Practica 4 Estructura de Computadores

-Javier Bautista Rosell NIA:100315139,
Correo: 100315139@alumnos.uc3m.es

-Alejandro Blanco Samaniego NIA:100315058,
Correo: 100315058@alumnos.uc3m.es

Ejercicio 1

1) La frecuencia del reloj en MH de la maquina en la que se realiza la práctica es de 1200 Mhz. Esto lo hemos obtenido poniendo en el terminal de Linux: cat /proc/cpuinfo y aquí observamos que pone que la CPU va a una velocidad de 1200MH.

2) El tamaño en GB de la memoria principal de la máquina en la que trabajamos es de 3.73393631 GB. Esto lo hemos obtenido poniendo en el terminal: cat /proc/meminfo

3) La máquina tiene 2 niveles de caché. Esto lo hemos obtenido buscando el procesador Pentium(R) Dual-Core CPU T4500 @ 2.30GHz, que es el procesador que tiene el computador de trabajo, en internet, obteniendo de varias web este resultado.

4) El Tamaño de caché Nivel 1 es de: 2 x 32 KB de cache de instrucciones y 2 x 32 KB cache de datos. El Tamaño de caché Nivel 2 compartida es de 1 MB.

5) El tamaño de línea del primer nivel es de 64 bytes en cache de instrucciones y 64 bytes en cache de datos. El tamaño de línea del segundo nivel es de 64 bytes.

6) El número de líneas de una caché se obtiene dividiendo el tamaño de la cache entre el tamaño de cada línea. De esta manera obtenemos:

- El número de líneas de la cache de nivel 1: [(2x32x1024)/64]+[2x32x1024/64]= 2048 líneas

- El número de líneas de la caché de nivel 2: (1024x1024)/ 64= 16384 líneas

Ejercicio 2:

Dadas las siguientes configuraciones de cache de datos:

[1] L1 de 32KB con un tamaño de línea de 32B
L2 de 256KB con un tamaño de línea de 64B

[2] L1 de 32KB con un tamaño de línea de 64B
L2 de 256KB con un tamaño de línea de 128B

[3] L1 de 32KB con un tamaño de línea de 64B
L2 de 256KB con un tamaño de línea de 64B

Apartado 1)

Análisis del número de accesos a memoria debidos a la lectura/escritura de datos que se realizan en las configuraciones dadas:

Configuración 1:

El número de accesos a memoria debido a lectura es: 2102 (por parte de la caché L1)+1165 (por parte de la caché L2)=3267

El número de accesos a memoria debido a escritura es: 16778313(caché L1)+16777695(caché L2)=33555826

Configuración 2:

El numero de accesos a memoria debidos a la lectura es de 1310(parte de la caché L1)+731 (por parte de la caché L2)=2041

El número de accesos a memoria debidos a escritura, es 16777725(por parte de la caché L1)+16777476(caché L2) = 33555201

Configuración 3:

El numero de accesos a memoria debidos a la lectura es de 1310(parte de la caché L1)+1162 (por parte de la caché L2)=2472

El número de accesos a memoria debidos a escritura, es 16777725(por parte de la caché L1)+16777694(caché L2) = 33555419

Apartado 2)

Análisis del número de fallos de caché que se produce en cada nivel.

Configuración 1:

Fallos en la caché de nivel 1: 1678233
Fallos en la caché de nivel 2: 1677860
http://gyazo.com/53487a8ecdcfab5500fac0ebccb33f6c

Configuración 2:

Fallos en la caché de nivel 1: 16779035
Fallos en la caché de nivel 2:16778207
http://gyazo.com/9779b8d5a2d15420458e0f46c4c7cc91

Configuración 3:

Fallos en la caché de nivel 1: 16779035
Fallos en la caché de nivel 2:16777694
http://gyazo.com/1793c0e62e1ef30d567389e97ddbd0a9

Apartado 3)

Indicamos los cambios que realizamos sobre el código inicial a fin de reducir su tasa de fallos. Tras ello, vienen los análisis que confirman nuestro acierto en la modificación del código.

El código inicial proporcionado es el siguiente:

```
#include <stdio.h>
#include <stdlib.h>

#define SIZE 4096
int array[SIZE][SIZE];

int main(int argc, char *argv[])
{
        int i;
        int j;
        for(i = 0; i < SIZE; i++)
        {
                for(j = 0; j < SIZE; j++)
                {
                        array[j][i] = 25;
                }
        }
}
```

Aquí vemos un fallo clave, sencillo y fácil de solucionar, en la línea: “array [j] [i]=25” El problema está en que j es modificado más veces que i, y se usa para indicar las columnas de la matriz. C guarda los datos de las filas de forma contigua en memoria, pero no las columnas, de esta manera tenemos que acceder muchas más veces a memoria. Cuando la caché se llene, se sobrescribirán datos que se volverán a usar.

Entonces cambiamos esta línea del programa por la siguiente: “array [i] [j]=25”.

El acceso ahora en lugar de por columnas será por filas, pasando a la siguiente una vez terminada la fila anterior. Entonces ahora trabajamos con posiciones de memoria contiguas. Cuando la caché se llene, se sobrescribirán datos que no se volverán a usar.

