Universidad Rey Juan Carlos

Arquitectura de Computadores/Arquitectura de Sistemas Audiovisuales II

Práctica 6: Benchmark de memoria

Katia Leal Algara

INTRODUCCIÓN:

- Estudio del rendimiento de la jerarquía de memoria
- Obtener el tamaño de las cachés de nivel 1 y 2 y de la memoria principal
- Medidas de ancho de banda
- Mejora del rendimiento por medio de la programación
- Mejora del rendimiento empleando distintas opciones de compilación

A. Características de nuestra máquina

Antes de comenzar con la implementación de los benchmark de memoria vamos a determinar las características de nuestra máquina, paro ello utilizaremos el comando `lshw` con la opción `-short`. Este comando nos permitirá obtener información sobre al capacidad de la caches de nivel 1 y 2 y de la memoria principal. También muestra información sobre el microprocesador. Comprueba la información relativa al procesador, y a las memorias cachés y principal.

B. Estudio del rendimiento de la jerarquía de memoria

Con este ejercicio se pretende medir la velocidad de transferencia entre la CPU y la jerarquía de memoria transfiriendo repetidas veces cada elemento de un array de datos a un registro de CPU y viceversa. **Se realizarán mediciones de tiempos de lectura y escritura**.

Por ejemplo, el código que se encarga de la lectura de memoria a CPU será parecido al siguiente:

```
for (i = 0; i < num_iteraciones; i++)
{
      for (j = 0; j < num_elementos; j++)
      {
            aux = array[j]
      }
}
```

Teniendo en cuenta este código, el número de bytes transferidos será igual a:

num_iteraciones X num_elementos X sizeof(elemento) = N bytes

Además, si medimos el tiempo justo antes de comenzar el proceso de lectura y justo después con la función `gettimeofday()` (man 2 gettimeofday), podremos calcular la velocidad de transferencia en Mb/s. Aunque `aux` es una variable, y por tanto está almacenada en memoria, suponemos que el compilador entre las optimizaciones que aplica a la hora de generar el código máquina correspondiente a este programa, utilizará un registro de la CPU para guardar `aux`.

El tamaño de los bloques transferidos irá variando, comenzando por 1KB hasta 5MB. Por ejemplo, se pueden realizar 6 pruebas, teniendo en cuenta el tamaño de las cachés, se pueden utilizar los siguiente tamaños de bloque: 1KB, 5KB, 50KB, 500KB, 1MB y 5MB. Por la salida se debe sacar una línea indicando para cada prueba el tamaño del bloque empleado y el tiempo empleado en la transferencia de dicho bloque para la lectura y para la escritura, así como la correspondiente velocidad de transferencia en Mb/s. Es decir:

tamaño_bloque tº_lectura Mb/s_lectura tº_escritura Mb/s_escritura

Se debe indicar en cada caso las unidades correspondientes.
El array de datos debe ser de tipo `long double` (12 bytes), haciendo un reserva inicial de 5MB, que se corresponde con el tamaño máximo de bloque. Sin embargo, para la realización de las pruebas ser irán empleando subconjutos de este bloque. El tamaño del bloque viene determinado por:

Tamaño del bloque = num_elementos X sizeof(elemento)

Es fundamental **inicializar el array**, por ejemplo con ceros, al comienzo del programa para que se provoquen todos los fallos de página oportunos antes de realizar las pruebas y no durante las mismas, puesto que los resultados no serían correctos.

Por último, hay que realizar un número suficiente de iteraciones tal que el sistema tarde un tiempo apreciable en realizar las operaciones, y así los resultados sean fiables. Con tal fin, se debe utilizar una cantidad fija de datos totales a transferir en cada prueba, en este caso, 1GB. Por lo tanto, el número de iteraciones se calculará en función del tamaño del bloque y de la cantidad fija de datos a transferir:

num_iteraciones = 1Gbyte / Tamaño del bloque

Debemos tener en cuenta que en un sistema multiprogramado es prácticamente imposible que no se produzcan interferencias en la ejecución de nuestro programa dado que se deben atender interrupciones constantemente y además el planificador de la CPU nos puede desalojar para pasar a ejecutar otro proceso. En nuestro caso, un programa de usuario no puede deshabilitar las interrupciones, pero sí podemos evitar que nuestro programa se vea interrumpido por otros indicando una política y un nivel de prioridad que nos sean más favorables. Esto lo podemos hacer mediante la llamada al sistema `sched_setscheduler()` (man 2 sched_setscheduler), en la que podemos indicar para el proceso actual una política de planificación más favorable, como

