Práctica 6: Medida de Prestaciones
30235 Procesadores Comerciales - Grado Ingeniería Informática
Esp. en Ingeniería de Computadores

Pablo Ibáñez Marín y Jesús Alastruey Benedé
Área Arquitectura y Tecnología de Computadores
Departamento de Informática e Ingeniería de Sistemas
Escuela de Ingeniería y Arquitectura
Universidad de Zaragoza

19-mayo-2020

Resumen

En esta práctica se evalúan las prestaciones de una máquina con un procesador RISC (Sun UltraSPARC T2) y de otra con procesador CISC (Intel Core i5) mediante la ejecución de un programa intensivo de cálculo en coma flotante. También se medirá la influencia del compilador en el rendimiento. Los índices escogidos van desde medidas independientes de la arquitectura (MFLOPS) hasta índices muy dependientes de la arquitectura/implementación (CPI).

Introducción

En las revistas de informática aparecen anuncios de computadores que contienen números acerca de sus prestaciones. No siempre estos índices tienen sentido, están en contexto o son reproducibles. Por ejemplo, ¿para qué sirve saber la máxima velocidad a la que puede trabajar el hardware, velocidad de pico [1]? Lo que importa en realidad es resolver problemas en el menor tiempo y/o con la máxima precisión posible. Una forma de medir la calidad del computador consiste en tomar medidas durante la ejecución de programas de prueba (*benchmarks*). Cualquier programa de prueba puede orientarnos en la comparación entre computadores si tenemos claro que:

a) El compilador va a representar un papel importante, un hardware superior puede ser la base de un computador mediocre si el compilador no es bueno.
b) Los resultados deben ser comparables y reproducibles: lista exhaustiva de condiciones de contorno (capacidad de los distintos niveles de cache y memoria principal -DRAM-, prestaciones de disco y de bus, nivel de optimización del compilador, número de procesos en la máquina, versión del sistema operativo…).

En esta práctica se medirán distintos índices de prestaciones del computador (MIPS, MFLOPS, CPI …) ejecutando un pequeño programa núcleo (*kernel*). Siempre teniendo en cuenta el tiempo de ejecución medido, podrán obtenerse índices operaciones/tiempo (a partir del número de operaciones realizadas), instrucciones/tiempo (a partir del número de instrucciones ejecutadas) …

LINPACK (LINear Algebra PACKage) es un programa de prueba muy utilizado para medir la potencia del computador en un ámbito de cálculo vectorial en coma flotante. El programa de prueba LINPACK inicializa

[1] De pico, es decir de "boca". Velocidad que por definición nunca se consigue en problemas reales.

una matriz y un vector con datos aleatorios, realiza una descomposición triangular superior/inferior (L/U) mediante eliminación Gaussiana con pivotado parcial, sustituye y finalmente verifica la respuesta. Existen variantes en el tamaño del sistema (matriz 100x100 ó 1000x1000) y en la precisión de las variables (simple, 4 bytes o doble, 8 bytes). La resolución del sistema completo implica $(2n^3)/3 + o(n^2)$ operaciones. La mayoría de estas operaciones se deben al bucle más interno, que resta un múltiplo de la fila pivote a cada fila de la submatriz implicada. Para simplificar el análisis del programa, vamos a experimentar tan sólo con ese bucle más interno sin alterar las conclusiones.

En esta práctica vamos a medir prestaciones de dos máquinas:

- `hendrix`: arquitectura SPARC (RISC). Cualquiera de las dos máquinas del cluster de prácticas del DIIS (`hendrix[01-02].cps.unizar.es`, accesibles como `hendrix-ssh`).
- `labxxx-yyy`: arquitectura x86-64 (CISC). Cualquier PC de una sala de prácticas del DIIS, al que puede accederse de forma local (arranque CentOS) o de forma remota. Todos estos equipos comparten el sistema de ficheros con `hendrix`. Vuestro nombre de usuario, contraseña y directorio de trabajo son los mismos en todas estas máquinas. Podéis acceder a una de estás maquinas (`lab000`) de forma remota mediante ssh (`ssh usuario@lab000.cps.unizar.es`).

