PRÁCTICA 2

1. INTRODUCCIÓN

Se plantea la realización de un análisis de la técnica de optimización de código conocida como "desenrolle de lazos internos y optimización de reducciones". En este informe se explicará en qué consiste esta técnica de optimización y los beneficios que aporta, además de realizar una demostración del funcionamiento de dicha técnica mediante la comparación de los tiempos de ejecución de dos fragmentos de código, uno sin optimizar y otro optimizado, que realizan la misma operación.

2. ANÁLISIS

En primer lugar, se comenzará describiendo el funcionamiento de la técnica de optimización asignada, desenrolle de lazos internos y optimización de reducciones.

DESENROLLE DE LAZOS INTERNOS

Este método consiste en incrementar el rendimiento de un programa a costa de aumentar el número de líneas del código fuente. Para ello, se reduce el número de iteraciones de un bucle y se modifica su contenido para que las operaciones realizadas muestren el mismo resultado, como se puede observar en el siguiente ejemplo:

```
int i, k;
float x[N];
float a;
for(k=0; k<ITER; k++){
   a=1.0;
   for(i=0; i<N; i++)
      a = a * x[i]; }
```

Ilustración 1: código sin optimizar

```
float a0, a1, a2, a3
for(k=0; k<ITER; k++){
   a = 1.0;
   for(i=0; i<N; i+=4){
      a0 = x[i];
      a1 = a0 * x[i+1];
      a2 = x[i+2];
      a3 = a2 * x[i+3];
      a = a * a1 * a3;
      }
   }
```

Ilustración 2: código optimizado

Se puede apreciar fácilmente que en el último fragmento de código las iteraciones realizadas por el bucle *for* se reducen significativamente, pasando de ser *N* a *N/4*. Esta optimización se podría realizar también con otros valores, siendo estudiada la influencia de la profundidad (*d*) en la siguiente sección.

Puesto que se conoce que los segmentos de código que más tiempo tardan en ejecutarse suelen ser los bucles *for* que realizan un gran número de iteraciones, al reducir el número de dichas iteraciones se consigue una disminución notable del tiempo de ejecución del programa.

OPTIMIZACIÓN DE REDUCCIONES

Una vez resuelto el problema de las iteraciones, es hora de trabajar con las operaciones internas del bucle. En este caso, como la profundidad es cuatro, será necesario encontrar el valor de la variable *a* en dicha iteración. La forma más intuitiva e inteligible de hacerlo es la siguiente:

```
a = 1.0;
for (i = 0; i < N; i += 4)
{
    a0 = a * x[i];
    a1 = a0 * x[i + 1];
    a2 = a1 * x[i + 2];
    a3 = a2 * x[i + 3];
    a = a3;
}
```

Ilustración 3

Aunque esta implementación es correcta, está claro que no es muy eficiente. Basándose en la propiedad conmutativa de la multiplicación, es posible alterar el orden en el que se realizan las

operaciones y mejorar el rendimiento. Como es lo mismo realizar una multiplicación antes o después y el valor de cada elemento depende del elemento anterior, se pueden agrupar algunas multiplicaciones de la forma mostrada en la ilustración 2, sustituyendo dos multiplicaciones por operaciones de asignación y añadiendo solo una multiplicación más. De esta manera, aunque el número total de multiplicaciones es el mismo, se reducen las dependencias entre las instrucciones, pues no es necesario conocer los valores de a_1 y a_3 hasta la última línea. Esto hace que el procesador pueda paralelizar las instrucciones, aumentando el rendimiento y reduciéndose el tiempo de ejecución del programa.

Observando el código en ensamblador generado por el compilador, es posible verificar que se está aplicando la técnica de optimización anteriormente descrita. En este caso, es bastante sencillo realizar esta comprobación, pues bastará con buscar en el código optimizado una secuencia de instrucciones similares a las presentes en el código sin optimizar que se repita cuatro veces, ya que el valor de *d* en este caso es cuatro.