Luego el código final será:

```
#include <stdio.h>
#include <stdlib.h>

#define SIZE 4096
int array[SIZE][SIZE];

int main(int argc, char *argv[])
{
        int i;
        int j;
        for(i = 0; i < SIZE; i++)
```

```
        {
                for(j = 0; j < SIZE; j++)
                {
                        array[i][j] = 25;
                }
        }
}
```

Entonces estos cambios producen las siguientes variaciones en los fallos de la caché en las configuraciones [1], [2], [3]:

Configuración 1:

El número de accesos a memoria debido a lectura es: 2096 (por parte de la caché 1)+1162 (por parte de la caché 2)=3258

El número de accesos a memoria debido a escritura es: 2098069(caché 1)+1049057(caché 2)=3147126

Fallos en la caché de nivel 1: 2100165
Fallos en la caché de nivel 2: 1050219
http://gyazo.com/8629592120542aee655a71be3031c6d0

Configuración 2:

El numero de accesos a memoria debidos a la lectura es de 1310(parte de la caché 1)+731 (por parte de la caché 2)=2041

El número de accesos a memoria debidos a escritura, es 1049086(por parte de la caché 1)+524550 (caché 2) = 1573636

Fallos en la caché de nivel 1: 1050396
Fallos en la caché de nivel 2: 525281
http://gyazo.com/5bcd77820494b3ed8640638fcfc999f2

Configuración 3:

El numero de accesos a memoria debidos a la lectura es de 1310(parte de la caché 1)+1162 (por parte de la caché 2)=2472

El número de accesos a memoria debidos a escritura, es 1049086(por parte de la caché 1)+1049056(caché 2) = 2098142

Fallos en la caché de nivel 1: 1050396
Fallos en la caché de nivel 2: 1050218
http://gyazo.com/d9c2476c11c8d759c40c6a92573f6f7d

Notar que no cambian los fallos de lectura, sino los de escritura, viéndose notablemente reducidos. Esto es debido a que la línea que modificamos es

exclusivamente de escritura. En general, hemos pasado de tener una tasa de fallos del 19,9% a otra de un orden del 2,5%.

Apartado 4)

Tomando como configuración inicial la número [3], realizamos los cálculos necesarios de forma manual para obtener la tasa de aciertos y fallos que se producen al agregar un nuevo nivel a la caché con la siguiente configuración:

L3 de 4MB con un tamaño de línea de 64B y asociativa por conjuntos de 16 vías.

Usaremos el código .c corregido por nosotros en el anterior apartado, para reducir la tasa de fallos.

```
#include <stdio.h>
#include <stdlib.h>

#define SIZE 4096
int array[SIZE][SIZE];

int main(int argc, char *argv[])
{
        int i;
        int j;
        for(i = 0; i < SIZE; i++)
        {
                for(j = 0; j < SIZE; j++)
                {
                        array[i][j] = 25;
                }
        }
```

}

Configuración inicial 3:

El numero de accesos a memoria debidos a la lectura es de 1310(parte de la caché 1)+1162 (por parte de la caché 2)=2472

El número de accesos a memoria debidos a escritura, es 1049086(por parte de la caché 1)+1049056(caché 2) = 2098142

Fallos en la caché de nivel 1: 1050396
Fallos en la caché de nivel 2: 1050218

A esto hay que añadirle la caché L3 especificada.

Tamaño de cache=4MB
Tamaño de línea=64 B
Número de líneas= 4MB/64B= 65536 líneas
Traigo 16 líneas de la Memoria cada vez que accedo.

A grandes rasgos, el código en ensamblador de este .c es:

```
.glblmanin

li $t0, 0
li $t2, 4096
la $t2, array                    #direccion de inicio
buclex:
beq $t0, $t2, fin
li $t1, 0

bucley:
beq $t1, $t2, actualiza

        mul $t6, $t0, 4096 #multiplica x por el contador de x
        mul $t6, $t6, 4      #multiplica por 4 el valor de la anterior multiplicación
        mul $t7, $t1, 4          #multiplica el valor del contador y por 4
        add $t6, $t6, $t7
        add $t6, $t2, $t6
        sw 25, ($t6)             #guarda el number en el array
addi $t1, 1
b bucley

actualiza:
addi $t0, 1
b buclex

fin:
li $v0, 10
syscall
```

Hasta “b bucley” nos traemos 16 lineas, tras el primer fallo. No necesito traer otras 16 hasta que no completo el “bucley”. Una vez hecho, traigo hasta b buclex. Luego hasta el fin. Debido al gran tamaño de la caché, la tasa de fallo será muy pequeña.

Observemos:
1 fallo al empezar
4096 aciertos del primer bucle
1 fallo de continuar
4095x4096 aciertos
1 fallo para terminar.

3 fallos, sobre 150962176 aciertos, nos deja una tasa de fallos=1,98x10^-6%

Una tasa de fallos muy reducida, debido a un tamaño de caché grande, un tamaño de línea grande y una asociatividad por conjunto de 16 vías, también elevada

Conclusión:

En comparación con otras, esta ha sido una práctica sencilla. Lo hubiera sido todavía más si se hubieran conocido los parámetros de Linux y valgrind que estábamos utilizando. Muchas veces nos perdíamos y buscábamos soluciones en internet, como por ejemplo el número y tamaño de las memorias cachés de nuestro procesador.

Luego la dificultad de los ejercicios 1 y 2 estaba en saber leer e interpretar los datos que aparecían en la consola. No tardamos mucho en saber en que fijarnos.

En cuanto al ejercicio 3, ya habíamos visto previamente un ejercicio similar, e identificamos rápidamente que el problema se encontraba en la escritura del 25 en las posiciones de la matriz, que se hacía por columnas y no por filas, como debe ser.

Desafortunadamente el tiempo no dio para más, y no pudimos completar el apartado 4 del ejercicio 2 con más detalles o mejores conclusiones.

En general esta práctica no resultó difícil, ya que toda dificultad se encontraba en saber interpretar y trabajar a contra-reloj. Consideramos bastante bueno nuestro trabajo, a pesar de no haber podido completarlo.