Laboratorio Sesión 09: Investigando la Jerarquía de Memoria

Objetivo de la Sesión

El objetivo de la sesión es desarrollar una serie de herramientas que nos permitirán averiguar alguno de los parámetros fundamentales de la memoria cache de un computador.

Introducción

La memoria cache es una memoria de alta velocidad y pequeño tamaño que pretende almacenar, de forma transparente al programador, aquella información que será referenciada en un futuro próximo. Las memorias cache son efectivas porque se basan en las propiedades de localidad, tanto de datos como de instrucciones, de los programas.

Algunos de los parámetros fundamentales que caracterizan una memoria cache son:

- Tamaño de la cache.
- Tamaño de la línea de cache.
- Grado de asociatividad de la cache.

En esta sesión vamos a desarrollar unos programas que nos permitirán averiguar los valores de estos tres parámetros en un computador cualquiera. Nota IMPORTANTE: la mayoría de los procesadores actuales disponen de una cache de datos y otra de instrucciones de primer nivel dentro del procesador. *Nosotros estudiaremos sólo la cache de datos*.

Metodología de trabajo

La herramienta ideal para desarrollar este tipo de aplicaciones son los contadores hardware. Los contadores hardware son circuitos internos del microprocesador cuyo objetivo es permitir que el usuario tenga conocimiento de ciertos parámetros relevantes en la ejecución de programas. Los contadores hardware, una vez programados, se incrementan cada vez que se produce un determinado evento: acceso a memoria, fallos cache datos L1, etc. Estos contadores nos permiten averiguar cuántos eventos de un determinado tipo se producen en una porción de código. En el siguiente código se muestra cómo se obtiene esta información.:

```
...
InitCounters(#Referencias, #FallosL1Datos); // Inicializar contadores
ReadCounters(&referencias1, &fallos1);      // Leer contadores

// CÓDIGO A EVALUAR

ReadCounters(&referencias2, &fallos2);     // Leer contadores
referencias = referencias2-referencias1;   // Calcular referencias totales
fallos = fallos2 - fallos1;                // Calcular fallos totales
m = fallos / referencias;
```

Otra forma de calcular estos parámetros es utilizar medidas de tiempo, según muestra el siguiente código:

```
...
t1 = GetTime();     // Leer reloj antes de empezar

// CÓDIGO A EVALUAR

t2 = GetTime();     // Leer reloj al acabar
tiempo = t2 - t1;   // La diferencia de tiempos es el tiempo total.
```

Dado que los fallos de cache son más costosos (eni tiempo) que un acierto, midiendo los tiempos de ejecución de distintos programas pueden deducirse el valor de algunos parámetros de la cache.

Sin embargo, cuando se utilizan medidas de tiempo, los resultados obtenidos suelen ser bastante imprecisos porque muchos otros eventos, y no sólo los fallos de cache, afectan al tiempo de ejecución.

En nuestro caso no vamos a utilizar ninguna de estas posibilidades. Vamos a utilizar un simulador, que para un programa determinado nos dirá cuántos fallos de cache se producen. Un ejemplo de cómo usar el simulador podría ser el siguiente código:

```
// 1) CODIGO A EVALUAR:   // 2) CODIGO EVALUADOR:

sum = 0;                  InitCache(137);          // Inicializar Cache #137
for(i=0; i<1000; i++)     for(i=0; i<1000; i++)
  sum = sum + v[i]          Referencia(&v[i]);     // Evaluar acceso v[i]

                          misses = Fallos();      // Obtener los fallos de cache


// 2) CODIGO A EVALUAR:   // 2) CODIGO EVALUADOR:

for(i=0; i<1000; i++)     InitCache(71);          // Inicializar Cache #71
  v[i] = v[i] + 44        for(i=0; i<1000; i++){
                            Referencia(&v[i]);     // Evaluar acceso lect v[i]
                            Referencia(&v[i]);     // Evaluar acceso escr v[i]
                          }
                          misses = Fallos();      // Obtener los fallos de cache
```

En este código, el número que se pasa como parámetro a la función InitCache indica el tipo de cache que se usará para hacer las mediciones (hay 200 caches diferentes programadas con distintos tamaños de cache, tamaños de línea, asociatividad, políticas de escritura, etc., por lo que los resultados del experimento dependerán del parámetro que se pase a esta función).

