TEC | Tecnológico de Costa Rica

Área Académica Ingeniera en Computadores

Arquitectura de Computadores I

Taller 2

Profesor:

Luis Chavarría Zamora

Estudiante:

Jose Ignacio Granados Marín

Grupo 1

IS-2022

1. ¿Cómo se aplican las optimizaciones y para qué son?

Según GCC GNU ORG, el objetivo del compilador es reducir el costo de la compilación y lograr que el proceso de debbuging produzca los resultados esperados, con o sin una optimización. Al activar un indicador de optimización, el compilador intentará mejorar el rendimiento, el tamaño y la capacidad de debbuging del programa con base en el conocimiento que se tenga del mismo.

Ahora bien, según Oracle, para gcc se pueden aplicar las siguientes optimizaciones:

- -O0:

Esta corresponde a la optimización por defecto y es equivalente a la optimización -O. Dicho comando, reduce el tiempo de compilación y permite que la depuración siempre produzca el resultado esperado. Sin embargo, si se aplica esta optimización, aún se encuentras habilitadas las opciones de -falign-loops, -finline-functions-called-once y -fmove-loop-invariants.

- -O1:

Si se emplea esta optimización, el compilador intentará reducir el tamaño del código binario de salida y la velocidad de ejecución. No obstante, no realizará ninguna optimización que aumente significativamente el tiempo de compilación.

- -O2:

Al utilizar esta optimización, el compilador realizará optimizaciones que no sacrifican espacio por velocidad. Asimismo, esta optimización mejora el rendimiento del binario de salida y aumenta el tiempo de compilación.

- -O3:

Esta optimización, en particular, hará que el compilador active las opciones -fgcse-after-reload, -finline-functions, -fipa-cp-clone, -fpredictive-commoning, -ftree-vectorize y -funswitch-loops, las cuales requieren una compensación de espacio por velocidad. Además, se aplicarán las optimizaciones del –O2.

- -Os:

Finalmente, esta optimización permitirá que el compilador reduzca el tamaño del código binario de salida en lugar de la velocidad de ejecución.

Para el caso de la optimización del archivo test-printf.c, se aplicaron las optimizaciones -O1, -O2 y -O3 a través del comando sim-safe, las cuales arrojaron las siguientes estadísticas:

```
sim: ** simulation statistics **
sim_num_insn               5918407 # total number of instructions executed
sim_num_refs               1217593 # total number of loads and stores executed
sim_elapsed_time                 1 # total simulation time in seconds
sim_inst_rate          5918407.0000 # simulation speed (in insts/sec)
ld_text_base             0x00400000 # program text (code) segment base
ld_text_size                  74768 # program text (code) size in bytes
ld_data_base             0x10000000 # program initialized data segment base
ld_data_size                  13540 # program init'ed `.data' and uninit'ed `.bss' size in bytes
ld_stack_base            0x7fffc000 # program stack segment base (highest address in stack)
ld_stack_size                 16384 # program initial stack size
ld_prog_entry            0x00400140 # program entry point (initial PC)
ld_environ_base          0x7fff8000 # program environment base address address
ld_target_big_endian              0 # target executable endian-ness, non-zero if big endian
mem.page_count                   30 # total number of pages allocated
mem.page_mem                   120k # total size of memory pages allocated
mem.ptab_misses                  35 # total first level page table misses
mem.ptab_accesses          26614243 # total page table accesses
mem.ptab_miss_rate           0.0000 # first level page table miss rate
```

Figura 1. Estadísticas de la optimización -O1.

```
sim: ** simulation statistics **
sim_num_insn               3803043 # total number of instructions executed
sim_num_refs                861474 # total number of loads and stores executed
sim_elapsed_time                 1 # total simulation time in seconds
sim_inst_rate          3803043.0000 # simulation speed (in insts/sec)
ld_text_base             0x00400000 # program text (code) segment base
ld_text_size                  74816 # program text (code) size in bytes
ld_data_base             0x10000000 # program initialized data segment base
ld_data_size                  13556 # program init'ed `.data' and uninit'ed `.bss' size in bytes
ld_stack_base            0x7fffc000 # program stack segment base (highest address in stack)
ld_stack_size                 16384 # program initial stack size
ld_prog_entry            0x00400140 # program entry point (initial PC)
ld_environ_base          0x7fff8000 # program environment base address address
ld_target_big_endian              0 # target executable endian-ness, non-zero if big endian
mem.page_count                   30 # total number of pages allocated
mem.page_mem                   120k # total size of memory pages allocated
mem.ptab_misses                  33 # total first level page table misses
mem.ptab_accesses          17435589 # total page table accesses
mem.ptab_miss_rate           0.0000 # first level page table miss rate
```