Utilizaremos como programa de prueba `gauss.c`, que es una simplificación de LINPACK. Para copiarlo a vuestro directorio (no olvidar el punto final):

```
hendrix: cp /home/chus/pub_code/gauss.c .
```

El fragmento de código a medir es el siguiente (bucle L/U):

```
for (i=0; i<m_size; i++) {
    for (j=0; j<m_size; j++) {
        A[i][j] = A[i][j] - P[i]*B[j];
    }
}
```

Este bucle se encuentra anidado entre otros dos bucles:

- El bucle `limit` (contador `s`) sirve para repetir la operación matricial y conseguir un tiempo suficientemente grande para ser medido con precisión. Si la medida de tiempo no es mayor que 20 veces la resolución de la rutina de medida de tiempos[2], el programa lo advierte: habrá que incrementar el número de iteraciones `limit`.
- El bucle más externo `repeat` (contador `r`) pretende conseguir resultados más fiables (reproducibles) calculando la media de varios experimentos. Si la variabilidad (varianza) de las medidas de dichos experimentos es alta, el programa lo avisará: habrá que realizar una nueva medición incrementando el valor de `repeat`.

Estudiad con detalle el código para comprender bien el papel de los dos bucles más externos.

MFLOPS, Millions of FLOating Point operations per Second

En este apartado se mide la eficiencia del computador ejecutando código de alto nivel. Es equivalente a ocultar los detalles del compilador, del lenguaje máquina y de la implementación en un caja negra y hacer medidas desde el exterior. Se miden prestaciones en términos de operaciones en coma flotante.

En primer lugar hay que compilar el programa `gauss.c` en las dos máquinas (`hendrix0n` y `labxxx-yyy`) y con varios niveles de optimización en cada una de ellas. Para ello, crea dos directorios llamados `sparc` y `x86`, y copia `gauss.c` en los dos directorios. Compila el programa en cada uno de los directorios **usando un terminal con sesión abierta en la máquina que corresponda** (`hendrix0n` o `labxxx-yyy`), con el compilador y con los niveles de optimización que se indican en la siguiente tabla:

[2]Resolución de la llamada al sistema `times()` o `clock()`.

Table 1: Órdenes de compilación del programa `gauss.c`.

Máquina	Sin optimización	Con optimización
`hendrix`	`cc gauss.c -xO1 -o g1.sparc`	`cc -xO2 gauss.c -o g2.sparc`
`labx-y`	`gcc gauss.c -O0 -o g0.x86`	`gcc -O3 gauss.c -fno-tree-vectorize -o g3.x86` `gcc -O3 gauss.c -o g3.x86.vec`
optativo		`gcc -O3 -march=native gauss.c -o g3.x86.nat`

En el caso de CentOS sobre x86, comprueba que estés utilizando la versión 9.2.0 de `gcc`:

```
$ gcc -v
gcc versión 9.2.0 (GCC)
```

Puede verificarse que los binarios generados son los correspondientes a las dos arquitecturas con la orden `file`:

```
hendrix01:~/sparc/ file gauss0.sparc
gauss0: ELF 32-bit MSB executable SPARC32PLUS Version 1,
V8+ Required, dynamically linked, not stripped

lab000:~/x86/ file gauss0.x86
gauss0: ELF 64-bit LSB executable, x86-64, version 1 (SYSV),
dynamically linked (uses shared libs), for GNU/Linux 2.6.32, not stripped
```

Ejecuta las distintas versiones del programa con diferentes tamaños de la matriz A (16x16, 64x64, 256x256, 1024x1024, 4096x4096).

Hay que ajustar el valor de la variable `limit` para cada tamaño de matriz, de modo que el tiempo de ejecución medido en cada iteración del bucle más externo (bucle `repeat`) sea mayor que 20 veces la resolución del temporizador. Así se limita el error de medición debido a la resolución del temporizador. Asimismo hay que ajustar el valor de la variable `repeat` de forma que se detecten experimentos poco fiables (probar con 5 ó 10 repeticiones).

