Informe Práctica 2

Unroll and jam

Compiladores e Intérpretes

1. Enunciado

Considera la optimización de ***unroll and jam***. Por ejemplo con el siguiente código para una profundidad **d**=4. Analiza la influencia del valor de **d** en los resultados:

```
        //Sin optimización
    int i, j, k, l;
    float a[N][N], b[N][N], c[N][N];

    for (l = 0; l < ITER; l++){
        for (i = 0; i < N; i++){
            for (j = 0; j < N; j++){
                for (k = 0; k < N; k++){
                    c[k][i] += a[k][j] * b[j][i];
                }
            }
        }
    }
        //Con optimización
    int i, j, k, l;
        float a[N][N], b[N][N], c[N][N];
        for (l = 0; l < ITER; l++){
            for (i = 0; i < N; i += 2){
                for (j = 0; j < N; j += 2){
                    for (k = 0; k < N; k++){
                      c[k][i] += (a[k][j] * b[j][i] + a[k][j + 1] * b[j + 1][i]);
                         c[k][i + 1] += (a[k][j] * b[j][i + 1] + a[k][j + 1] * b[j + 1][i +
1]);
                    }
                }
            }
        }
```

2. Técnica de optimización

Unroll and jam es una optimización de bucles que mejora la localidad de datos en la caché y el paralelismo a nivel de instrucción. Esta mejora se realiza aumentando el tamaño del bucle interior reduciendo el de los bucles más exteriores. Un ejemplo base sería el siguiente [1]:

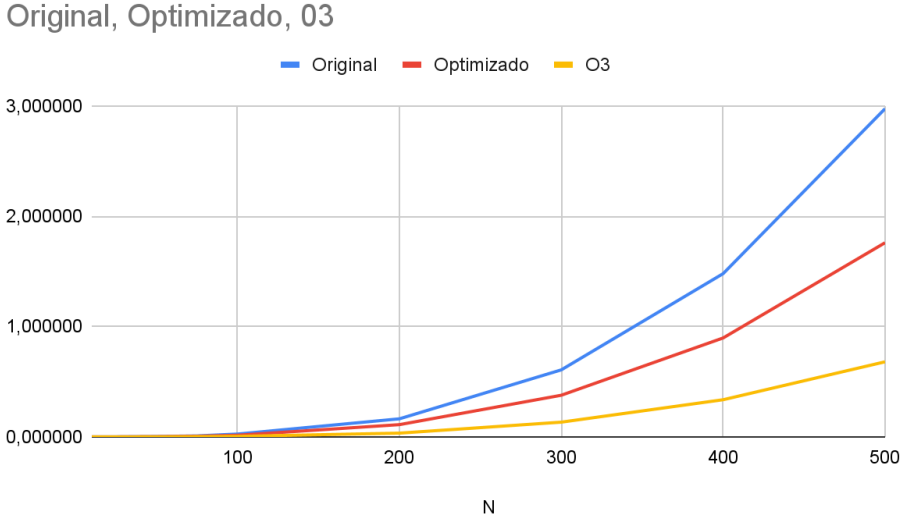
```
//Sin Optimización
DO I = 1, N*2
        DO J = 1, M
       A(J,I) = A(J-1,I) + B(J)
        ENDDO
ENDDO

//Con Optimización
DO I = 1, N*2, 2
        DO J = 1, M
                A(J,I) = A(J-1,I) + B(J)
                A(J,I+1) = A(J-1,I+1) + B(J)
        ENDDO
ENDDO
```

Como se puede apreciar, se realizan 2 iteraciones del bucle interior, saltando de 2 en 2 en el exterior. De esta manera se mejora la localidad y se reduce el número de iteraciones del bucle. Este número se puede modificar y se le refiere como el ***factor de unroll***. Para esta práctica, el *factor de unroll* es de 4, 2 para cada uno de los bucles exteriores.

