Desenrolle de lazos internos con optimización de operaciones de reducción

2 de mayo de 2024

Resumen

En este informe, se abordará el tema del desenrolle de lazos para la optimización de operaciones de reducción, tal que se verá en que consiste la técnica y se comentarán experimentos.

Palabras clave: bucles, lazos, unrolling, desenrollamiento, optimización...

I. Introducción

En el presente informe se describe una serie de experimentos diseñados para estudiar la técnica denominada desenrollamiento de lazos, en el contexto de operaciones de reducción.

Para el estudio, se realizarán varios programas breves escritos en el lenguaje de bajo nivel C, ejecutados en un sistema operativo UNIX. Para la medición de los tiempos de ejecución de los bucles for se utilizará la librería *sys/time.h*.

II. Descripción de la técnica analizada: Loop Unrolling

El desenrollamiento de lazos (loop unrollig en inglés), es una técnica que consiste en intentar minimizar el coste temporal de un bucle o lazo, mediante la reducción del número de iteraciones del mismo.

Básicamente lo que hacemos en esta es la de eliminar o reducir las iteraciones necesarias para realizar una operación, reduciendo así el tiempo de ejecución de la misma.

Un sencillo ejemplo es el siguiente:

```c
// This program does not uses loop unrolling.
#include<stdio.h>

int main(void)
{
    for (int i=0; i<5; i++)
        printf("Hello\n");
            //print hello 5 times

    return 0;
}
```

Listing 1: *Bucle sin desenrollar*

```c
// This program uses loop unrolling.
#include<stdio.h>

int main(void)
{
    // unrolled the for loop in program 1
    printf("Hello\n");
    printf("Hello\n");
    printf("Hello\n");
    printf("Hello\n");
    printf("Hello\n");

    return 0;
}
```

Listing 2: *Bucle aplicando la técnica*

Ambos programas [1] nos muestran un sencillo ejemplo de la aplicación de esta técnica, pudiendo observar en el segundo caso se ha eliminado por completo el bucle, reduciendo así el tiempo de ejecución del programa, debido a que dentro del bucle es necesario hacer una comprobación del índice, lo que ralentiza el código.

Otra manera diferente, para lazos donde eliminar el bucle por completo no es posible, se pueden reducir el número de iteraciones realizando varias operaciones en una sola iteración.

Por ejemplo: imaginemos que tenemos un bucle el cual almacena en una variable el sumatorio de los números del 1 al 10. En un bucle sin desenrollamiento tendríamos que realizar 10 iteraciones, donde en cada una realizamos una operación. Sin embargo, podemos reducir el número de iteraciones a la mitad si en vez de sumar de uno en uno, sumamos de dos en dos, y a su vez podríamos reducirlo a 2 iteraciones si sumamos de 5 en 5. Esto es la otra manera (y que a su vez veremos posteriormente en los experimentos) de aplicar la técnica de desenrollamiento de lazos.

III. Beneficios y desventajas

Como casi cualquier herramienta que podamos utilizar, esta técnica conlleva una serie de ventajas y desventajas. Algunas de las ventajas que podemos encontrar son:

Ventajas:

1. **Reducción de la Sobrecarga de Ciclos de Bucle:** En cualquier sistema, reducir la frecuencia con la que se evalúan las condiciones del bucle y se actualizan las variables de control puede mejorar el rendimiento al disminuir la sobrecarga general del bucle.

2. **Incremento en el Paralelismo:** En sistemas con capacidades de procesamiento paralelo, desenrollar bucles puede permitir que múltiples operaciones se ejecuten al mismo tiempo, mejorando el uso del paralelismo inherente al hardware.

3. **Mejor Uso de las Unidades de Procesamiento:** Al desenrollar un bucle, se pueden realizar más cálculos por cada entrada al bucle, lo cual puede ser eficiente en términos de aprovechamiento del tiempo de CPU y la ejecución en pipelining.

4. **Reducción en el Tiempo de Ciclo de Reloj:** A medida que se desenrollan los bucles y el área utilizada se reduce inicialmente, puede resultar en un decremento en el tiempo de ciclo de reloj, lo cual potencialmente incrementa la frecuencia de operación y por ende la velocidad de procesamiento.

5. **Disminución en el Uso del Área de FPGA (Field Programmable Gate Array) en Ciertos Casos:** Inicialmente, el desenrollamiento puede llevar a una reducción en el área utilizada, lo que es beneficioso cuando el área es una limitante crítica en el diseño del sistema.