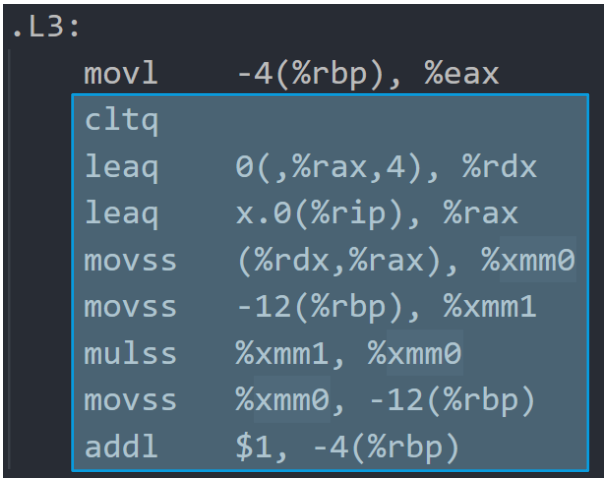


Ilustración 4: fragmento de código en ensamblador sin optimizar

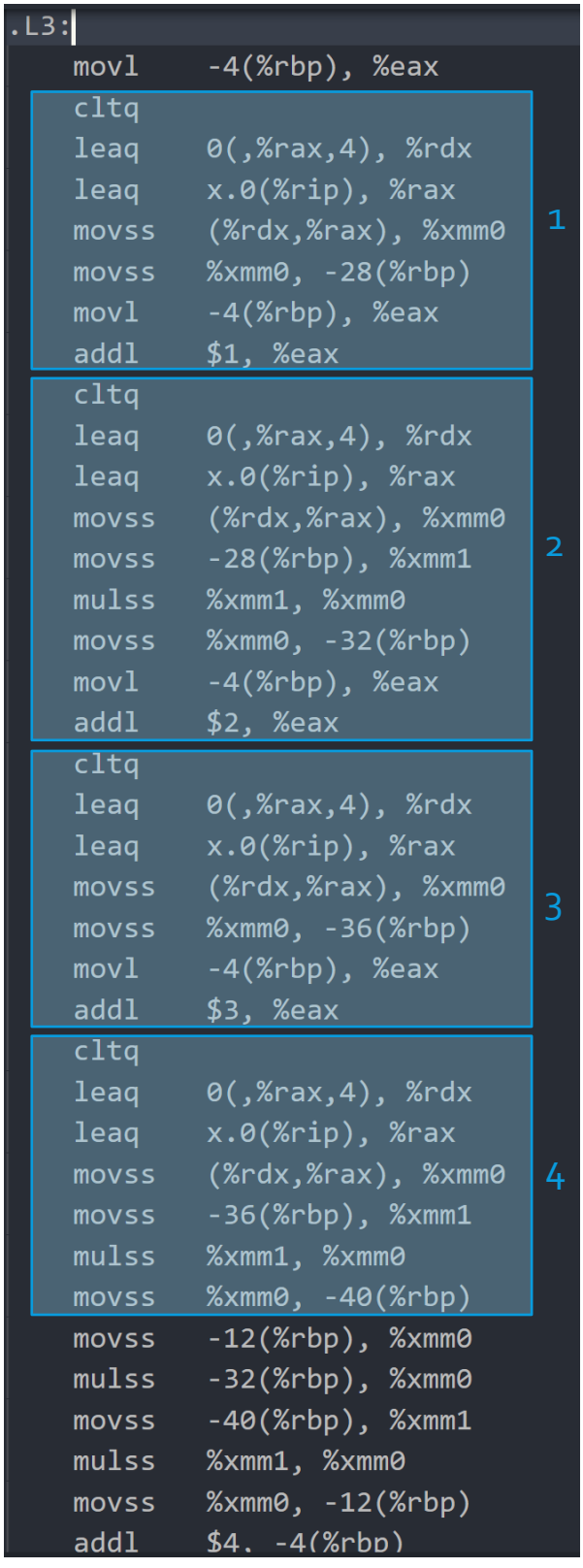


Ilustración 5: fragmento de código en ensamblador optimizado

3. RESULTADOS

Tras comprender la naturaleza del problema y hacer cábalas sobre los resultados esperados, se procederá a medir los tiempos de ejecución de la versión optimizada y de la versión sin optimizar, estudiando la influencia del tamaño del problema (*N*) y de la profundidad (*d*). En la columna *tiempo* de la tabla se recoge la media aritmética de los resultados obtenidos en cuatro ejecuciones del mismo programa.

