Desenrolle de lazos internos con optimización de operaciones de reducción

6 de mayo de 2024

Resumen

En este informe se abordará el tema del desenrrolle de lazos y como afecta su produndidad en términos de tiempo de ejecución del programa.

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, y su profundidad. Con su profundidad nos referimos a la cantidad de veces que se ejecuta el cuerpo del bucle en cada iteración.

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:

```
1  // This program does not uses loop unrolling.
2  #include<stdio.h>
3
4  int main(void)
5  {
6      for (int i=0; i<5; i++)
7          printf("Hello\n");
               //print hello 5 times
8
9      return 0;
10 }
```

Listing 1: *Bucle sin desenrollar*

```
1  // This program uses loop unrolling.
2  #include<stdio.h>
3
4  int main(void)
5  {
6      // unrolled the for loop in program 1
7      printf("Hello\n");
8      printf("Hello\n");
9      printf("Hello\n");
10     printf("Hello\n");
11     printf("Hello\n");
12
13     return 0;
14 }
```

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.

A. Experimentos

Para la realización de los experimentos hay que tener en cuenta un factor fundamental, y es referido al tamaño del array. Este debe ser divisible entre el número de desenrolles

que queramos hacer, de tal manera que que no se produzca ningún error de acceder a posiciones que no existen.

Por tanto, y teniendo en cuenta lo anterior lo que hacemos es utilizar los múltiplos de 2 desde el 16 hasta el 8192, para que no haya problemas de acceso a memoria. A su vez ponemos un número de iteraciones suficientemente alto para que se pueda apreciar la diferencia entre los bucles desenrollados y los que no lo están. En este caso, hemos puesto 2 000 000 iteraciones.

También es destacable que utilizamos factores de desenrollamiento que sean potencias de 2 para que se alineen mejor con los tamaños de los resigtros y la arquitecutra del hardware.

A.1. Desenrolle de tamaño 4

Para empezar con los experimentos, haremos un desenrolle de factor 4, es decir, que en cada iteración del bucle se realizarán 4 operaciones. Para esto, mediremos el tiempo de ejecución de un bucle normal y de uno desenrollado y los compararemos.

De los resultados temporales obtenemos la siguiente gráfica:

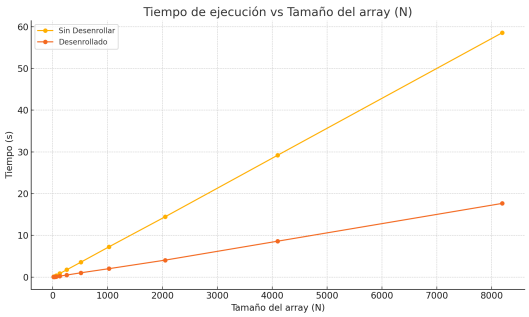

Figura 1: *Tiempos de ejecución para un bucle desenrollado de factor 4*

Como podemos comprobar en la gráfica, el desenrolle de lazos mejora el tiempo de ejecución del programa, y a medida que incremental el tamaño del array, la diferencia entre ambos bucles se hace más notable, como comprobamos en la siguiente gráfica donde medimos el porcentaje de memoria:

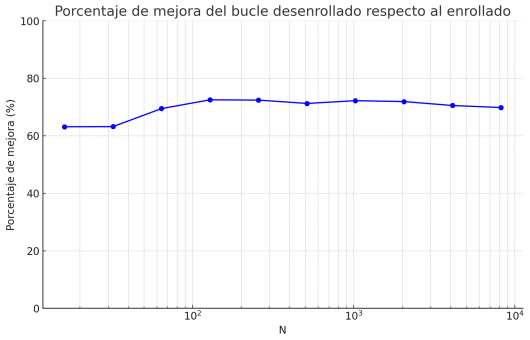

Figura 2: *Porcentaje de mejora del bucle desenrollado respecto al enrollado*

Por tanto podemos concluir que en términos de eficiencia temporal, el desenrolle de lazos mejora la eficiencia temporal del programa, y que a mayor el tamaño del array, mayor será la diferencia entre ambos bucles.

