Compiladores e Intérpretes

Práctica 2: Generación y optimización de código

Reemplazo de multiplicaciones y divisiones enteras por operaciones de desplazamiento

AUTOR ANÓNIMO

Universidade de Santiago de Compostela

Índice

Versión: 30 de abril de 2025, 18:56

1. Introducción

El rendimiento de los programas informáticos depende en gran medida de cómo se implementen las operaciones aritméticas, especialmente en bucles intensivos en cálculos. En esta práctica se analiza una técnica clásica de optimización a bajo nivel: el reemplazo de multiplicaciones y divisiones enteras por potencias de dos mediante operaciones de desplazamiento binario. Esta transformación permite, en muchos casos, reducir el coste computacional asociado a estas operaciones, aprovechando que los desplazamientos ($\ll$, $\gg$) se ejecutan generalmente más rápido que las multiplicaciones o divisiones en la mayoría de arquitecturas modernas.

El objetivo principal del estudio es implementar dicha optimización sobre el código proporcionado, evaluar su impacto en el rendimiento y analizar su escalabilidad con respecto al tamaño del problema y al número de repeticiones (`ITER`). Para obtener medidas fiables, se realizan múltiples ejecuciones de cada versión del código (con y sin optimización), compiladas con el nivel de optimización `-O0` para evitar interferencias de otras transformaciones automáticas del compilador. Asimismo, se analiza el código ensamblador generado con el fin de confirmar que la sustitución ha tenido efecto y para entender mejor las diferencias observadas en los tiempos de ejecución.

La práctica se completa con una interpretación de los resultados experimentales y un estudio de cómo influyen factores como la memoria caché y el número de repeticiones sobre la estabilidad y precisión de las medidas de tiempo, aportando una visión más completa sobre los beneficios reales de aplicar esta optimización.

2. Descripción de la técnica

El desplazamiento de bits es una operación que consiste en mover los bits de un número binario hacia la izquierda o hacia la derecha una cantidad determinada de posiciones. Esta técnica es fundamental en programación de bajo nivel y se utiliza con frecuencia en ámbitos como sistemas embebidos, procesamiento gráfico y optimización de rendimiento, ya que permite realizar operaciones rápidas y con control detallado a nivel de bits, según se explica en 1 .

Una operación de **desplazamiento a la izquierda** ($\ll$) mueve todos los bits de un número binario hacia la izquierda, rellenando con ceros los bits vacíos del lado derecho. Esta operación equivale a multiplicar el número original por 2^n, donde n es el número de posiciones desplazadas. Por ejemplo, desplazar un número una posición a la izquierda es equivalente a multiplicarlo por 2.

De forma análoga, una operación de **desplazamiento a la derecha** ($\gg$) traslada los bits hacia la derecha. Si el desplazamiento es lógico, los bits vacíos del lado izquierdo se rellenan con ceros; si es aritmético, se conserva el bit de signo en números con representación en

complemento a dos. Esta operación es equivalente, en muchos casos, a una división entera por potencias de dos.

El uso de desplazamientos puede reemplazar multiplicaciones o divisiones enteras por potencias de dos, lo que permite reducir el coste computacional de estas operaciones en ciertas arquitecturas. Esto puede dar lugar a algoritmos más rápidos y eficientes, especialmente en aplicaciones críticas para el rendimiento, como el procesamiento gráfico, la criptografía o la indexación de grandes volúmenes de datos.

El código con multiplicaciones y divisiones inicialmente propuesto es:

```
int i, j, m3 = 8, m5 = 32, a = 0, b = 0;

for (j = 0; j < ITER; j++) {
        for (i = 0; i < N; i++) {
                a = i * m3;
                b += a / m5;
        }
}
```

Es claro que `a` representa el valor del índice (sobre el tamaño del problema) multiplicado por $2^3 = 8$, para luego sumar de forma acumulada `b` con `a` dividido entre $2^5 = 32$.

La optimización que se implementará será la siguiente. Un simple análisis llega para ver que:

$$\mathtt{b} = \mathtt{b} + \frac{\mathtt{a}}{2^5} = \mathtt{b} + \frac{i \cdot 2^3}{2^5} = \mathtt{b} + \frac{i}{2^2}. \tag{1}$$

En la versión optimizada del código tenemos en cuenta (1) e implementamos la operación con un desplazamiento de 2 bits a la derecha, como se ha explicado.

```
for (j = 0; j < ITER_local; j++) {
        for (i = 0; i < N_local; i++) {
                b += i >> 2;
        }
        a = (N_local - 1) << 3;
}
```

De manera similar, para intentar ser fiel con el cálculo de `a` en cada uno de los índices sobre `ITER`, terminamos la iteración multiplicando como sigue. El término $\mathtt{a}_i$ se refiere al valor de `a` antes de comenzar la iteración $i+1$ sobre `ITER`.

$$\forall i,\ \mathtt{a}_i = i \cdot 2^3 \implies \mathtt{a}_{N-1} = (N-1) \cdot 2^3. \tag{2}$$

Además de su uso en operaciones aritméticas, el desplazamiento de bits permite manipular directamente bits individuales, lo cual resulta útil para configurar, borrar o alternar banderas

dentro de registros binarios. Comprender estas operaciones es esencial para aquellos que deseen profundizar en el funcionamiento interno del hardware y en la eficiencia del software a bajo nivel.

3. Beneficios y desventajas esperados

4. Códigos en ensamblador

5. Interpretación de los resultados

6. Conclusiones

7. Referencias

1. https://wraycastle.com/es/blogs/knowledge-base/bit-shift-calculator#:~:text=El%20desplazamiento%20