Calcula los MFLOPS para cada tamaño y nivel de optimización (velocidad de las operaciones suma y producto de coma flotante efectuados por el programa). La cifra cambia según el tamaño de la matriz, ¿por qué?

El procesador SPARC empleado es un UltraSPARC T2[3] de 4 núcleos, cada uno capaz de ejecutar 8 threads (más detalles en la sección 7). Su frecuencia de reloj es 1165 MHz, y cada núcleo es capaz de lanzar dos instrucciones por ciclo, de las que una puede ser de punto flotante. Por tanto, puede ejecutar 1.165 GFLOPS de pico.

El procesador x86-64 en cada equipo de los laboratorios L0.04 o L1.02 es un Intel i5-4590 (Haswell) a 3.3 GHz (más detalles en la sección 7). Cada uno de sus 4 núcleos dispone de dos unidades funcionales capaces de ejecutar la instrucción FMA (*fused multiply-add*), por lo que en principio cada núcleo ejecutaría un máximo de 13.2 GFLOPS (2 UFs x 2 FLOPs/UF x 3.2 GHz). Sin embargo, puede aumentarse este valor mediante el uso de las instrucciones SIMD (Single Instruction Multiple Data). En concreto, una CPU de la microarquitectura Haswell soporta instrucciones AVX (Advanced Vector Extensions), capaces de empaquetar 4 operaciones de coma flotante de doble precisión en una única instrucción. De esta forma, las 2 unidades funcionales pueden lanzar 4 operaciones cada ciclo, lo que supone 52.8 GFLOPS.

¿Por qué la cifra que habéis obtenido al ejecutar `gauss` es muy inferior a los MFLOPS de pico anunciados por la propaganda? ¿Qué tipo de programas rondaría los MFLOPS de pico?

[3]También conocido como Niágara 2.

MIPS, Millions of Instructions Per Second[4]

A continuación abriremos parcialmente la caja negra y consideraremos el nivel lenguaje máquina. Conseguid el listado en ensamblador del programa para cada arquitectura y nivel de optimización utilizando la opción -S del compilador. Para un nivel de optimización n:

```
hendrix:~/ cc -xOn -S -o g1.sparc.s gauss.c

labxxx-yyy:~/ gcc -On -S -o g0.x86.s gauss.c
```

Las instrucciones correspondientes al cuerpo del bucle L/U[5] de cada una de las versiones se encuentran en las líneas que se indican en la siguiente tabla:

Table 2: Ubicación del cuerpo del bucle L/U.

Máquina	Nivel optimización	Líneas inicio / final
`hendrix`	`-xO1`	683 / 726
	`-xO2`	623 / 634
`labxxx-yyy`	`-O0`	212 / 245
	`-O3 -fno-tree-vectorize`	236 / 245
	`-O3`	293 / 301
	`-O3 -march=native`	321 / 327

Para cada nivel de optimización y arquitectura, ¿cuántas instrucciones correspondientes al cuerpo del bucle L/U (`A[i][j] = ...`) se ejecutan en cada iteración? Tener presente que la arquitectura SPARC es 1-retardada.

Calcular las velocidades en MIPS suponiendo que los tiempos medidos en el apartado anterior se deben exclusivamente a la ejecución de las instrucciones correspondientes al cuerpo del bucle L/U. Ayuda: en las versiones sin optimizar, las instrucciones asociadas al cuerpo del bucle L/U se ejecutan `m_size`^{2} veces. En las versiones optimizadas no ocurre así. Por ejemplo, en la versión `-xO3` de SPARC pueden observarse las agresivas transformaciones que ha realizado el compilador con la orden `er_src gauss3.sparc` (previamente hay que compilar añadiendo la opción `-g`).

