Estudi Previ Pràctica 1

Marta Granero I Martí

24 Febrer 2021

ESTUDI PREVI

1. Busca que significa hacer “inlining” de una función.

   Quan fem "inlining" d'una funció significa que podem indicar-li al compilador GCC que integri el codi de la funció al codi font(.c) a cada lloc on es cridada. Això fa que l'execució sigui més ràpida el·liminant la sobrecàrrega de crides a les funcions, i pot suposar una optimització important. L'efecte sobre la mida del codi és una mica imprevisible, el codi objecte(.o) pot ser més gran o més petit segons cada cas en concret. "L'inline" de les funcions és una optimització i només funciona quan fem una optimtzació de la compilació, és a dir, si no fem servir `-O`, cap funció realment és "inline". També hauríem de tenir en compte que si realment un funció no la cridem amb freqüència, fer-la inline no milloraria notablement el temps d'execució del programa. Nogensmenys, el compilador pot decidir fer una funció "inline" encara que no ho haguem assignat en el codi.

2. Busca que opción de compilación individual (no grupos de opciones como -O) de gcc permite al compilador hacer “inlining” de todas las funciones simples. Averigua si esta opción se activa al activar las optimizaciones -O2 del compilador. ¿Para qué sirve la opción -finline-limit?

   Podem fer servir la opció de compilació: `-finline-functions` per fer totes les funcions simples inline.
   Aquesta opció s'activarà si s'activen les optimitzacions `-O2, -O3` i `-Os`.
   La opció `-finline-limit` permet un control aproximat del límit de mida de les funcions que podem fer inline. Té un comportament per defecte si no li indiquem n(sent n la mida de les funcions que poden ser inline) tal que (`-finline-limit`), o per altra banda, `-finline-limit=n`.

3. Explica una forma práctica de saber si en un programa ensamblador existe la función “Pedrito”. Explica cómo averiguar si, además de existir, esa función es invocada o no.

   Podem cercar al programa ensablador(.s) a l'apartat `.globl` si la funció "Pedrito" hi apareix. I podem saber si una funció és invocada si apareix la instrucció `call _Pedrito`, o `callq _Pedrito`

4. Calcula cuántas instrucciones estáticas y dinámicas tiene el anterior segmento de código. Si la ejecución tarda 10 ms y 14000000 de ciclos, calcula cuántos MIPS y que IPC, CPI y frecuencia tiene el procesador al ejecutar este código.

   # Instruccions dinàmiques = 5000000
   # Instruccions estàtiques = 5

   MIPS = $\frac{\#insdin}{10^6 * 10 * 10^{-3}}$ = 500 mil·lions ins

CPI = $\frac{Cicles}{Instruccions} = \frac{14000000}{5000000} = 2.8$ cicles/ins

IPC = $\frac{1}{CPI} = 0.357$ ins/cicles

f = # cicles per segon = $\frac{N \times CPI}{Temps\ exec} = \frac{5000000 \times 2.8}{10 * 10^{-3}} = 1.4$*$10^9 Hz = 1.4GHz$

5. En este caso el programa tarda 5 ms y 7000000 de ciclos en ejecutarse. Calcula nuevamente los MIPS, el CPI y la frecuencia del procesador e indica cuál es el Speedup respecto a la versión anterior. Intenta explicar cuáles son las posibles fuentes de las igualdades y diferencias observadas con respecto al apartado anterior.

   # Instruccions dinàmiques = 4000000
   # Instruccions estàtiques = 4
   MIPS = $\frac{\#insdin}{10^6 * 5 * 10^{-3}} = 800$ mil·lions ins

   CPI = $\frac{Cicles}{Instruccions} = \frac{7000000}{4000000} = 1.75$ cicles/ins

   f = # cicles per segon = $\frac{N \times CPI}{Temps\ exec} = \frac{4000000 \times 1.75}{5 * 10^{-3}} = 1.4$*$10^9 Hz = 1.4GHz$

   $SpeedUp$ =Tempso4$\frac{}{Tempsmill5} = \frac{10}{5} = 2$

   Tardem la meitat de temps i la meitat de cicles en executar el programa perquè hem vist que el # d'instruccions dinàmiques ha disminuït en un factor de 1.25x ja que hem optimitzat el programa. A més a més, el CPI es veurà decrementat en un factor de 1.6x, tardant molts menys cicles per instrucció. D'altra banda, es mantindrà la freqüència del processador ja que dependrà del cicles i del temps d'execució i és degut a que els dos es redueixen en un factor de 2x. Per dir-ho d'alguna forma, se'ns contraresten els factors.

6. Si el código anterior es parte de un programa que tarda en ejecutarse (en total) 200ms (compilado todo él con la opción -O0), calcula cuál es el Speedup máximo del programa total si consiguiéramos ejecutar el código anterior de manera instantánea. Calcula también el Speedup del programa completo obtenido al compilar solo el código del ejemplo con la opción -O.

   -O0 equival a no optimitzar
   Codi anterior tarda = 4.417000 ms
   FracciómillioradaMàxim = $\frac{4.417000}{200} = 2.2085\%$
   SpeedUp màxim(Guany màxim) = $\frac{1}{1 - 0.022085} = 1.023$

   ```
   gcc -O Simple.c
   ./a.out
   ```

   Mil·lisegons = 0.52600 FracciómillioradaMàxim = $\frac{0.526}{200} = 0.263\%$
   SpeedUp(Guany) = $\frac{1}{1 - 2.63 * 10^{-3}} = 1.002$

7. A partir de las herramientas de medida prácticas que se han visto en los apartados anteriores define una forma (qué medir, cómo y qué hacer con el número) para medir el rendimiento (MIPS y CPI) del programa en C que acabamos de ver. Aunque un programa donde se miden los ciclos y el tiempo de ejecución ejecuta más instrucciones que uno que no, no es necesario que tengáis esto en cuenta en esta práctica ya que el efecto es muy pequeño.

Les hem obtingut executant `valgrind --tool=lackey ./a.out`(mirant l'apartat `guest instrs`). També necessitem, el seu temps d'execució en ms obtingut amb la funció `GetTime()` del programa `tiempo.c`.

D'altre banda, pero obtenir el IPC, necessitarem el # Cicles per segon que executa el programa, i.e, la freqüència del processador. Per portar a terme el càlcul, necessitarem fer el quocient de # Instruccions dinàmiques que executa el programa entre el # Cicles, o més aviat, fer l'invers, si ja l'hem calculat, del CPI.

8. Un programa dado lo hemos ejecutado 5 veces con los siguientes resultados de tiempo de ejecución: 10 ms, 8ms, 13 ms, 4ms y 2ms. Calcula la media aritmética y geométrica de las 5 medidas. A continuación descarta las dos medidas extremas y calcula de nuevo la media geométrica y aritmética de los resultados. Explica cuales son los principales efectos que observáis.

   Mitjana Aritmètica = 7.4 ms
   Mitjana Geomètrica = $\sqrt[5]{1081342} = 6.082$ ms

   Sense les dues mesures extremes:

   Mitjana Aritmètica = 7.333 ms
   Mitjana Geomètrica = $\sqrt[3]{10 \times 8 \times 4} = 6.840$ ms

   Descartant els valors extrems, puc observar que la mitjana aritètica no ha set suceptible a aquests, ja que probablement poden ser valors possibles, i s'han obtingut en les mateixes condicions que els altres valors. La mitjana geomètrica té la particularitat que no és tant sensible com la mitjana aritmètrica als valors extrems, la mitjana aritmètica ens pot donar un estimador raonable del promig de temps d'execució en ms del programa