Fusión de Lazos

La fusión de lazos es un tipo de optimización que como bien indica su nombre permite fusionar algunos bucles que son adyacentes con el fin de reducir la sobrecarga del bucle y mejorar el rendimiento en cuanto a tiempo de ejecución. Como gran parte de las optimizaciones, no siempre mejora el tiempo de ejecución llegando en algunas circunstancias incluso a empeorarlo. Esto puede producirse por motivos como el soporte que hace la arquitectura que se esté usando de estos bucles.
Para comprobar el funcionamiento de esta optimización, usaremos el siguiente código:

```
int i, j;
float x[N], y[N], z[N], k[N], t[N], u[N];
for(j=0; j<ITER; j++)
{
        for(i=0; i<N; i++)
        z[i] = x[i] + y[i];
        for(i=0; i<N; i++)
        k[i] = x[i] - y[i];
        for(i=0; i<N; i++)
        t[i] = x[i] * y[i];
        for(i=0; i<N; i++)
        u[i] = z[i] + k[i] + t[i];
}
```

Al cúal le aplicaremos la fusión de lazos y convertiremos en:

```
float tmpx, tmpy;
for(j=0; j<ITER; j++)
        for(i=0; i<N; i++)
        {
                tmpx = x[i];
                tmpy = y[i];
                z[i] = tmpx + tmpy;
                k[i] = tmpx - tmpy;
                t[i] = tmpx * tmpy;
                u[i] = z[i] + k[i] + t[i];
        }
```

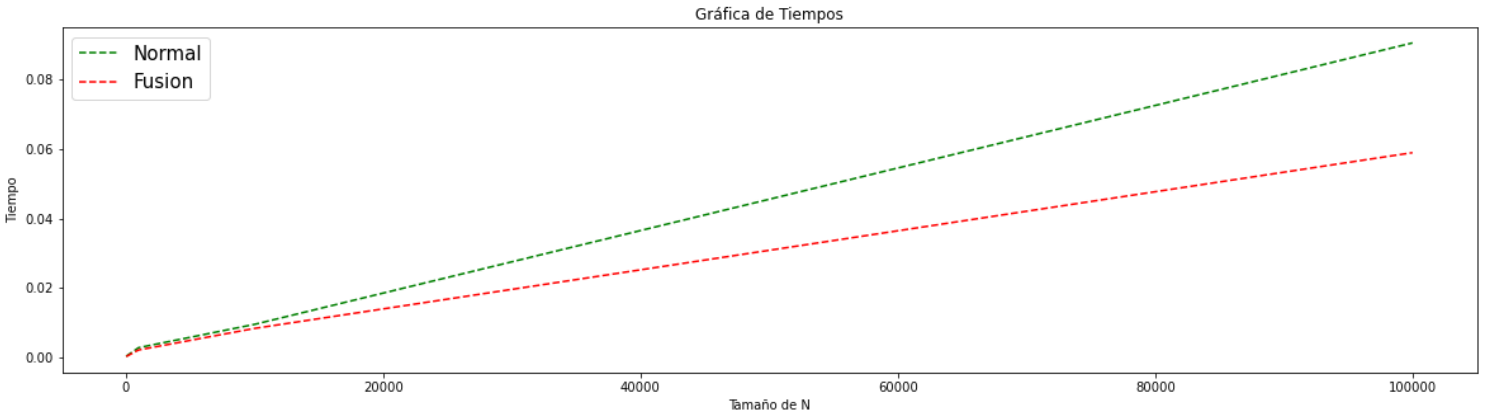
Los detalles técnicos del dispositivo y el compilador usado son los siguientes:

- Sistema operativo: Linux Mint 19.2 Cinnamon
- Versión cinnamon: 4.2.3
- Memoria RAM: 8 GiB
- CPU: Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz
- 3 niveles de caché: 7,256 MiB
- Arquitectura: x86_64
- gcc: (Ubuntu 7.5.0-3ubuntu1~18.04) 7.5.0

Las pruebas se realizarán de la siguiente manera. Se ejecutarán ambos códigos mediante un script que lo ejecutará un total de 100 veces. A mayores, se probará también a medir los tiempos con distintos N. Los códigos se compilaron usando la opción -O0 para evitar que se realice ningún tipo de optimización a mayores, y se utiliza para todos ITER = 100.

N	Normal	Fusión
50	0.0002485700	0.0001869400
100	0.0005196000	0.0003941600
1000	0.0028113300	0.0021460500
10000	0.0095141800	0.0075709800
100000	0.0905444000	0.0589475500
1000000	Al llegar a este tamaño, es demasiado para el vector y se produce “core dumped” al estar usando memoria estática.	

Si lo representamos gráficamente podemos ver:



Como vemos en la gráfica, a mayor número de iteraciones hay en el bucle (o bucles), la mejora en tiempo es mayor para el código optimizado con la fusión de bucles. Si tenemos en cuenta que la optimización realizada permite reducir la sobrecarga entre bucles adyacentes, podemos sacar como conclusión que esta sobrecarga depende del tamaño del bucle, pues si no lo hiciera, la mejora debería haber sido constante para todos los casos.