Desventajas:

1. **Aumento del Tamaño del Código:** Desenrollar bucles aumenta la cantidad de código que se ejecuta, lo cual puede llevar a un mayor uso de la memoria de instrucciones y potencialmente a un caché de instrucciones más ocupado.

2. **Complicaciones en la Gestión de la Memoria:** El aumento en la cantidad de operaciones ejecutadas simultáneamente puede hacer que la gestión de memoria sea más compleja y puede presionar al ancho de banda de memoria disponible.

3. **Limitaciones por Dependencias de Datos:** Si existen dependencias de datos dentro de las iteraciones de un bucle, el desenrollamiento puede no ser aplicable sin modificar significativamente el algoritmo o puede llevar a la ejecución ineficiente debido a la espera de datos.

4. **Inflexibilidad en la Modificación del Código:** El código desenrollado es menos flexible para modificaciones futuras o ajustes de parámetros del bucle, ya que cualquier cambio requiere más esfuerzo para actualizar todas las instancias desenrolladas del bucle.

Por tanto, viendo las ventajas y las desventajas, el programador deberá valorar si aplicar esta técnica en su código, ya que aunque puede mejorar el rendimiento, también puede conllevar una serie de problemas.

IV. Experimentos

Para los experimentos, se ha realizado un programa en C que mide los dos siguiente bucles:

```
1    for(k=0; k<ITER; k++){
2        a=0.0;
```

```
3           for(i=0; i<N; i++){
4               a = a + x[i] * y[i];
5           }
6       }
```

Listing 3: *Bucle sin desenrollar*

```
1       for(k=0; k<ITER; k++){
2           a = a1 = a2 = a3 = 0.0;
3
4           for(i=0; i<N; i+=4){
5               a = a + x[i] * y[i];
6               a1 = a1 + x[i+1] *
                    y[i+1];
7               a2 = a2 + x[i+2] *
                    y[i+2];
8               a3 = a3 + x[i+3] *
                    y[i+3];
9           }
10
11          a = a + a1 + a2 + a3;
12      }
```

Listing 4: *Bucle aplicando la técnica*

En el primer bucle [3], se realiza la operación de producto escalar de dos vectores de tamaño N, y se repite ITER veces. En el segundo bucle [4], se ha desenrollado el bucle interno, realizando 4 operaciones en cada iteración.

Por tanto sobre estos dos lazos realizarán experimentos modificando el valor de N y de ITER, para ver como afecta el desenrollamiento de lazos en el tiempo de ejecución del programa.

Un detalle a destacar sobre la implementación del código, es la necesidad de limpiar la caché antes de realizar las mediciones, ya que si no se hace, puede afectar a la precisión de los datos medidos. Para ello, se ha implementado la función *clearCache()* la cual, sirviéndose del tamaño de la caché L1d, crea un array de este tamaño para posteriormente rellenarlo entero y posteriormente liberándolo. Debido al tamaño del programa no es necesario limpiar la caché L2, ya que el tamaño de la anterior es de 64KB, y no llega a sobrepasar este tamaño.

A. Tiempos de ejecución

Para el primer experimento vamos a realizar una serie de mediciones variando el tamaño de ITER, con un tamaño de array fijo, y veremos como afecta al tiempo de ejecución del programa.

Para analizarlo correctamente nos serviremos de la siguiente gráfica:

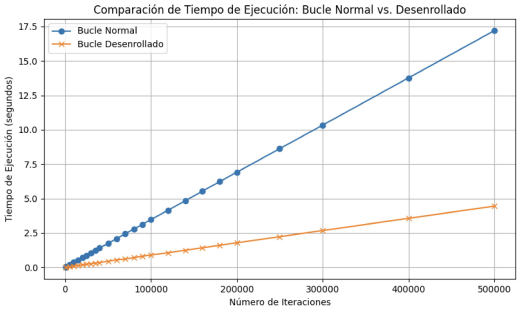


Figura 1: *Gráfica de tiempo de ejecución en función de ITER*

La gráfica nos muestra dos curvas, azul para el bucle normal y naranja para el bucle desenrollado. Como se puede fácilmente observar obtenemo slas siguientes conclusiones:

- **Mejora del rendimiento con el desenrollamiento:** El bucle desenrollado muestra consistentemente tiempos de ejecución inferiores en comparación con el bucle normal a lo largo de todo el rango de iteraciones. Esto indica que el desenrollamiento de bucles puede ser una técnica efectiva para optimizar el rendimiento de los programas en términos de tiempo de ejecución.
- **Escalabilidad:** A medida que aumenta el número de iteraciones, el impacto del desenrollamiento de bucles en la mejora del rendimiento parece ser más pronunciado. La curva del bucle normal muestra un crecimiento más rápido que la del bucle desenrollado, lo que sugiere que la técnica de desenrollamiento se vuelve más ventajosa a gran escala.
- **Crecimiento lineal:** Ambas curvas parecen seguir un patrón de crecimiento lineal, lo cual es esperable ya que el tiempo de ejecución debería aumentar proporcionalmente con el número de iteraciones. No obstante, la pendiente de la curva del bucle desenrollado es claramente menos inclinada, lo que sugiere una menor adición de tiempo por cada iteración adicional en comparación con el bucle normal.

