2º curso / 2º cuatr.

Grado en Ing. Informática

ETSIIT Universidad de Granada

Arquitectura de Computadores: Prácticas

Optimización de Código en Arquitecturas ILP

Material elaborado por los profesores responsables de la asignatura:
Julio Ortega, Mancia Anguita

0 Contenido

1 Bloque Práctico 4. Optimización de código

1.1 Objetivos

En este bloque el estudiante aprenderá alternativas para optimizar código teniendo en cuenta las características de la microarquitectura y la arquitectura del computador. Se pretende poner de manifiesto la importancia que en el proceso de optimización tiene el conocimiento de la máquina donde se ejecuta el programa. Si bien es posible escribir programas correctos para un computador a partir del modelo de programación propio del lenguaje de alto nivel que se utiliza, y sin conocer prácticamente nada de las características de la máquina, la generación de código eficaz, que extraiga el máximo partido de las características del computador, requiere que se tengan en cuenta detalles de la arquitectura (realizando ciertas optimizaciones a nivel ensamblador) y de la forma en que se ejecutan las instrucciones máquina (características de la microarquitectura).

1.2 Seminario

El seminario de este bloque práctico presenta las principales alternativas para la mejora de prestaciones en arquitecturas con paralelismo entre instrucciones, particularmente las relacionadas con el acceso a memoria, la predicción de saltos, y la ejecución de operaciones, también se ilustrará la posibilidad de utilizar varios núcleos que ya se ha visto al utilizar OpenMP.

1.3 Práctica o trabajo a desarrollar

En las clases presenciales y en casa se trabajarán los siguientes ejercicios y cuestiones:

1. Para el núcleo que se muestra en la Figura 1, y para un programa que implemente la multiplicación de matrices (use variables globales):

   1.1 Modifique el código C para reducir el tiempo de ejecución del mismo. Justifique los tiempos obtenidos (use `-O2`) a partir de la modificación realizada. Incorpore los códigos modificados en el cuaderno.

1.2 Genere los códigos en ensamblador con `-O2` para el original y dos códigos modificados obtenidos en el punto anterior (incluido el que supone menor tiempo de ejecución) e incorpórelos al cuaderno de prácticas. Destaque las diferencias entre ellos en el código ensamblador.

1.3 (Ejercicio EXTRA) Intente mejorar los resultados obtenidos transformando el código ensamblador del programa para el que se han conseguido las mejores prestaciones de tiempo

2. El benchmark Linpack ha sido uno de los programas más ampliamente utilizados para evaluar las prestaciones de los computadores. De hecho, se utiliza como base en la lista de los 500 computadores más rápidos del mundo (el Top500 Report). El núcleo de este programa es una rutina denominada DAXPY (***D**ouble precision- real **A**lpha **X** **P**lus **Y***) que multiplica un vector por una constante y los suma a otro vector (Lección 3/Tema 1):

for (i=1;i<=N,i++) y[i]= a*x[i] + y[i];

1.1 Genere los programas en ensamblador para cada una de las opciones de optimización del compilador (`-O0`, `-O2`, `-O3`) y explique las diferencias que se observan en el código justificando las mejoras en velocidad que acarrean. Incorpore los códigos al cuaderno de prácticas y destaque las diferencias entre ellos.

1.2 (Ejercicio EXTRA) Para la mejor de las opciones, obtenga los tiempos de ejecución con distintos valores de `N` y determine para su sistema los valores de `Rmax` (valor máximo del número de operaciones en coma flotante por unidad de tiempo), `Nmax` (valor de `N` para el que se consigue `Rmax`), y `N1/2` (valor de N para el que se obtiene `Rmax`/2). Estime el valor de la velocidad pico (`Rpico`) del procesador (consulte en [4] el número de ciclos por instrucción punto flotante para la familia y modelo de procesador que está utilizando) y compárela con el valor obtenido para `Rmax`. Consulte la Lección 3 del Tema 1.

```
struct {
        int a;
        int b;
}  s[5000];

main()
{
   …
   for (ii=0; ii<40000;ii++) {
      X1=0; X2=0;
      for(i=0; i<5000;i++)  X1+=2*s[i].a+ii;
      for(i=0; i<5000;i++)  X2+=3*s[i].b-ii;

      if (X1<X2)  R[ii]=X1  else  R[ii]=X2;
   }
   …
}
```

Figura 1. Núcleo de programa en C para el ejercicio 1.

NOTA: Se debe incluir en el cuaderno de prácticas: (1) la versión de `gcc` que se está usando y (2) todos los códigos, todos los resultados de ejecución, todas las gráficas, todas las tablas, todos los razonamientos, etc. Incluya en el fichero `.zip` que entregue el cuaderno de prácticas, el fichero `/proc/cpuinfo` de la máquina en la que ha tomado las medidas y todos los ficheros fuente.

2 Bibliografía

2.1 C/C++

[1]. C *reference card*: http://users.ece.utexas.edu/~adnan/c-refcard.pdf

[2]. Tutorial de programación en C http://www.cs.cf.ac.uk/Dave/C/CE.html

2.2 Gcc

[3]. The GNU Compiler Collection manuals: http://gcc.gnu.org/onlinedocs/

2.3 Optimización de Código

[4]. Manuales de Intel para procesadores x86 (IA-32), especialmente el manual de optimización de código (busque en la web de referencia *Software Optimization Reference Manual*) http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html

[5]. [Gerber 2006] R. Gerber, et al. "The Software Optimization Cookbook. High Performance Recipes for the IA-32 Platforms". Intel Press, 2006.

[6]. [Fog 2004] A. Fog. "How to Optimize for the Pentium family of microprocessors", http://cr.yp.to/2005-590/fog.pdf, 2004

[7]. [Gerber 2002] R. Gerber. "The Software Optimization Cookbook. High Performance Recipes for the Intel Architecture". Intel Press, 2002.

[8]. Web que genera el ensamblador para un código Fuente dado con diferentes compiladores y opciones de optimización: http://godbolt.org/