Para obtener alguna pista a mayores del porqué, podemos echar un ojo al código ensamblador de ambos y compararlos. En el código ensamblador sin la optimización, podemos distinguir cuatro grupos de código de tamaño medio que se corresponden cada uno a uno de los bucles de N, sin embargo y como era de esperar, al analizar el código fusionado, solo encontraremos un gran bloque de código que realiza las acciones de los 4 que veíamos en el primero. El resto del código ensamblador se mantiene igual para ambos.

La mejora a medida que N crece debe producirse en la eliminación de los cálculos de cada bucle para contabilizar hasta N. Estos se reducen ya que al solo tener un bucle, únicamente es necesario hacer este cálculo una vez por iteración, mientras que antes se realizaba después de cada cálculo individual (ya que estaban separados en bucles). Por lo tanto antes se realizaban N * 4 * ITER operaciones de cuenta de bucle, mientras con la fusión se realizan N * ITER, un cuarto de las realizadas antes.

```
int i, j;
float  x[N], y[N], z[N], k[N], t[N], u[N];
for(j=0; j<ITER; j++){   ITER
   for(i=0; i<N; i++)  N
    z[i] = x[i] + y[i];
   for(i=0; i<N; i++)   N
    k[i] = x[i] - y[i];
   for(i=0; i<N; i++)  N
    t[i] = x[i] * y[i];
   for(i=0; i<N; i++)  N
    u[i] = z[i] + k[i] + t[i];
   }
```

ITER * (N+N+N+N)

```
float tmpx, tmpy;
for(j=0; j<ITER; j++)   ITER
   for(i=0; i<N; i++) {   N
      tmpx = x[i];
      tmpy = y[i];
      z[i] = tmpx + tmpy;
      k[i] = tmpx - tmpy;
      t[i] = tmpx * tmpy;
      u[i] = z[i] + k[i] + t[i];
      }
```

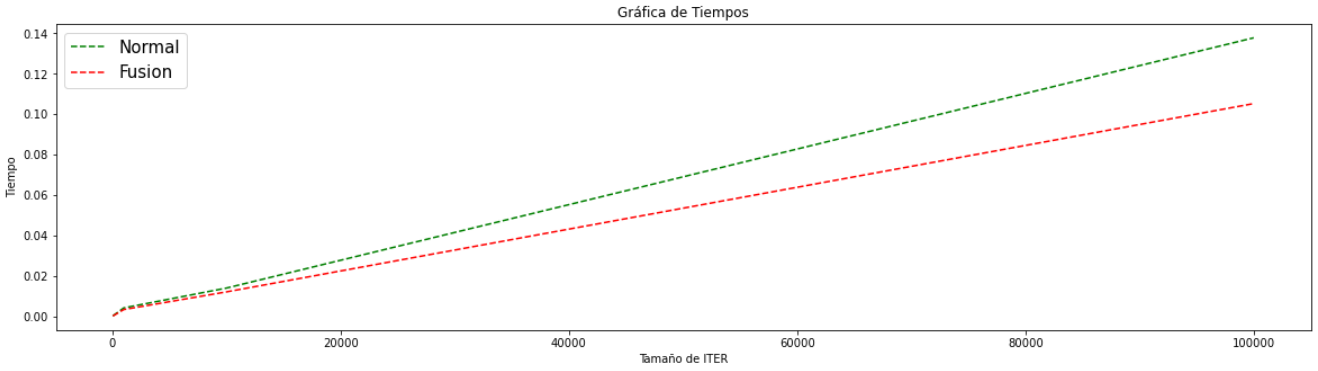
ITER * N

Esto se refleja ligeramente en el tiempo:

N	Mejora Tiempo
50	24,8%
100	24,14%
1000	23,67%
10000	20,42%
100000	34,89%

Por último, realizaré las mismas pruebas que antes pero intercambiando el valor de ITER y N (N=100) para ver cómo afecta el aumento del bucle exterior a las mejoras de tiempo:

ITER	Normal	Fusion
50	0.0002622200	0.0002035400
100	0.0005304400	0.0004091400
1000	0.0042314800	0.0034226000
10000	0.0140575400	0.0121666100
100000	0.1378013900	0.1052742300



ITER	Mejora Tiempo
50	22,39%
100	22,87%
1000	19,10%
10000	13,44%
100000	23,59%

Como vemos el resultado de las pruebas es bastante similar. Existe mejora en todos los puntos, siguiendo un porcentaje poco uniforme de mejora pero que en el último elemento crece con respecto a los anteriores, cumpliendo con el teórico (aunque no tan práctico) aumento de mejora junto con el de tamaño.

Conclusiones

Como hemos podido ver mediante las pruebas, la fusión de lazos consigue mejorar el tiempo de ejecución de nuestro código. A mayores hemos identificado, que el aumento de N (tamaño del bucle/s a fusionar) afecta a la posible mejora que se produzca.
Como comenté al principio, que consigamos una mejora para este código en este dispositivo, no implica que esta optimización lo lograse de igual forma en otro dispositivo o con otro código. Es por ello que los resultados de esta prueba no pueden ser completamente extrapolados al caso general, pero sí muestran la posible utilidad de este tipo de optimizaciones.