Es importante destacar que esta conclusion se repetiría para cualquier factor de desenrolle, ya que la mejora en el tiempo de ejecución es notable en todos los casos.

A.2. Comparación de los desenrolles

Una vez comparado el bucle desenrollado de tamaño 4 con el enrollado, vamos a comparar los tiempos de ejecución de los bucles desenrollados de tamaño 2, 4, 8 y 16. Aquí veremos hasta que punto es rentable hacer grandes desenrolles, y si hay un punto en el que el rendimiento empeora, o que simplemente el porcentaje de mejora respecto al anterior no es significativo:

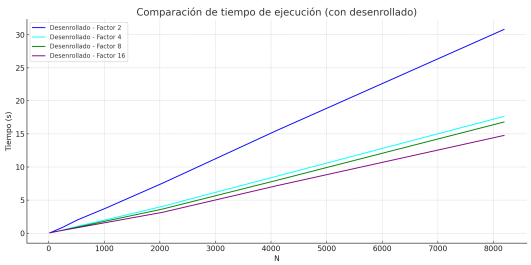

Figura 3: *Tiempos de ejecución para los diferentes factores de desenrolle*

Como vemos en la gráfica, a medida que incrementamos el factor de desenrolle, el tiempo de ejecución del programa disminuye, pero a partir del factor 8, la mejora en el tiempo

de ejecución es menos significativa, y con 16 podemos apreciar una diferencia ligeramente mayor a la que tiene 8 respecto a 4.

Siendo esta la vista del tiempo total, miremos la gráfica de porcentaje de mejoras:

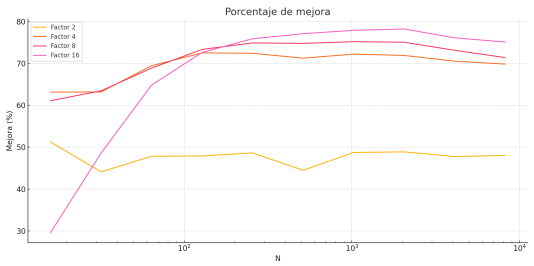

Figura 4: *Porcentaje de mejora respecto al bucle sin desenrollar*

Al poner la escala logarítmica en el eje de las x, podemos apreciar mejor las diferencias entre los diferentes factores. Podemos apreciar que para valores pequeños el factor 4 y 8 son similares, e incluso podemos apreciar que para valores realmente pequeños, el factor de 16 es peor incluso que 2.

A medida que vamos aumentando el tamaño del array, el factor 16 va subiendo su porcentaje de mejora, hasta llegar a ser el mejor factor de desenrolle para arrays grandes. Tambien el factor 8 se desmarca ligeramente del factor 4, siendo el segundo mejor factor de desenrolle.

Una constante, es que menos para los valores muy pequeños, donde el factor 16 es peor que el 2, el propio 2 se mantiene muy distanciado del resto, estando hasta 30 puntos porcentuales por debajo del factor 4.

Sin embargo, y a pesar de lo anterior, hay que destacar que la razón por la que el factor 16 es peor en pequeñas cantidades es porque, al ser el tamaño del array 16, el bucle desenrollado solo se ejecuta una vez, por lo que empeora la mejora del tiempo de ejecución. Es decir, si el factor de desenrollado es igual o mayor al tamaño del array, entonces el desenrollado del bucle no tiene el mismo impacto positivo que cuando el factor de desenrollado es menor.

Por eso, a menudo se recomienda ajustar el factor de desenrollado para que sea significativamente menor que el tamaño del array, permitiendo múltiples iteraciones del bucle desenrollado y optimizando el rendimiento.