Cada núcleo del procesador UltraSPARC T2 puede lanzar hasta 2 instrucciones por ciclo a 1165 MHz, por tanto puede ejecutar 2.33 GIPS pico por núcleo. En cuanto al Intel i5-4590, cada núcleo puede jubilar hasta 4 instrucciones[6] por ciclo a 3.3 GHz, lo que supone 13.2 GIPS pico por núcleo.

¿Por qué la cifra que habéis obtenido al ejecutar gauss es muy inferior a los MIPS de pico anunciados por la propaganda? ¿Qué tipo de programas conseguiría los MIPS de pico?

CPI (Ciclos Por Instrucción)

Conociendo el tiempo de ciclo del procesador, calculad el CPI de este programa.

¿Para qué tamaño y nivel de optimización se obtiene el mayor CPI? ¿Y el menor? ¿Cómo se explican las diferencias, si existen? ¿Menor CPI implica siempre menor tiempo de ejecución?

[4] Medida desprestigiada. Por ejemplo, algunos dicen que MIPS es *Meaningless Indicator of Processor Speed*, o sea, indicador inútil de la velocidad del procesador.

[5] Línea de código `A[i][j] = A[i][j] - P[i]*B[j];`

[6] En realidad puede jubilar 4 uops/ciclo, que en el mejor de los casos se corresponden con 4 instrucciones/ciclo.

Ancho de banda

En este apartado debéis estudiar cuidadosamente el código ensamblador. Considerad solamente las versiones optimizadas (la escalar para el caso de x86). Calculad el ancho de banda en megabytes por segundo (MBps) de:

- Las instrucciones, es decir, ¿cuántos MBps de instrucciones está consumiendo el procesador? Para determinar los bytes de código buscados en x86 sigue el siguiente procedimiento:
  - Compilar añadiendo la opción `-g`. Por ejemplo:
    ```
    labxxx-yyy:~/ gcc -O3 -fno-tree-vectorize -g -o g3.x86 gauss.c
    ```
  - Generar ensamblador:
    ```
    labxxx-yyy:~/ objdump -Sd g3.x86
    ```
  - Localizar el código y contar los bytes que ocupan las instrucciones
- Los datos, es decir, ¿cuántos MBps de datos en memoria está consumiendo el procesador? Considerar el tráfico correspondiente a las lecturas y a las escrituras.
- El banco de registros, o sea MBps de lecturas y escrituras en cualquier registro (salvo el contador de programa -PC-).

Rellenad una tabla en la que se especifiquen, para las versiones optimizadas de las dos arquitecturas, los bytes de cada tipo leídos/escritos en una iteración del bucle interno (ver la siguiente tabla).

Table 3: Bytes leídos y escritos en una iteración del bucle interno.

Máquina	Instrucciones (bytes leídos)	Datos (bytes leídos/escritos)	Banco de registros (bytes leídos/escritos)
`hendrix`	ir	dr / dw	r / w
`labxxx-yyy`	ir	dr / dw	r / w

Rellenad ahora otra tabla en la que, para cada tamaño de matriz y para cada arquitectura, se especifiquen los tiempos de ejecución y los MBps de cada tipo.

(Optativo) Medidas sobre otras máquinas

Puede realizarse el análisis de otras máquinas, por ejemplo, vuestro computador. Se recomienda hacerlo sobre un sistema Linux, ya que de este modo pueden utilizarse el fichero fuente `gauss.c`. En caso de hacerlo sobre una máquina x86 (AMD, Intel …) con Windows, tendrás que modificar el fichero fuente `gauss.c`.

Características de los computadores

A continuación se detallan las características de los equipos de prácticas. En el caso de los PCs de los laboratorios puede haber algún equipo diferente, por lo que se recomienda verificar el tipo de sistema con las órdenes `hinv`, `cat /proc/cpuinfo` y `dmesg`.

hendrix01 y hendrix02

- Nombre / dir.IP: hendrix[01-02].cps.unizar.es / 155.210.152.[183-184]
- Máquina: Sun Enterprise-T5120
- CPU: UltraSPARC-T2 1165 MHz, 4 núcleos, 8 threads por núcleo. Cada núcleo puede lanzar hasta 2 instrucciones por ciclo, entre ellas máximo 1 de coma flotante