TABLA 1: CÓDIGO SIN OPTIMIZAR

N	ITER	Tiempo (s)	**Tiempo/ITER**
10	1.000.000.000	18,975911	1,90E-08
100	100.000.000	27,763852	2,78E-07
1.000	10.000.000	29,097568	2,91E-06
10.000	1.000.000	28,642792	2,86E-05
100.000	100.000	28,798384	0,00028798
1.000.000	10.000	30,677902	0,00306779
10.000.000	1.000	29,353813	0,02935381
100.000.000	100	36,044811	0,36044811

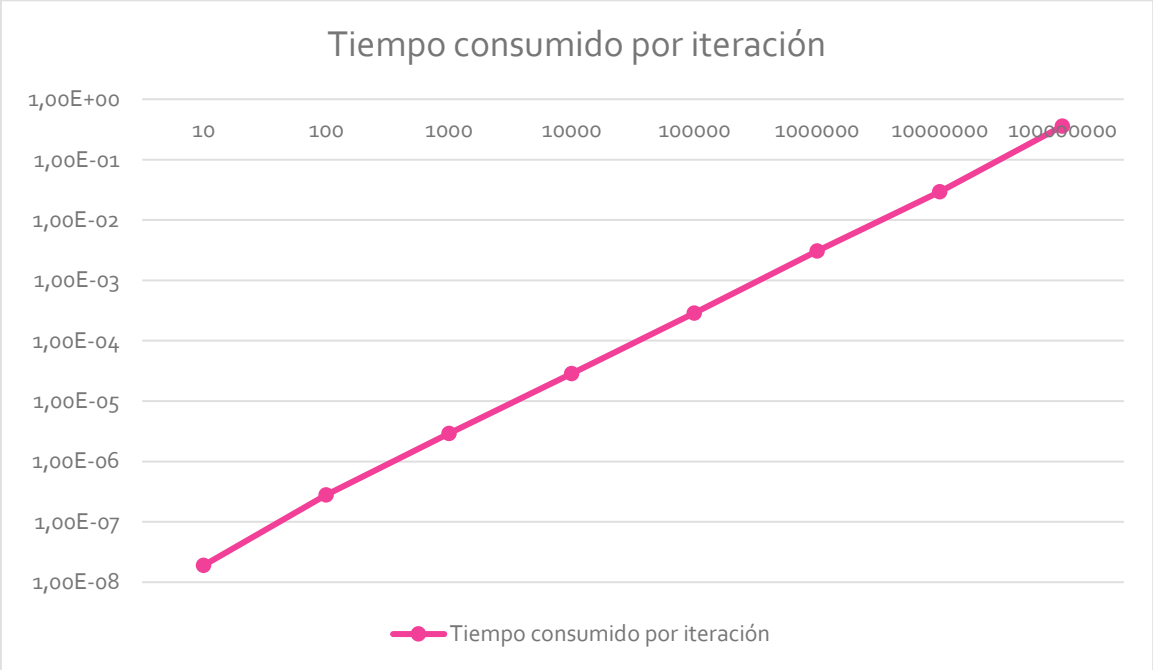
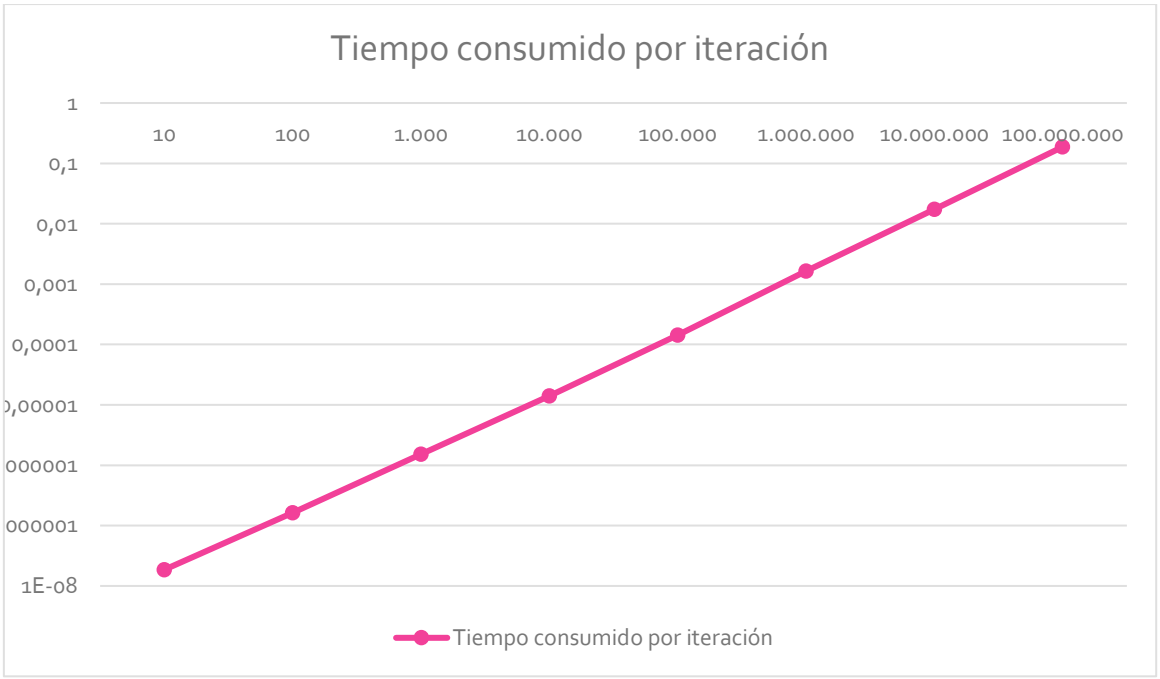