IV. Código Ensamblador

En esta sección se analizarán los códigos ensamblador generados para los distintos factores de desenrollamiento, comparando su tamaño y las implicaciones de estos tamaños en el rendimiento y en otros aspectos del programa.

A. Código con Factor de Desenrollamiento 2

El código ensamblador para el factor de desenrollamiento 2 tiene un tamaño total de 1279 líneas. Es el código más compacto de los cuatro debido a que realiza el menor número de operaciones en cada iteración del bucle desenrollado.

B. Código con Factor de Desenrollamiento 4

El código ensamblador para el factor de desenrollamiento 4 tiene un tamaño total de 1310 líneas. Esto es un incremento respecto al tamaño del código con desenrollamiento de factor 2, lo cual es esperable, ya que el bucle desenrollado procesa más elementos en cada iteración.

C. Código con Factor de Desenrollamiento 8

El código ensamblador para el factor de desenrollamiento 8 tiene un tamaño total de 1344 líneas. Al igual que en el caso anterior, el aumento en el factor de desenrollamiento implica más operaciones en cada iteración y, por lo tanto, un código más grande.

D. Código con Factor de Desenrollamiento 16

El código ensamblador para el factor de desenrollamiento 16 tiene un tamaño total de 1388 líneas, siendo el más extenso de los cuatro. Esto se debe a que procesa 16 elementos en cada iteración, lo que requiere un código más largo.

E. Análisis del Tamaño del Código

En general, el tamaño del código aumenta con el factor de desenrollamiento, ya que un mayor factor implica más operaciones en cada iteración del bucle. Este aumento en el tamaño del código puede tener varias implicaciones:

1. Uso de Memoria: Un código más grande ocupa más memoria, lo que puede ser problemático en sistemas con recursos limitados o cuando se busca optimizar el uso de la memoria.

2. Caché de Instrucciones: Un código más grande puede impactar negativamente el rendimiento al ocupar más espacio en la cache de instrucciones, especialmente si el código no cabe completamente en la cache.

Por tanto es importante estudiar el tamaño del código generado, y ponerlo en relación con el porcentaje de memoria para escoger correctamente el factor de desenrollamiento.

V. Conclusiones

En este informe, se exploró el impacto del desenrollamiento de lazos en operaciones de reducción, enfocándose en cómo la profundidad del desenrollamiento afecta el tiempo de ejecución del programa. A través de una serie de experimentos con factores de desenrollamiento de 2, 4, 8 y 16, se observaron mejoras en el tiempo de ejecución en comparación con un bucle sin desenrollar.

Los resultados mostraron que el desenrollamiento de lazos generalmente mejora el tiempo de ejecución, y que los factores más altos de desenrollamiento proporcionan mejores mejoras, aunque con rendimientos que se acaban por estabilizar (en torno al 75-80 %).

Para tamaños de array pequeños, como 16 y 32, el factor 16 resultó ser menos eficiente, ya que el desenrollamiento resultó en una única (o dos) iteración, eliminando la mejora esperada. Esto indica que es importante ajustar el factor de desenrollamiento a un valor significativamente menor que el tamaño del array para obtener mejoras óptimas.

Por lo tanto, el desenrollamiento de lazos es una técnica valiosa para optimizar el rendimiento, pero debe usarse con cuidado y ajustarse según el tamaño del array y la arquitectura del sistema. La elección del factor de desenrollamiento adecuado puede depender del tamaño del array y las limitaciones del hardware, pero generalmente, factores como 4 u 8 son buenos puntos de partida.

Un punto importante, es que se debe tener en cuenta el aumento de complejidad que acarrea el desenrollamiento de lazos, lo que puede dificultar el mantenimiento del código, así como también el aumento del tamaño del código como vimos en la sección anterior.

En resumen, el desenrollamiento de lazos es una técnica poderosa para mejorar el rendimiento de los programas, pero debe usarse con precaución y ajustarse cuidadosamente para obtener los mejores resultados.

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.