- Arquitectura: Sparc (sparcv9+vis)
- Cache L1:
  - Instrucciones: 16 KB, 8-way, bloques de 32 bytes (por núcleo)
  - Datos: 8 KB, 4-way, bloques de 16 bytes, write-through, write-around
- Cache L2: compartida por los núcleos, 4 MB, 16-way, bloques de 64 bytes, 8 bancos
- Memoria principal: 3968 MB DRAM
- Disco duro: 2x136 GBytes HD
- Sistema Operativo: SunOS, version 5.10.
- Compiladores: Sun Studio 12 C 5.11, gcc (5.2.0)

Más información: órdenes version, sprtconf, uname -a

laboratorios L0.04 (lab004) y L1.02 (lab102)

- Nombre / dir.IP: lab102-nnn.cps.unizar.es, con nnn=[190-210] / 155.210.154.[190-210]
- CPU: Intel i5-4590 @ 3.3GHz (4 núcleos x 1 thread/núcleo), microarquitectura Haswell. Cada núcleo decodifica hasta 4 instr. por ciclo (1 compleja y 3 sencillas), 2 UFs de punto flotante de 256 bits (2 FMA)
- Arquitectura: Intel 64 (x86-64)
- Cache L1: 4 x (32KB instrucciones + 32KB datos)
- Cache L2: 4 x 256 KB
- Cache L3: 6 MB KB (en chip, compartida por los cuatro núcleos).
- Memoria principal: 16 GB DRAM.
- Sistema operativo:
  - lab102: 3.10.0-957.5.1.el7.x86_64 x86_64 GNU/Linux CentOS Linux 7.4.1810
- Compilador: gcc version 9.2.0 (GCC).