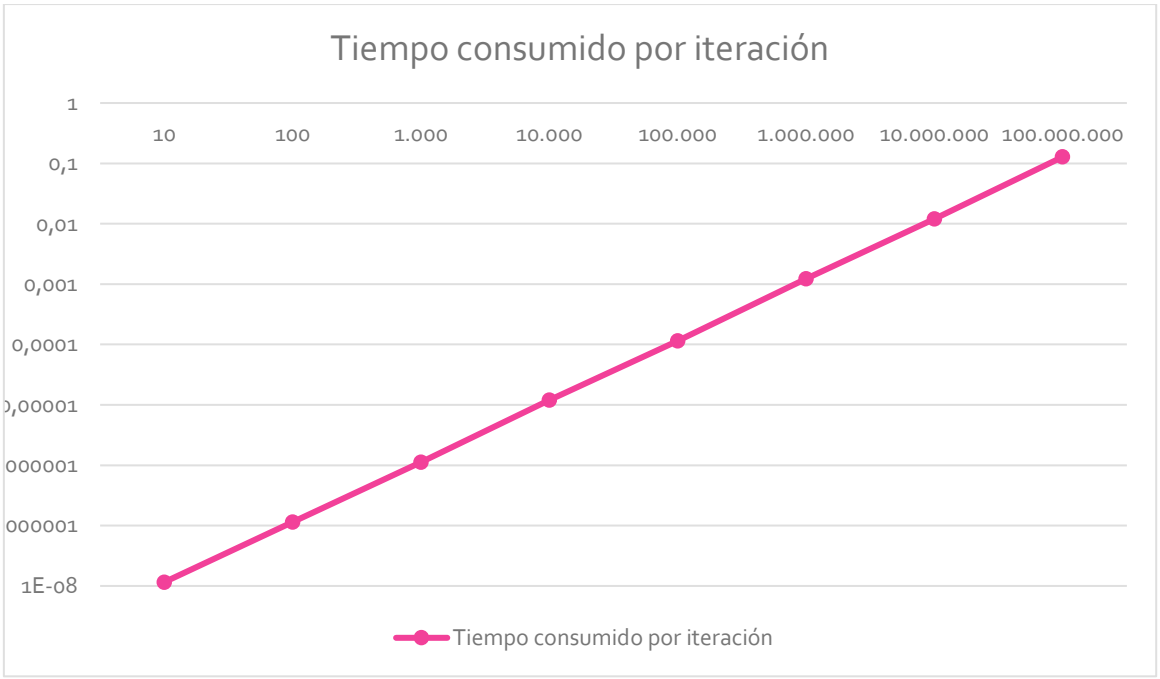
TABLA 2: CÓDIGO OPTIMIZADO

N	ITER	D	Tiempo (s)	**Tiempo/ITER**
10	1.000.000.000	4	12,782060	1,27821E-08
100	100.000.000	4	11,86992	1,18699E-07
1.000	10.000.000	4	10,543490	1,05435E-06
10.000	1.000.000	4	10,461141	1,04611E-05
100.000	100.000	4	10,653865	0,000106539
1.000.000	10.000	4	11,637233	0,001163723
10.000.000	1.000	4	11,838914	0,011838914
100.000.000	100	4	11,781451	0,11781451

