion

En este programa se ve la implementacion de la multiplicacion de matrices de 300 x 300, usando una optimizacion de cache y una sin el, se puede notar que hay una correspondencia de 1 a 3/5 en el tiempo, es decir 48ms a 30ms

1.4 Con N=200

```

class MultiplicacionMatriz {
    static int N = 200;
    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();
        //inicializamos las 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;
            }
        }

        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");
    }
}

```



```

    }
}

```

Código sin optimizacion de uso de Cache y con N= 200

```

class MultiplicacionMatriz_2 {
    static int N = 200;
    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();
        //inicializamos las 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;
            }
        }

        // transpone la matriz B, la matriz traspuesta queda
        en B

        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");
    }
}

```

Código con optimizacion de uso de Cache y con N= 200

```

→ ManejoDeCache git:(main) X
javac MultiplicacionMatriz.java
→ ManejoDeCache git:(main) X

```

Figure 13: Compilación N= 200, sin optimizacion

```

→ ManejoDeCache git:(main) X
javac MultiplicacionMatriz_2.java
→ ManejoDeCache git:(main) X

```

Figure 14: Compilación N= 200, con optimizacion

```

→ ManejoDeCache git:(main) X
java MultiplicacionMatriz
Tiempo: 22ms
→ ManejoDeCache git:(main) X

```

Figure 15: Ejecución N= 200, sin optimizacion

```

→ ManejoDeCache git:(main) X
java MultiplicacionMatriz_2
Tiempo: 22ms
→ ManejoDeCache git:(main) X

```

Figure 16: Ejecución N=200, con optimizacion

En este programa se ve la implementacion de la multiplicacion de matrices de 200 x 200, usando una optimizacion de cache y una sin el, Con este tamaño de matrices no se puede ver una diferencia en rendimiento

1.5 Con N=100

```

class MultiplicacionMatriz {
    static int N = 100;
    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();
        //inicializamos las 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;
        }
    }

    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");
}

}

```

Código sin optimizacion de uso de Cache y con N= 100

```

class MultiplicacionMatriz_2 {
    static int N = 100;
    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();
        //inicializamos las 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;
            }
        }

        // transpone la matriz B, la matriz traspuesta queda
        en B

        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;
            }
    }
}

```

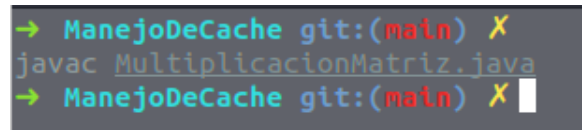
```

        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");
    }
}

```

Código con optimizacion de uso de Cache y con N= 100

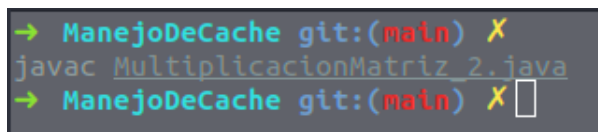


```

→ ManejoDeCache git:(main) X
javac MultiplicacionMatriz.java
→ ManejoDeCache git:(main) X

```

Figure 17: Compilación N= 100, sin optimizacion

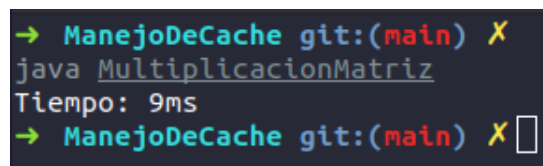


```

→ ManejoDeCache git:(main) X
javac MultiplicacionMatriz_2.java
→ ManejoDeCache git:(main) X

```

Figure 18: Compilación N= 100, con optimizacion



```

→ ManejoDeCache git:(main) X
java MultiplicacionMatriz
Tiempo: 9ms
→ ManejoDeCache git:(main) X

```

Figure 19: Ejecución N=100, sin optimizacion

En este programa se ve la implementacion de la multiplicacion de matrices de 100 x 100, usando una optimizacion de cache y una sin el, Con este tamaño de matrices no se puede ver una diferencia en rendimiento

```

→ ManejoDeCache git:(main) X
java MultiplicacionMatriz_2
Tiempo: 9ms
→ ManejoDeCache git:(main) X

```

Figure 20: Ejecución N=100, con optimizacion

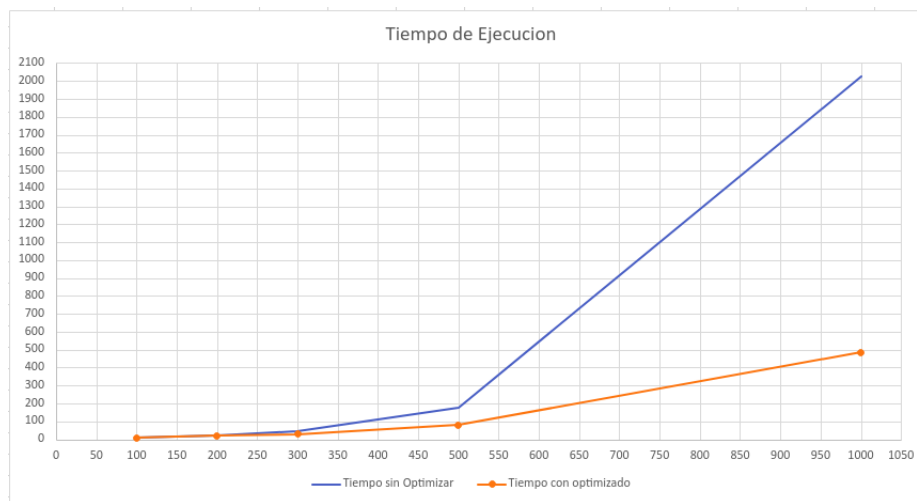


Figure 21: Ejecución N=100, con optimizacion

2 Grafica de dispersión

3 Datos del CPU

- AMD Ryzen 5 3500U - 6 Nucleos con 16MB de caché L3 - 8 RAM