Código `gauss.c`

```
/*
 * Fichero: gauss.c
 * Version: 1.3 - Dic 2010
 *
 * Adaptación de gauss.c  (CS252, Spring 1994)
 *
 * Pablo Ibáñez
 * Jesús Alastruey
 */

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/times.h>
#include <time.h>
#include <limits.h>
#include <math.h>

 /* No hay que cambiar MAXSIZE en cada ejecución (se define constante)
 *  por tanto, no es necesario recompilar para ejecutar
 * el programa con distintos tamaños de problema (matriz, vector)
 */
#define MAXSIZE     4104
```

```
double A[MAXSIZE][MAXSIZE];
double P[MAXSIZE];
double B[MAXSIZE];

int
main(int argc, char *argv[])
{
  int r, s, i, j, intervalos;
  int repeat, limit, m_size;
  struct tms time_info;
  clock_t user0, user1, sample_cpu, sample_elapsed;
  clock_t total = 0, total_sqrd = 0;
  clock_t tinicio, tfinal, ttotal = 0;
  clock_t etime0, etime1;
  float timer_resolution;
  float loop_mean, loop_var, test_mean, loop_std_ratio_c;

  if (argc < 4) {
    printf("Uso:%s <tamaño-matriz> <numero-de-iteraciones> <numero-de-pruebas>\n", argv[0]);
    exit(-1);
  }

  m_size = atoi(argv[1]);
  if (m_size > MAXSIZE) {
      fprintf(stderr,"ERROR: el tamaño de la matriz (%d) debe ser menor que MAXSIZE (%d)\n",
            m_size, MAXSIZE);
      exit(1);
  }

  limit = atoi(argv[2]);
  repeat = atoi(argv[3]);
  if ((limit < 1) || (repeat < 1)) {
      fprintf(stderr,"ERROR: el número de iteraciones y el de pruebas debe ser mayor que 0\n");
      exit(1);
  }

  /* Inicializacion */
  for (i = 0; i < m_size ;i++) {
    B[i] = (double) i + 4.0;
    P[i] = (double) i * 2.0;
    for (j = 0; j < m_size ; j++) {
      A[i][j] = (double) i + (double) j;
    }
  }

  /*
   * timer_resolution = resolucion en microsegundos del temporizador del computador
   * (tiempo en milisegundos entre dos ticks del temporizador)
   * Se calcula a partir del nº de intervalos que el temporizador cuenta cada segundo.
   * Para saber este nº de intervalos hay que usar la llamada sysconf(_SC_CLK_TCK).
   * El programa realiza esta llamada e inicializa la variable timer_resolution.
   */
  intervalos = sysconf(_SC_CLK_TCK);
```

```
timer_resolution = 1000000.0/intervalos;

printf ("\n******** Datos ********\n");
printf("Resolucion del temporizador: %7.1f usg (%d intervalos/seg)\n",
      timer_resolution, intervalos);
printf("Tamaño_matriz, limit, repeat:  %dx%d, %d, %d\n", m_size, m_size, limit, repeat);
/*printf("Tamaño de float: %d\n", sizeof(float)); */
printf("Ejecutando gauss v1.3.\n\n");

 /* Inicio del cálculo del tiempo de ejecución de todos los bucles */
times(&time_info);    /* man -s2 times */
tinicio = time_info.tms_utime;

/* Ejecuta el experimento "repeat" veces */
for (r = 0; r < repeat; r++)
{
  /* Ponemos en marcha el cronometro */
  etime0 = times(&time_info);
  //user0 = time_info.tms_utime + time_info.tms_stime;
  user0 = time_info.tms_utime;

  for (s = 0; s < limit; s++)
  {
    /* Impide optimizaciones agresivas del compilador */
    A[0][0] = 0.0;

    /* Estos dos bucles anidados son que queremos medir: L/U  */
    for (i = 0; i < m_size; i++)
    {
  for (j = 0; j < m_size; j++)
     A[i][j] = A[i][j] - P[i]*B[j];
    } /* Fin L/U */
  } /* Fin s */

  //sleep(1);

  /* Paramos el cronometro */
  etime1 = times(&time_info);
  //user1 = time_info.tms_utime + time_info.tms_stime;
  user1 = time_info.tms_utime;

  sample_cpu = user1 - user0;
  sample_elapsed = etime1 - etime0;

  printf("*** Test %02d = %8.3f usg (%8.3f)\n", r+1,
          sample_cpu*timer_resolution/limit,
          sample_elapsed*timer_resolution/limit);

  if (sample_cpu == 0)
    printf("AVISO: tiempo de ejecución menor que la resolución del reloj\n");
  else if (sample_cpu < 20)
      printf("AVISO: tiempo de ejecución (%4.2f usg) menor que 20 ticks
          del reloj (%4.2f usg)\n", sample_cpu*timer_resolution,
          20.0*timer_resolution);
```

```
    total += sample_cpu;
    total_sqrd += (sample_cpu * sample_cpu);
  } /* fin repeat */

  /* Cálculo del tiempo de ejecución de todos los bucles */
  times(&time_info);
  tfinal = time_info.tms_utime;
  //tfinal = time_info.tms_utime + time_info.tms_stime;
  ttotal = tfinal - tinicio;

  /* Cálculo de la varianza y desviación estándar
   * de las medidas obtenidas por el bucle repeat */
  /* VAR[X] = E[X^2] - E[X]^2 */
  /* STD[X] = sqrt(VAR[X]) */
  loop_mean = (float) total / repeat;
  loop_var = ((float) total_sqrd / repeat) - (loop_mean * loop_mean);
  loop_std_ratio_c = loop_var / (loop_mean*loop_mean);
  if (loop_std_ratio_c > 0.0025)
    printf("AVISO: Varianza de las mediciones demasiado alta\n");

  test_mean = loop_mean / limit;
  printf("\n******** Resultados ********\n");
  printf("Tiempo medio de ejecución de L/U    = %8.3f usg\n", test_mean*timer_resolution);
  printf("Tiempo medio de ejecución de limit  = %8.0f usg\n", loop_mean*timer_resolution);
  printf("Tiempo total de ejecución de repeat = %8.0f usg\n\n", ttotal*timer_resolution);

  exit(0);
}
```