Figura 2. Estadísticas de la optimización -O2.

```
sim: ** simulation statistics **
sim_num_insn               6994228397 # total number of instructions executed
sim_num_refs               1614008845 # total number of loads and stores executed
sim_elapsed_time                 161 # total simulation time in seconds
sim_inst_rate          43442412.4037 # simulation speed (in insts/sec)
ld_text_base             0x00400000 # program text (code) segment base
ld_text_size                  75504 # program text (code) size in bytes
ld_data_base             0x10000000 # program initialized data segment base
ld_data_size                  13540 # program init'ed `.data' and uninit'ed `.bss' size in bytes
ld_stack_base            0x7fffc000 # program stack segment base (highest address in stack)
ld_stack_size                 16384 # program initial stack size
ld_prog_entry            0x00400140 # program entry point (initial PC)
ld_environ_base          0x7fff8000 # program environment base address address
ld_target_big_endian              0 # target executable endian-ness, non-zero if big endian
mem.page_count               525349 # total number of pages allocated
mem.page_mem                2101396k # total size of memory pages allocated
mem.ptab_misses              623657 # total first level page table misses
mem.ptab_accesses         31205409052 # total page table accesses
mem.ptab_miss_rate           0.0000 # first level page table miss rate
```

Figura 3. Estadísticas de la optimización -O3.

De las figuras anteriores, se puede observar que cada optimización arrojó diferentes valores de tiempo de ejecución, tamaño del código, uso de memoria y tiempo de compilación. Por lo que se puede resaltar las siguientes diferencias:

La optimización -O1 posee:

- La mayor velocidad de simulación.
- El menor tamaño de código.
- Igual uso de memoria que la optimización -O2.
- Igual tiempo de compilación que la optimización -O2.

La optimización -O2 posee:

- La menor velocidad de simulación.
- Un mayor tamaño de código que la optimización -O1, pero menor que la optimización -O3.
- Igual uso de memoria que la optimización -O1.
- Igual tiempo de compilación que la optimización -O1.

La optimización -O3 posee:

- Una menor velocidad de simulación que la optimización -O1, pero menor que la optimización –O2.
- El mayor tamaño de código.
- El mayor uso de memoria.
- El mayor tiempo de compilación.

2. ¿Qué información le proporciona el simulador sim-fast sobre la aplicación?

Luego de compilar y ejecutar el archivo test-printf.c con la optimización -O2, el simulador de sim-fast arrojó las siguientes estadísticas:

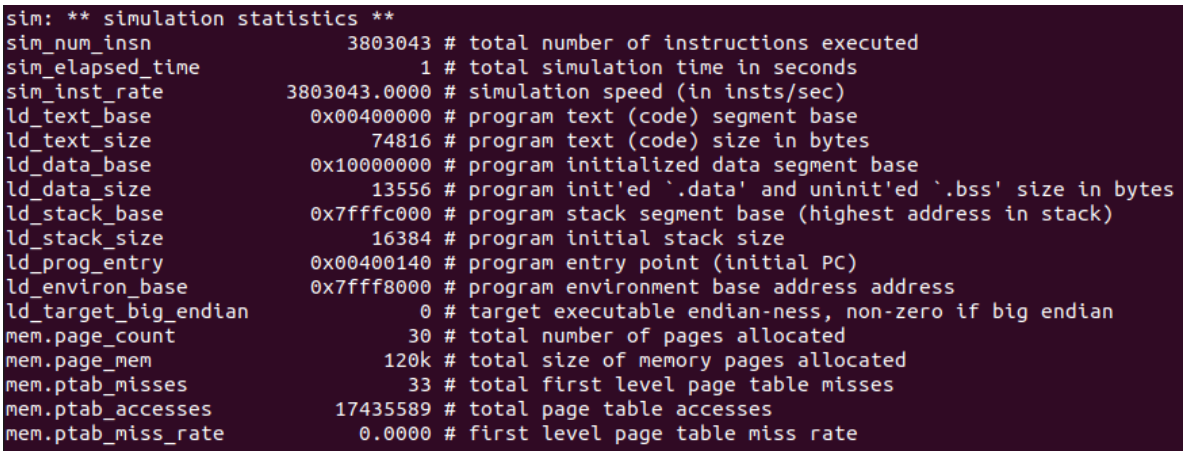

Figura 4. Estadísticas del simulador de sim-fast del archivo test-printf.c.