SCHED_FIFO y una prioridad máxima de la siguiente manera:

```
…
struct sched_param sp;
int policy;
...
if ((policy = sched_getscheduler(0) == -1))
{
      perror("…");
}

if (policy == SCHED_OTHER)
{
      sp.sched_priority = sched_get_priority_max(SCHED_FIFO);
      sched_setscheduler(0, SCHED_FIFO, &sp);
      ...
} ...
```

Sin embargo, obtendremos un error si el proceso que hace la llamada no tiene los privilegios apropiados. **Solamente los procesos de root tienen permiso para activar la politica SCHED_FIFO**. Por lo tanto, en el caso del laboratorio no podrá lanzarse el programa como root.

Finalmente, se debe ejecutar el programa (denominado **p6_B.c**) varias veces para comprobar que los tiempos obtenidos para cada tamaño de bloque son similares. Se debe compilar el benchmark con el gcc, sin emplear flags de optimización: `gcc -o p6_B p6_B.c`. Se debe guardar la salida de una de estas ejecuciones en un fichero de texto para comparar los resultados obtenidos en base al tamaño de los bloques transferidos, al tipo de acceso, a los distintos niveles en la jerarquía de memoria y al tamaño de las cachés y de la memoria principal, así como de todos aquellas variables que el alumno estime oportunas.

C. Mejora del rendimiento mediante técnicas de programación

En cualquier programa escrito en un lenguaje de alto nivel nos podemos declarar matrices de varias dimensiones, es decir, multidimensionales. Sin embargo, estas matrices se deben almacenar finalmente en memoria, que es básicamente un espacio de almacenamiento lineal. En C, una matriz bidimensional se almacena en memoria colocando las filas una detrás de otra, siendo la lógica del compilador la que permite el acceso a los elementos de la misma por filas y columnas por medio de índices.

Esto quiere decir que cuando accedemos a los elementos de una misma fila (por ejemplo a[0][0], a[0][1], a[0][2], …) en realidad estamos accediendo a posiciones consecutivas de memoria. Sin embargo, si accedemos a los elementos de una misma columna (por ejemplo a[0][0], a[1][0], a[2][0], …), accedemos a posiciones no continúas en memoria.

La siguiente matriz de 4x4:

	0	1	2	3
0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
2	a[2][0]	a[2][1]	a[2][2]	a[2][3]
3	a[3][0]	a[3][1]	a[3][2]	a[3][3]

Se almacenaría en memoria en posiciones consecutivas:

a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[1][0]	a[1][1]
a[1][2]	a[1][3]	a[2][0]	a[2][1]	a[2][2]	a[2][3]
a[3][0]	a[3][1]	a[3][2]	a[3][3]		

Teniendo esto en mente y sabiendo que de la multiplicación de dos matrices A y B, se obtiene como resultado una tercera matriz C, cuyos elementos [i][j] resultan de sumar los productos de cada elemento de la **fila *i*** de la matriz A por cada elemento de la **columna *j*** de la matriz B. Es decir, recorremos los elementos de la matriz A por filas y los de la matriz B por columnas, lo cual significa accesos a posiciones continuas de memoria para el caso de la matriz A y discontinuos para la matriz B, con lo que con mucha probabilidad se

incrementará la talla de fallos de la caché de primer nivel.
Podemos mejorar el rendimiento simplemente **transponiendo la matriz B**, de forma que las filas se transforman en columnas y las columnas en filas. De esta manera el elemento C[i][j] será la suma de los productos de cada elemento de la **fila *i*** de la matriz A por cada elemento de la **fila *j*** de la matriz B. El resultado será el mismo pero la velocidad de ejecución será mayor.