Trabajo previo de la sesión

Antes de explicar en qué consiste la sesión, plantearemos el trabajo previo. Todos los puntos del trabajo previo están relacionados con la práctica a realizar.

1) Calculad cuántos fallos de cache provoca el acceso "v[i]" en los siguientes casos:

Código	Memoria Cache	stepA	stepB	stepC	stepD

`for (j=0, i=0; j<10000; j++) {`
`  sum = sum + v[i];`
`  i = i + step;`
`}`
	Cache Directa
	Tamaño: 4KB
	Tamaño línea: 16B
		step = 1	step = 2	step = 4	step = 8

`for (j=0, i=0; j<10000; j++) {`
`  sum = sum + v[i];`
`  i = i + step;`
`}`
	Cache 2-asociativa
	Tamaño: 4KB
	Tamaño línea: 32B
		step = 1	step = 2	step = 4	step = 8

Suponed que cada elemento de v[i] ocupa 4 bytes.

2) Dado el siguiente código:

```
for (j=0, i=0; j<10000; j++) {
  sum = sum + v[i];  // cada elemento de v[i] ocupa 4 bytes
  i = i + step;
}
```

Suponiendo que la cache es directa (16KB) con líneas de 32B, dibujad una gráfica donde se represente el número de fallos que se producen (eje y) variando la variable step de 1 a 16 (eje x).

3) Calculad cuántos fallos de cache provoca el acceso "v[i]" en los siguientes casos:

Código	Memoria Cache	valores de limit

`for (i=0,j=0; j<64; j++){`
`  sum = sum+ v[i];`
`  i = i + 4;`
`  if (i >= limit) i = 0;`
`}`
	Cache Directa
	Tamaño: 4 líneas
	Tamaño línea: 16B
		limit = 32B, 64B, 80B, 96B, 128B y 256B
		o lo que es lo mismo
		limit = 8 int, 16, int, 20 int, 24 int, 32 int y 64 int

`for (i=0,j=0; j<64; j++){`
`  sum = sum+ v[i];`
`  i = i + 4;`
`  if (i >= limit) i = 0;`
`}`
	Cache 2-asociativa
	Tamaño: 4 líneas
	Tamaño línea: 16B
		limit = 32B, 64B, 80B, 96B, 128B y 256B

`for (i=0,j=0; j<64; j++){`
`  sum = sum+ v[i];`
`  i = i + 4;`
`  if (i >= limit) i = 0;`
`}`
	Cache 4-asociativa
	Tamaño: 4 líneas
	Tamaño línea: 16B
		limit = 32B, 64B, 80B, 96B, 128B y 256B

Suponed que cada elemento de v[i] es un int y ocupa 4 bytes.

4) Dado el siguiente código:

```
for (i=0,j=0; j<M; j++) {    // M vale 1024*1000
  sum = sum+ v[i];           // cada elemento de v[i] ocupa 4 bytes
  i = i + 8;
  if (i >= limit) i = 0;
}
```

Suponiendo que la cache es directa (4KB) con líneas de 32B, dibujad una gráfica con los fallos que se producen (eje y) variando la variable limit de 512 a 4096 con incrementos de 256 (eje x).

5) Dado el siguiente código

```
for (i=0,j=0; j<M; j++) {    // M vale 1024*1000
  sum = sum+ v[i];           // cada elemento de v[i] ocupa 4 bytes
  i = i + 8;
  if (i >= limit) i = 0;
}
```

Suponiendo que la cache es 2-asociativa (4KB) con líneas de 32B, dibujad una gráfica con los fallos que se producen (eje y) variando la variable limit de 512 a 4096 con incrementos de 256 (eje x). En este apartado, como en el anterior, no es necesario calcular todos los puntos. Sólo es necesario calcular algunos puntos importantes (revisad el apartado 3).

6) Las diferencias entre las gráficas de los dos anteriores apartados son debidas a la asociatividad. Comentad y explicad el por qué de esa diferencia.