Sin embargo, y a pesar de las mejoras descritas, hay que tener en cuenta que con números bajos de iteraciones, ambos bucles no distan tanto en tiempo de ejecución, por lo que habría que estudiar si en estos casos sería beneficioso aplicar la técnica.

B. Por tamaño de array

En este segundo experimento a realizar, haremos lo contrario al anterior, es decir, mantendremos un tamaño fijo de iteraciones (5000) y variaremos el tamaño del array.

Esto nos permitirá ver como el unrolling afecta a diferentes tamaños de datos, y si verdaderamente es eficaz, frente a un bucle normal sin desenrollar.

Como en el anteriro experimento nos serviremos de la siguiente gráfica, donde el eje X es el tamaño del array (N) y el eje Y el tiempo de ejecución:

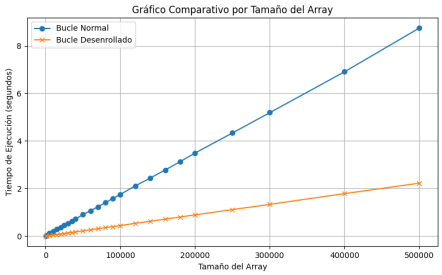


Figura 2: *Gráfica de tiempo de ejecución en función de N*

Al igual que en el experimento anterior, podemos sacar unas conclusiones similares:

- El tiempo de ejecución para el bucle normal aumenta linealmente con el tamaño del array, lo que indica una relación directa entre el tamaño del array y el tiempo de cómputo requerido.
- El bucle desenrollado demuestra una eficiencia notablemente mayor. Aunque el tiempo de ejecución también crece con el tamaño del array, lo hace a una tasa menor que el bucle normal.
- La diferencia de tiempo entre los dos enfoques se amplía con el aumento del tamaño del array. Esto sugiere que para arrays grandes, el desenrollamiento de bucles puede ser especialmente beneficioso.
- La técnica de desenrollamiento de bucles se confirma como una optimización efectiva para la reducción de la sobrecarga de los bucles y el aprovechamiento del paralelismo del hardware, lo cual es crítico en aplicaciones de procesamiento de datos de gran volumen.

Estos resultados son muy similares a los obtenidos en el experimento anterior, lo que nos lleva a pensar que la técnica de desenrollamiento de lazos es efectiva en la reducción de tiempos de ejecución en bucles de gran tamaño.

C. Mejora de Rendimiento

En este apartado vamos a calcular el porcentaje de mejora de rendimiento de una técnica respecto a la otra aplicando la siguiente fórmula:

$$\text{Mejora de Rendimiento (\%)} = \left(\frac{ts - td}{ts}\right) \times 100\,\% \tag{1}$$

Donde ts representa el tiempo sin desenrollar y td el tiempo con desenrollado.

Para representar esto nos serviremos del siguiente historiograma:

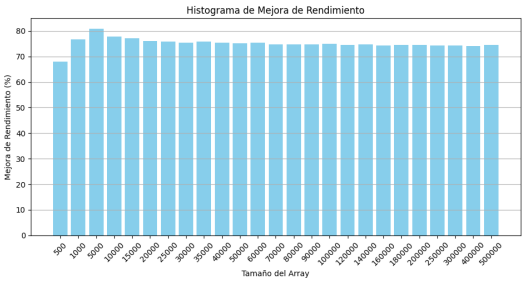


Figura 3: *Histograma de la mejora de rendimiento al aplicar la técnica de desenrollamiento de bucles.*

A partir del histograma, se pueden extraer varias conclusiones:

- La mejora de rendimiento tras aplicar el desenrollamiento de bucles es consistentemente positiva, lo que indica que esta

técnica ofrece beneficios en todos los casos probados.

- La mejora de rendimiento tiene un pico en torno al 80 %, y posteriormente se mantiene constante en valores en torno al 75 %. Esto sugiere que la técnica es especialmente efectiva en la reducción de tiempos de ejecución en cuando tenemos grandes cantidades de datos.
- Hasta el tamaño de array mencionado anteriormente, la mejora de rendimiento aumenta, a partir de esa cantidad, se reduce ligeramente y se mantiene constante.