Se debe implementar un programa p6_C.c que multiplicará dos matrices cuadradas de elementos tipo `long double` mediante los dos métodos anteriormente expuestos, midiendo los tiempos de multiplicación en ambos casos. Se deben medir los tiempos para matrices de distintos tamaños, incrementando cada vez el número de bytes transferidos para así desbordar los distintos niveles de caché. Al igual que para el apartado B, se realizará la medición para 6 matrices distintas, de tal manera que la primera vez se transfiera 1KB (suma del número de bytes que ocupan las tres matrices, que son aproximadamente tres matrices de 5x5 elementos cada una).

Las tres matrices (las dos a multiplicar y la matriz resultado) se reservarán al principio del programa con el tamaño máximo que se vaya a usar (el correspondiente a 5MB), se inicializarán con unos valores cualesquiera para provocar los fallos de página correspondientes para finalmente usar subconjuntos de estas matrices en la realización de las 6 pruebas.

Para simplificar, como la matriz B es cuadrada, supondremos que está transpuesta. Lógicamente, en este caso la matriz resultante empleando el segundo método será distinta de la obtenida empleando el primero, pero el resultado de la multiplicación no es indiferente. Lo realmente importante son los tiempos de multiplicación empleando ambos métodos. Por la salida y para cada una de las seis iteraciones se mostrarán los siguientes datos:
Dimensión_matriz tº_metodo1 tº_metodo2 ganancia
La ganancia de rendimiento es igual al tiempo empleado por el algoritmo tradicional partido por el tiempo empleado por el algoritmo mejorado.

Finalmente, se debe ejecutar el programa varias veces para comprobar que los tiempos obtenidos para cada tamaño de matriz son similares. Se debe compilar el benchmark con el gcc, sin emplear flags de optimización: `gcc -o p6_C p6_C.c`. Se debe guardar la salida de una de estas ejecuciones en un fichero de texto para comparar los resultados en base al tamaño de las matrices, a los distintos niveles en la jerarquía de memoria y al tamaño de las cachés y de la memoria principal, así como de todos aquellas variables que el alumno estime oportunas.

D. Influencia de las opciones de compilación

Otro de los factores determinantes a la hora de mejorar el rendimiento de un programa son las opciones de compilación. Si en lugar de utilizar las opciones por defecto, que realizan una compilación para la arquitectura más genérica (En plataformas Intel el software se compila por defecto para procesadores i386), se emplean opciones de compilación más adecuadas a la arquitectura concreta en la que se ejecutará el programa, podremos generar un código ensamblador que obtendrá un mayor rendimiento del hardware.

Algunas de las opciones más importantes a la hora de compilar un programa son (man gcc):

- `-On`: nivel de optimización, con `n` desde 0 hasta 3. Por defecto el compilador no optimiza nada. A mayor nivel de optimización mayor tiempo de compilación, pero se genera un código más rápido y eficiente. El nivel 1 obtiene un rendimiento suficiente sin emplear demasiado tiempo en la compilación.
- `-march=arquitectura`: genera código optimizado para la arquitectura indicada. Esto incluye la utilización de juegos de instrucciones avanzados como MMX o SSE (Streaming SIMD Instructions). Con esta opción también se tienen en cuenta las peculiaridades de la arquitectura. Entre las arquitecturas soportadas están: `pentium-m` (Low power version of Intel Pentium3 CPU with MMX, SSE and SSE2 instruction set support. Used by Centrino notebooks), `pentium4`, `pentium4m` (Intel Pentium4 CPU with

MMX, SSE and SSE2 instruction set support), `athlon`, `athlon-tbird` (AMD Athlon CPU with MMX, 3dNOW!, enhanced 3DNow! and SSE prefetch instructions support), `athlon-4`, `athlon-xp`, `athlon-mp` (Improved AMD Athlon CPU with MMX, 3DNow!, enhanced 3DNow! and full SSE instruction set support)

- `-m128bit-long-double`: emplear 128 bits (16 bytes) para los datos de tipo `long double`, en lugar de los 96 bits que indica el estándar (12 bytes). Esto hace que los datos de este tipo están alineados en memoria, lo que incrementa el rendimiento en las transferencias.

Compile de nuevo los programas p6_B.c y p6_C.c empleando las opciones -O1 y –m128bit-long-double. Incluya los nuevos resultados para cada programa en un fichero de texto y compara el rendimiento obtenido empleando las opciones de compilación indicadas.