Tamaño de la línea de la cache de datos

El primer parámetro que vamos a averiguar es el tamaño de la línea de cache. La línea de cache es la unidad de transferencia entre la memoria cache y la memoria principal (o el siguiente nivel de la jerarquía de memoria). Recordad que, cada vez que se produce un fallo de cache, se trae del nivel superior de la jerarquía la línea de memoria que contiene el dato solicitado.

Ejercicio teórico/práctico

Si revisamos los 2 primeros apartados del trabajo previo, el código era similar a éste:

```
char v[N];

i = 0;
for (j = 0; j < M; j++) {
  sum = sum + v[i];
  i = i + step;
  if (i >= N) i = 0;
}
```

Había que calcular el número de fallos en la cache de datos de este programa, teniendo en cuenta lo siguiente:

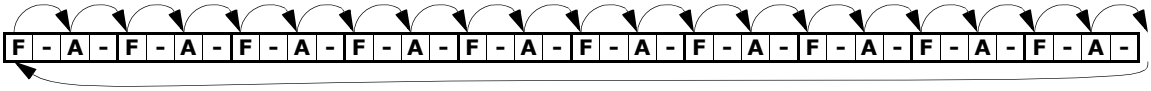
- El vector `v` es varias veces más grande que la cache de datos.
- M es un valor grande (p.e. 1000*1024)
- El tamaño de línea es 4 bytes.
- Sólo se contabilizan los accesos en lectura a `v[i]`.
- Hay que calcular los fallos teniendo en cuenta diferentes valores de la variable `step`.

Estudiando el código, podemos llegar a las siguientes conclusiones:

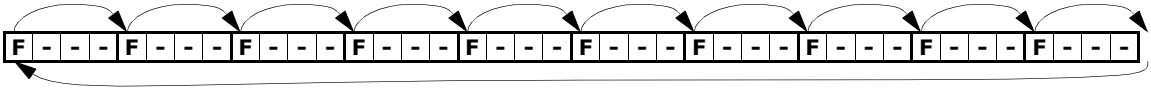
- Se hace un número constante de accesos a memoria: M.
- Como el vector v es mucho más grande que la memoria cache, no tenemos localidad temporal. Por ejemplo, con step=1, si empezamos con i=0 y recorremos el vector hasta N-1, cuando volvamos a i=0 volveremos a fallar porque no quedará nada de los primeros elementos del vector en la cache.
- Sólo tenemos localidad espacial, que además depende de la variable step.

El comportamiento del algoritmo se muetra de forma gráfica en la siguiente figura:

- step=1, 1 fallo y 3 aciertos por línea, M/4 fallos en total.



- step=2, 1 fallo y 1 acierto por línea, M/2 fallos en total.



- step=4, 1 fallo por línea, M fallos en total.

En definitiva, podemos concluir que:

- si L es el tamaño de línea, entonces P = L/4 es el número de elementos por línea,
- entonces min(step, P) · M / P será el número de fallos totales.

Notad que, cuando el step sea mayor o igual que P, todos los accesos serán fallos.

Ejercicio Experimental

Utilizando como base el código del ejercicio teórico anterior, es fácil escribir un programa en C que permita averiguar el tamaño de línea de nuestro computador.

Ese código podría ser el siguiente:

```
// CODIGO A EVALUAR:

for (step=1; step<=16; step++) {
   i = 0;
   sum = 0;
   for (j=0; j<M; j++) {
      sum = sum + v[i];
      i = i + step;
      if (i >= tam) i = 0;
   }
}
```

```
// CODIGO EVALUADOR:

for (step=1; step<=16; step++) {
   InitCache(COD);
   i = 0;
   sum = 0;
   for (j=0; j<M; j++) {
      Referencia(&v[i]);
      i = i + step;
      if (i >= tam) i = 0;
   }
   misses = Fallos();
}
```

Este programa lo podéis encontrar en el fichero "LineSize.c". La variable "COD" de la función InitCache identifica el tipo de cache que usaréis para hacer la práctica. El valor de la variable os lo diremos durante la clase de laboratorio.