N	ITER	D	Tiempo (s)	**Tiempo/ITER**
10	1.000.000.000	3	18,588109	1,85881E-08
100	100.000.000	3	16,359825	1,63598E-07
1.000	10.000.000	3	15,248357	1,52484E-06
10.000	1.000.000	3	14,156832	1,41568E-05
100.000	100.000	3	14,356044	0,00014356
1.000.000	10.000	3	16,369119	0,001636912
10.000.000	1.000	3	17,307834	0,017307834
100.000.000	100	3	18,772339	0,18772339



N	ITER	D	Tiempo (s)	**Tiempo/ITER**
10	1.000.000.000	5	11,567844	1,15678E-08
100	100.000.000	5	11,439638	1,14396E-07
1.000	10.000.000	5	11,287983	1,1288E-06
10.000	1.000.000	5	11,937958	1,1938E-05
100.000	100.000	5	11,539156	0,000115392
1.000.000	10.000	5	12,306839	0,001230684
10.000.000	1.000	5	12,003545	0,012003545
100.000.000	100	5	12,804534	0,12804534



4. CONCLUSIONES

INFLUENCIA DEL TAMAÑO DEL PROBLEMA (*N*)

El valor que será evaluado para analizar la influencia de la variable *N* es el tiempo en segundos dividido entre el número de iteraciones realizadas, es decir, el tiempo que lleva recorrer por completo el vector de tamaño *N* y realizar las operaciones que hay dentro del bucle. Puesto que este factor presenta un crecimiento exponencial, para visualizar mejor los valores se emplearon gráficas en escala logarítmica en base 10.

Fijándose en los resultados obtenidos, y tras constatar el crecimiento exponencial del tiempo en función de la variable *N*, se puede concluir que el tamaño del problema tiene una influencia notable en el tiempo de ejecución, sobre todo cuando se trabaja con tamaños grandes. Este problema puede ser atajado desenrollando el bucle que recorre el vector de tamaño *N*, reduciendo así el número de iteraciones y, por tanto, el tiempo de ejecución del programa. Los resultados obtenidos confirman esta hipótesis, pues los tiempos de ejecución obtenidos con el código optimizado son considerablemente inferiores a los que se obtuvieron con el código sin optimizar, llegando a reducirse en más de un 50% (con *d*=4).

INFLUENCIA DE LA PROFUNDIDAD (*D*)

Como se comentó en la sección inicial, la profundidad (*d*) es el factor que establece el número de elementos que se "desenrollarán" del bucle principal. Así, el número final de iteraciones que se realizarán para recorrer un vector de tamaño *N* se obtiene mediante la expresión *N/d*.

Es obvio que, para cualquier valor de *d*, el tiempo de ejecución de un fragmento de código optimizado va a ser notablemente inferior al que se obtendría al ejecutar el mismo código sin optimizar (se puede

comprobar esto observando las tablas y las gráficas del apartado anterior). Sin embargo, también se podría inferir que el tiempo de ejecución se iría reduciendo a medida que aumenta la *d*, pero la estrategia de optimización de reducciones desempeña también un importante papel en la obtención de los resultados finales. De esta manera, sí es cierto que un valor de *d* más grande implicará que se realice un menor número de iteraciones, pero los mejores resultados se obtienen con valores de *d* pares, debiéndose esto a la naturaleza de esta estrategia de optimización. Es posible contrastar esta conclusión comparando las medidas tomadas con *d*=3 y *d*=5: ambas son peores que las obtenidas con *d*=4, en el primer caso porque el valor de *d* es inferior y en el segundo porque, aunque es superior, es impar. No se construyó una tabla con los valores obtenidos con *d*=6, pero sí se comprobó que, como era de esperar, los tiempos eran inferiores a todos los que se habían medido anteriormente (0,10497159 segundos por iteración con *N*=100.000.000).