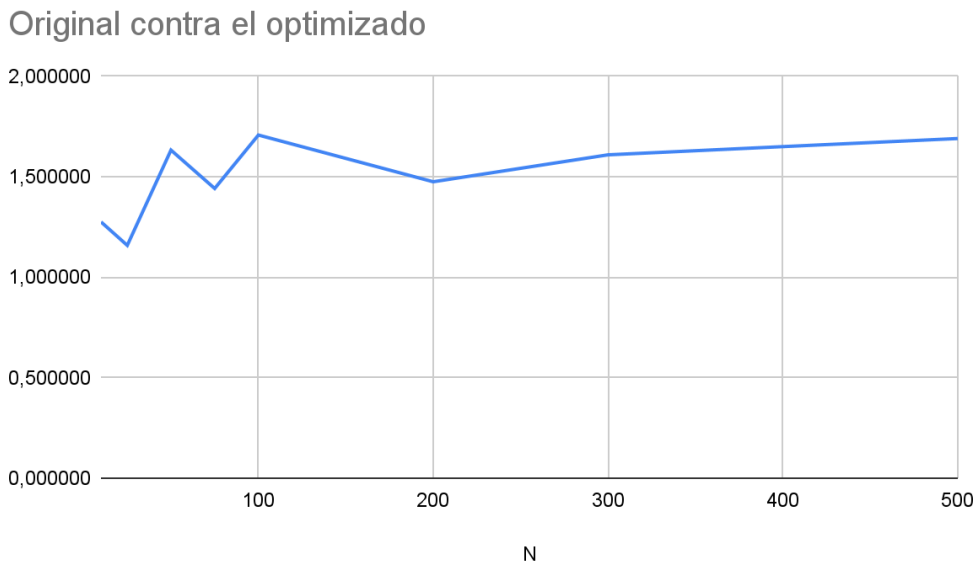
3. Resultados

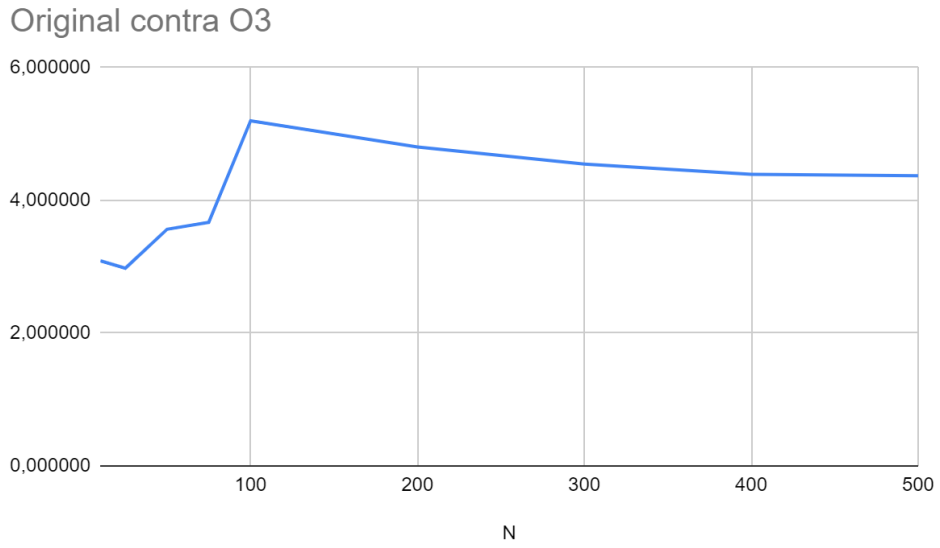
Para conseguir los resultados de los tiempos de ejecución, se han modificado los códigos ofrecidos por el profesor añadiendo las funciones de cálculo de tiempo (***getimeofday***). Se ha utilizado un valor de **10** para el número de iteraciones (***ITER***) dentro del propio código y se ha ejecutado **5** veces ambos códigos desde un script de *Bash* utilizando distintos tamaños de matrices (10, 25, 50, 75, 100, 200, 300, 400, 500). A mayores se ha ejecutado el código base con O3 para ver una mayor optimización. No se han utilizado tamaños mayores a 500 debido a que sería necesario modificar la memoria de estática a dinámica:



Como se puede apreciar en la gráfica anterior, la versión original es más lenta que la optimizada con ***unroll and jam*** para todos los tamaños de N. Además, esta diferencia se amplía cuanto mayor sea el tamaño de las matrices utilizadas. En los datos recogidos en tamaños de N pequeños, los resultados han sido casi instantáneos, por lo que la propia llamada a la función de tiempo debe ser considerada.

Otra gráfica útil para analizar los datos conseguidos es el porcentaje de mejora:



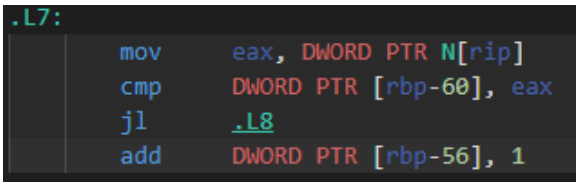


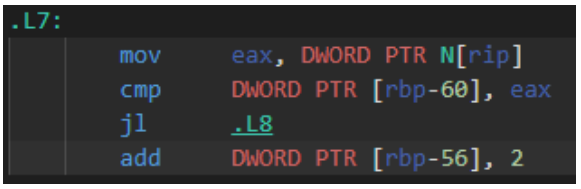
La mejora de la optimización contra la básica se estabiliza a partir del tamaño 300, estableciéndose en un valor de 1'7.

4. Ensamblador

Al comparar el código en ensamblador del código original y el optimizado utilizando unroll and jam se puede apreciar directamente la diferencia. En vez de realizar el cálculo una única vez se realiza dos, lo que aumenta el tamaño del código a casi 100 líneas más

En cambio, el código de los bucles exteriores se puede analizar más fácilmente:





El bucle de la izquierda, el cual hace referencia al código sin optimizar, realiza los saltos de 1 en 1. Por otra parte, el de la derecha los realiza de 2 en 2. Esto se replica en el bucle más externo.

5. Conclusión

Después de analizar los resultados obtenidos, se puede confirmar que esta técnica mejora considerablemente la velocidad de ejecución de los bucles utilizando la localidad y el uso eficiente de registros. Esta mejora se podría ampliar si se utilizase un factor de unroll mayor aunque llegaría un punto en que empezase a declinar ya que los registros donde se almacenan las variables no son suficientes para mantener la localidad y con ello la mejora que conlleva.

Aún con este aumento de la velocidad, no se acerca a lo que logra realizar el compilador con otro tipo de mejoras, llegando a reducir a la mitad el tiempo de ejecución del código optimizado.

6. Bibliografía

[1] Yin Ma and Steve Carr. Register Pressure Guided Unroll-and-Jam. Department of Computer Science, Michigan Technological University, Houghton. (https://www.capsl.udel.edu/conferences/open64/2008/Papers/104.pdf)