Es importante destacar que se debe ejecutar el comando “sslittle-na-sstrix-gcc -o test test-printf.c” en vez del comando “. /test” debido a que el segundo, únicamente para ejecutar y no compilar.

3. Utilice el simulador sim-outorder por medio del uso de la opción -issue:inorder/ outoforder (defecto) , determine si la ejecución fuera de orden proporciona algún beneficio para la aplicación y explique según la teoría.

Al ejecutar el archivo test-printf.c con la optimización -O2, el simulador de sim-outorder arrojó las siguientes estadísticas:

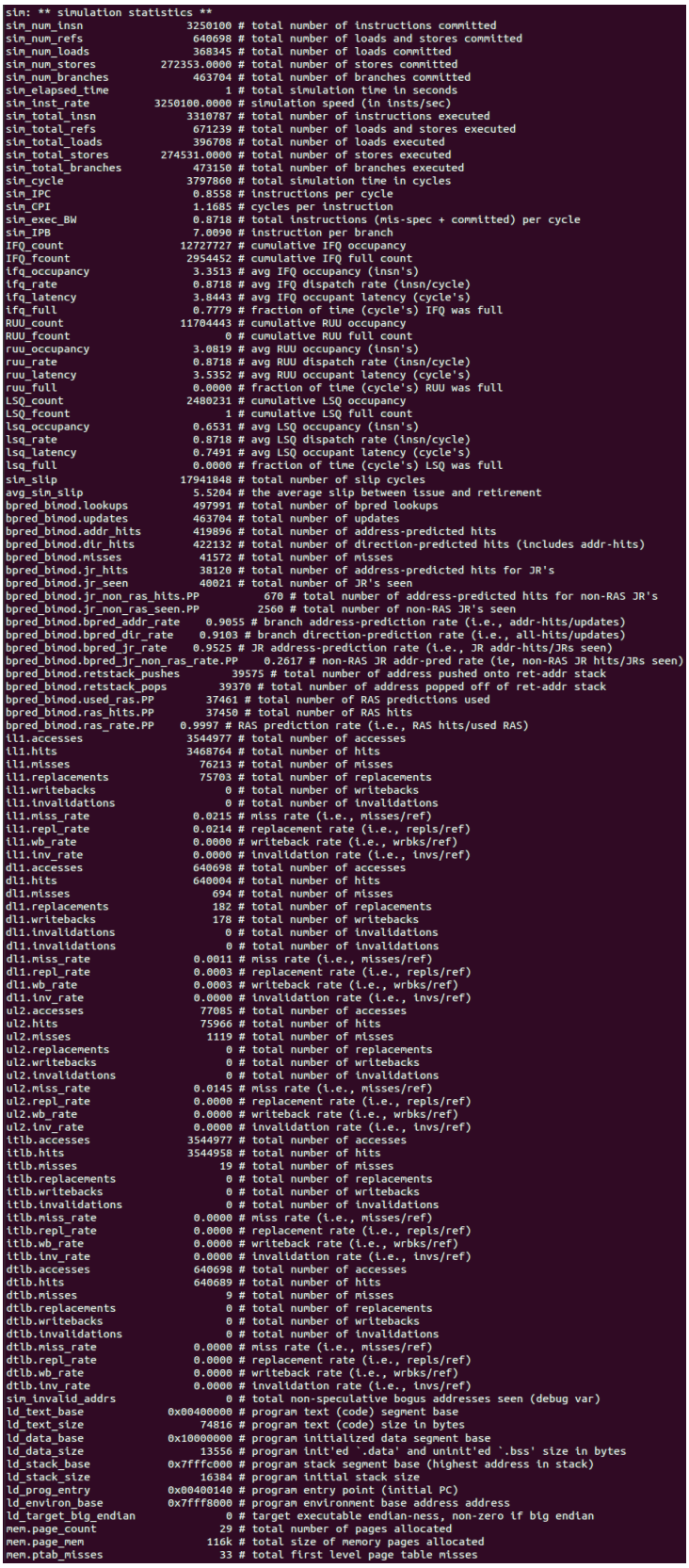

Figura 5. Estadísticas del simulador de sim-outorder del archivo test-printf.c.

