Obtención de rendimiento y análisis de optimizaciones

Autores: Abel Chils Trabanco(718997) y Diego Santolaya Martínez(727209)

Las pruebas se han realizado en dos máquinas diferentes. Los tiempos de la tabla C-C y C-ARM se ejecutaron en una máquina y el resto de tiempos en otra.

Los tiempos que se han calculado han sido al ejecutar las funciones sobre un bucle un millón de iteraciones.

Pruebas de ejecución y análisis optimizaciones C-C

El número de instrucciones de la combinación c-c con los diferentes niveles de optimización así como el código generado se comprobó tanto llamando a la función desde dentro de un bucle (función bucle)[1], siendo en este caso posible también medir el tiempo de ejecución y llamando a la función solamente (función sola)[2], en este caso fue imposible medir el tiempo de ejecución.

Resultados:

Pruebas C-C (función bucle)

Nivel optimización	Número instrucciones ficha_valida	Número instrucciones patron_volteo	Tamaño código (bytes)	Número instrucciones ejecutadas	Tiempo ejecución (segundos)
O0	45	69	456	68	9,58
O1	45	69	456	68	9,58
O2	45	69	456	68	9,58
O3	45	69	456	68	9,58
Os	45	69	456	68	9,58

Pruebas C-C (función sola)

Nivel optimización	Número instrucciones ficha_valida	Número instrucciones patron_volteo	Tamaño código (bytes)	Número instrucciones ejecutadas	Tiempo ejecución (segundos)
O0	45	69	456	68	-
O1	14	49	252	44	-
O2	7	43	200	20	-
O3	0	0	0	2	-
Os	6	49	220	40	-

Como se puede comprobar los resultados son diferentes según el contexto desde el que se llama a la función. En el caso de que se llame sola a la función se puede comprobar como con las diferentes optimizaciones esta cambia su y número de instrucciones ejecutadas, en cambio al llamarse desde un bucle esta con optimizaciones no varía. Esto es debido a que al llamarse desde un bucle que realiza tantas iteraciones el compilador es incapaz de prever el comportamiento de la función para hacer optimizaciones sobre ella y por eso hace una traducción literal.

En cambio, al llamarse sola a la función esto cambia.

Con el nivel de optimización o0 produce un código bastante ineficiente el cual solo mantiene en registro los valores con los que está operando, esto hace el debuggear más sencillo a costa de tener que realizar muchas llamadas a memoria.

En el nivel de optimización o1 el compilador utiliza todos los registros, de esta forma evita accesos innecesarios a memoria, además empieza a realizar optimizaciones sobre el código. Por ejemplo, en la funcion_fichavalida aprovecha el hecho de que, al representar un número negativo, al menos el último bit ha de estar a 1, por lo que, si lo interpretas como un número sin signo, su valor será muy elevado. Si este número es de tipo char y quieres comprobar si se encuentra en el rango 0,7 sólo has de hacer una comparación si es mayor a 7 interpretando el número como si no tuviese signo. Si sale falso está en el intervalo.

Esta mejora la hemos aplicado a nuestro código.

En el nivel de optimización o2 aparte de las mejoras anteriores se realiza un inline de la función ficha valida, lo cual evita hacer la llamada a la función.

En el nivel de optimización o3, directamente precalcula el resultado, asignando a las variables que modifica la función su resultado después de aplicarla, por ello el tamaño del código es de 0 bytes.

En el nivel de optimización oS se presentan las mismas características que en el o1 aunque el código resultante es diferente.

Pruebas de ejecución y análisis optimizaciones C-ARM

El número de instrucciones de la combinación c-ARM con los diferentes niveles de optimización así como el código generado y el tiempo empleado se comprobó llamando a la función desde dentro de un bucle (función bucle)[1].

Resultados:

Pruebas C-ARM (función bucle)

Nivel optimización	Número instrucciones ficha_valida	Número instrucciones patron_volteo	Tamaño código (bytes)	Número instrucciones ejecutadas	Tiempo ejecución (segundos)
O0	15	69	336	45	7,76
O1	15	50	260	46	7,66
O2	15	50	260	46	7,66
O3	15	50	260	46	7,66
Os	15	50	260	46	7,66

En este caso menos con el nivel de optimización O0, el código generado para la función patrón_volteo fue el mismo. La diferencia entre ambos es el uso intensivo de la memoria en el nivel o0.

Esto es debido a que se le restringen las posibilidades al compilador teniendo implementada la función ficha_valida, debido a que ha de cumplir el estándar de llamadas lo cual produce un código modular pero no óptimo.

Pruebas de ejecución combinaciones entre C, ARM y THUMB

Tanto los tiempos como el número de instrucciones se comprobaron sobre la función desde dentro de un bucle (función bucle) [1].

Los tiempos se obtuvieron desde una máquina diferente a desde la que se obtuvieron las combinaciones C-ARM y C-C, por ello estos tiempos no son directamente comparables.

Resultados:

Pruebas C-ARM (función bucle)

Nivel optimización	Número instrucciones ficha_valida	Número instrucciones patron_volteo	Tamaño código (bytes)	Número instrucciones ejecutadas	Tiempo ejecución (segundos)
O0	15	69	336	45	5,5

Pruebas C-THUMB (función bucle)

Nivel optimización	Número instrucciones ficha_valida	Número instrucciones patron_volteo	Tamaño código (bytes)	Número instrucciones ejecutadas	Tiempo ejecución (segundos)
O0	23	69	322	53	6,1

Pruebas C-C (función bucle)

Nivel optimización	Número instrucciones ficha_valida	Número instrucciones patron_volteo	Tamaño código (bytes)	Número instrucciones ejecutadas	Tiempo ejecución (segundos)
O0	45	69	456	68	7,1

Pruebas ARM-ARM (función bucle)

Nivel optimización	Número instrucciones ficha_valida	Número instrucciones patron_volteo	Tamaño código (bytes)	Número instrucciones ejecutadas	Tiempo ejecución (segundos)
O0	15	43	232	33	4,7

Pruebas ARM-THUMB (función bucle)

Nivel optimización	Número instrucciones ficha_valida	Número instrucciones patron_volteo	Tamaño código (bytes)	Número instrucciones ejecutadas	Tiempo ejecución (segundos)
O0	23	43	218	39	5,3

Pruebas ARM-C (función bucle)

Nivel optimización	Número instrucciones ficha_valida	Número instrucciones patron_volteo	Tamaño código (bytes)	Número instrucciones ejecutadas	Tiempo ejecución (segundos)
O0	45	43	352	55	6,3

Como se puede comprobar la combinación más eficiente tanto en tiempo como número de instrucciones ejecutada es la formada por ARM-ARM y la que menos espacio ocupa es la formada por ARM-THUMB

Referencias sobre funciones utilizadas para la comprobación de los tiempos:

1 - Función bucle

```
while ( i < 1000000 ){
  flip = 0;
  result = patron_volteo(tablero_tiempos, &flip, 2, 3, -1, 0, 2);
  i++;
}
```

2 - Función sola

```
  flip = 0;
  result = patron_volteo(tablero_tiempos, &flip, 2, 3, -1, 0, 2);
```