Es importante considerar que el desenrollamiento de bucles puede introducir complejidad adicional en el código y podría aumentar el tamaño del binario final. Por lo tanto, la decisión de aplicar esta técnica debe balancear la mejora de rendimiento contra estos posibles inconvenientes.

D. Dispersión de Eficiencia de Caché

Como útlimo experimento vamos a estudiar la eficiencia de caché de ambas técnicas. Para ello, vamos a calcular los "misses"de caché para cada técnica, y veremos como eso afecta al tiempo de ejecución.

En términos de arquitectura de computadoras, los "misses"de caché, también conocidos como fallos de caché, se producen cuando los datos requeridos por la CPU no están disponibles en la memoria caché. En consecuencia, se realiza una operación de acceso a la memoria principal para recuperar estos datos. Este proceso implica una latencia significativamente mayor comparada con la recuperación de datos directamente desde la caché, lo que resulta en un aumento en el tiempo de ejecución del programa.

Para el estudio de lo anterior, utilizaremos la herremienta valgrind, con el siguiente comando:

```
valgrind --tool=cachegrind
    ./unrolling <array size>
    <no. of iterations> -O0
```

Listing 5: *Comando para ejecutar valgrind*

Para el estudio de los resultados, separamos las técnicas en dos códigos diferentes, para poder hacer uso del valgrind de manera correcta. Fijamos el numero de iteraciones en 5000 y aumentamos el tamaño del vector, empezando por 500 y acabando en 100000, dando saltos de 5000 en 5000.

Una vez ejecutamos el código, obtenemos los siguientes resultados:

1. En primer lugar vamos a comparar los misses de caché totales para ambas técnicas:

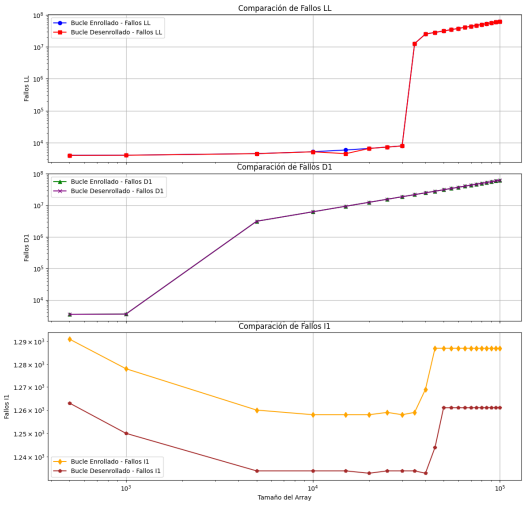


Figura 4: *Comparación de misses de caché totales para bucles normal y desenrollado.*

En la imagen podemos observar tres gráficas, que representan los misses de caché para la LL (Last Level), la I (Instrucciones) y la D (Datos). Es destacable que tanto en la LL como en la D1 (Datos de nivel 1) los fallos son similares para ambas ténicas, solo distando en ciertos puntos que puede deberse a una ejecución concreta del programa, es decir, podemos decir que en general para estas dos cachés el número de misses es similar.

Por otra parte cabe destacar el enorme crecimiento que se procuve en la gráfica de LL que coincide con el tamaño de array de 35 000. Esto puede deberse a que a partir de cierto tamaño entre 30 000 y 35 000, el array supera el tamaño de la caché, lo que provoca un gran aumento de fallos en la caché.

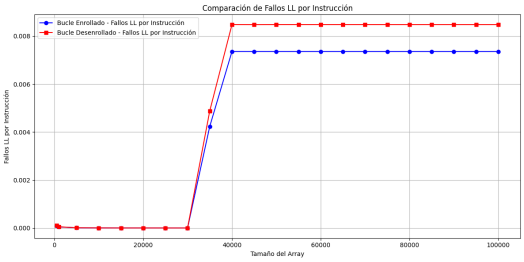
En la caché D1, podemos ver que el crecimiento es cuasi-lineal, solo distando de

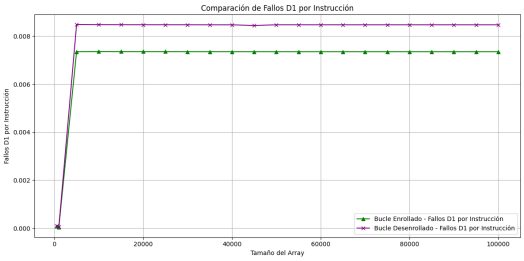
una lineal en los primeros tamaños, que no producen tantos fallos.