Si se comparan los datos de las figuras 4 y 5 se puede observar que la ejecución fuera de orden generó las siguientes diferencias respecto a la ejecución en orden:

- Menor número de instrucciones involucradas.
- Menor velocidad de simulación.
- Menor número de páginas asignadas.
- Menor tamaño total de las páginas de memoria asignadas.
- Mayor acceso total a la tabla de páginas.

Al utilizar un simulador superescalar, es de esperar que la ejecución fuera de orden proporcione más beneficios que la ejecución en orden dado que dicho tipo de procesadores dispone de unidades funcionales replicadas, pipelines independientes, mayor nivel de paralelismo y dinamismo. Es por esta razón que es posible ejecutar una aplicación, con un flujo diferente al que se encuentra programada, para obtener un mayor rendimiento y eficiencia al momento de ejecutar dicho programa.

En el caso en cuestión, la ejecución del archivo test-printf.c con el simulador sim-outorder, al reordenar las múltiples instrucciones de dicha aplicación, se evitó la inserción de una gran cantidad de instrucciones NOP o stalls necesarias para manejar diversos tipos de riegos. De menara que, se generó una disminución en la cantidad de instrucciones, asignación de páginas, uso de memoria y rapidez de simulación. Por otra parte, la ejecución fuera de orden generó un incremento en el acceso total de páginas debido al amplio uso del paralelismo a nivel de instrucción.

4. El simulador sim-outorder le permite modificar el número de unidades funcionales. Modifique el número de unidades funcionales y el issue, analice como beneficia el rendimiento (comience desde 1 unidad funcional). Explique sus resultados con base en el código fuente de la aplicación que se está analizando. Contraste el funcionamiento con respecto al test-fmath.c.

Para esta sección se analizará la ejecución fuera de orden con:

- 1 alu entera, 1 multiplicación entera, 1 alu flotante y 1 multiplicación flotante.
- 8 alu’s enteras, 8 multiplicaciones enteras, 8 alu’s flotantes y 8 multiplicaciones flotantes.

Luego de realizar cada una de las ejecuciones fuera de orden con las unidades anteriores, se obtuvieron los siguientes resultados:

Instrucciones por ciclo:

- -sim_IPC => 0.9109
- +sim_IPC => 1.4332

Tiempo total de simulación en ciclos:

- -sim_cycle => 3568068
- +sim_cycle => 2267801

Ciclos por instrucción:

- -sim_CPI => 1.0978
- +sim_CPI => 0.6978

Es importante aclarar que los que tienen el símbolo - tienen solo 1 unidad funcional y los que tienen símbolo + tienen 8 unidades funcionales.

Tal y como se puede observar en los resultados anteriores, mientras mayor sea la cantidad de unidades funcionales que se posea, mayor será el rendimiento que se obtendrá al momento de ejecutar una determinada aplicación. En particular, el código fuente del archivo test-printf.c está formado por múltiples llamadas a funciones, ciclos anidados, condiciones, estructuras y múltiples impresiones en consola, la mayoría de ellas pueden ser ejecutadas en diferente orden sin problema dado que no poseen ningún tipo de dependencia. Dado lo anterior, fue posible ejecutar 0.5223 más instrucciones en 1300267 ciclos menos a 0.4 ciclos menos por instrucción con 8 unidades funcionales.

Caso similar a la ejecución fuera de orden del archivo test-fmath.c ya que se obtuvo una ganancia de 0.2525 más instrucciones en 30486 ciclos menos a 0.3796 ciclos menos por instrucción con 8 unidades funcionales, considerando que dicho algoritmo no dispone de la misma complejidad que la aplicación del archivo test-printf.c.

Referencias

Optimize Options (Using the GNU Compiler Collection (GCC)). GCC GNU ORG. https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html

Optimizing gcc Compilation. (2021, 31 marzo). Oracle. https://docs.oracle.com/en/operating-systems/oracle-linux/6/porting/ch04s03.html