Por último, en la caché de instrucciones, podemos ver que hay un decrecimiento en el número de fallos, hasta que llega un determinado tamalo de array, donde se produce un salto creciente y despues se mantiene constante. Este saltop tambien coincide con el tamaño de array de 35 000, lo que nos lleva a pensar que a partir de este tamaño, el programa empieza a tener problemas con la caché de instrucciones, y que después a partir de 45 000 se mantiene constante, lo que puede deberse a que accede a todas las intrucciones únicas necesarias para el programa y las guarda en la caché.

Esta última es la única gráfica donde podemos ver que el bucle desenrollado tiene menos fallos que el bucle normal (lo que se debe a que accede a menos instrucciones), lo que nos lleva a pensar que el desenrollamiento de lazos puede ser beneficioso en este aspecto.

2. Por otra parte vamos a realizar las mismas gráficas pero ahora haciendo el ratio fallo por acceso, para ver como afecta el desenrollamiento de lazos a este ratio:





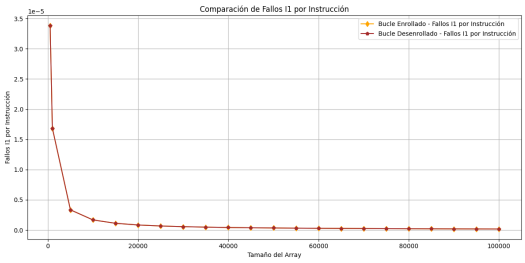


Figura 5: *Comparación de ratio de misses de caché para bucles normal y desenrollado.*

Para mejorar la visualización, ponemos la gráfica sin escalas logarítmicas, ya que al ser un ratio, no es necesario.

Si nos fijamos tanto en las gráfica de la caché LL como la de la D1, podemos ver un resultado que se diferencia de las conclusiones del anterior punto. Esto es, a pesar de que en fallos totales es mejor el bucle desenrollado, al hacer el ratio podemos ver que el bucle normal tiene menos fallos por acceso que el desenrollado. Esto se puede deber a que al desenrollar el bucle se accede a más datos en memoria por iteración, lo que puede aumentar la presión sobre la caché de datos, provocando un aumento general en el número de fallos por acceso.

Por otra parte, podemos ver que a su vez el ratio para la caché de instrucciones es el mismo, y que es completamente decreciente por número de accesos.

Con los resultados anteriores podemos concluir que en términos de eficiencia caché, a pesar de que el bucle desenrollado tiene menos fallos totales, el ratio de fallos por accesos es mayor, por lo que habría que considerarlo a la hora de aplicar la técnica.

V. Análisis del código ensamblador

Para finalizar, estudiamos el código ensamblador generado con la herramienta Godbolt, para ver poder comparar el código generado por ambas técnicas.

Como podemos comprobar al meter el código en la herramienta el código generado por el bucle desenrollado es mucho más largo

que el generado por el bucle normal, lo que puede llevar a pensar que el desenrollamiento de lazos puede ser beneficioso en términos de tiempo de ejecución, pero perjudicial en términos de tamaño del binario final.

Por tanto, al ser el código generado más largo, la eficiencia espacial del código se ve reducida al introducir el desenrollamiento de bucles, lo que puede ser un factor a tener en cuenta a la hora de aplicar la técnica.

VI. Conclusiones

Podemos concluir que una vez realizados los experimentos, el desenrollamiento de lazos es una técnica efectiva para la optimización de bucles en términos de tiempo de ejecución, pero tenemos que tener en cuenta que aumenta el tamaño del binario final y puede reducir la eficiencia espacial.

A pesar de todo esto, el ünrollingresulta en la mayoría de casos una herramienta efectiva, y puede ayudar a reducir el tiempo de ejecución de los programas en bucles con un gran número de datos.

Referencias

[1] GeeksforGeeks. *Loop Unrolling*. `https://www.geeksforgeeks.org/loop-unrolling/`, [online]

[2] O. S. Dragomir, T. Stefanov, and K. Bertels. Optimal Loop Unrolling and Shifting for Reconfigurable Architectures. *ACM Transactions on Reconfigurable Technology and Systems*, Vol. 2, No. 4, Article 25, September 2009, 24 pages. DOI = 10.1145/1575779.1575785. `http://doi.acm.org/10.1145/1575779.1575785`, [online]

[3] Betul Buyukkurt, Zhi Guo, and Walid A. Najjar. *Impact of Loop Unrolling on Area, Throughput and Clock Frequency in ROCCC: C to VHDL Compiler for FPGAs*. Department of Computer Science and Engineering, University of California - Riverside, Riverside, CA 92507, USA.