Se pide que hagáis lo siguiente:

- Compilad y Ejecutad el programa. Para compilar, debéis utilizar el siguiente comando:

  ```
  $> gcc LineSize.c Simulador.o –o LINE
  ```

  En "Simulador.o" están las rutinas que simulan el comportamiento de la cache.
- Deducid el tamaño de línea de la cache de datos de primer nivel que estáis simulando . Es posible que sea necesario modificar el valor máximo que puede alcanzar la variable "step".

Tamaño de la memoria cache de datos

Un parámetro fundamental de la memoria cache es su tamaño. Vamos a diseñar un programa escrito en C que nos permita averiguar el tamaño de la memoria cache de datos de primer nivel de nuestro computador.

Ejercicio teórico / práctico

Igual que antes, si revisamos el apartado 3 del trabajo previo, teníamos un código similar a éste:

```
char v[N];

i = 0;
step = 4;
for (j = 0; j < M; j++) {
   sum = sum + v[i];
   i = i + step;
   if (i >= limit) i = 0;
}
```

Hay que calcular el número de fallos en la cache de datos de este programa, teniendo en cuenta lo siguiente:

- El vector `v` es varias veces más grande que la cache de datos.
- M es un valor muy grande (p.e. 10*1000*1024).
- El tamaño de línea es 4 bytes.
- Sólo se contabilizarán los accesos en lectura a `v[i]`.
- Hay que calcular los fallos teniendo en cuenta diferentes valores de la variable `limit`.

Estudiando el código, podemos llegar a las siguientes concluisones:

- Se hace un número constante de accesos a memoria: M.
- El programa se comportará de forma muy diferente en función del valor de `limit`:
  - Si `limit` es menor o igual que el tamaño de la cache, estaremos accediendo repetidamente a un vector que cabe en la cache. Todos los accesos serán aciertos, a excepción de los fallos iniciales de carga.
  - Si `limit` es un poco más grande que la cache, tendremos algunos fallos de capacidad.
  - A partir de un cierto valor de `limit` (dependiendo de la asociatividad), todos los accesos serán fallos.

Ejercicio Experimental

Utilizando como base el código del ejercicio teórico anterior, es fácil escribir un programa en C que permita averiguar el tamaño de la cache de datos de primer nivel.

Ese código podría ser el siguiente:

```
// CODIGO A EVALUAR:

for (size=512; size<=8*1024; size+=256) {
   LimpiarCache();
   i = 0;
   for (j=0; j<M; j++) {
     sum = sum + v[i];
     i=i+STEP;
     if (i >= size) i=0;
   }
}
```

```
// CODIGO EVALUADOR:

for (size=512; size<=8*1024; size+=256) {
   InitCache(COD);
   i = 0;
   for (j=0; j<M; j++) {
      Referencia(&v[i]);
      i=i+STEP;
      if (i >= size) i=0;
   }
   misses = Fallos();
}
```

Este programa lo podéis encontrar en el fichero "CacheSize.c".

Se pide que hagáis lo siguiente:

- Compilad y Ejecutad el programa. Para compilar debéis utilizar el siguiente comando:

  ```
  $> gcc CacheSize.c Simulador.o –o SIZE
  ```

- Deducid el tamaño de la cache de datos de primer nivel del computador que estáis usando. Para ello, es posible que sea necesario modificar el bucle

  ```
  for (size=512; size <= 8*1024; size+=256)
  ```

  Modificad éstos valores teniendo en cuenta los posibles tamaños que puede tomar la memoria cache.
- Utilizad el step obtenido en el apartado anterior.

Trabajo de la sesión

Durante la sesión, el profesor de laboratorio os dará una hoja que deberéis rellenar. En esa hoja estará indicado el código de dos caches (será el parámetro que deberéis pasarle a la rutina InitCache). Para cada una de las caches deberéis averiguar:

- Tamaño de línea (rango posible de valores: entre 16 y 128 bytes).
- Tamaño de cache (rango posible de valores: entre 1 y 2048 Kbytes).
- Asociatividad (rango posible de valores: entre directa y 8-asociativa).

Nota: Los ficheros para esta sesión los podéis encontrar en la página web de la asignatura: "`Programas.Sesion09.tar.gz`".

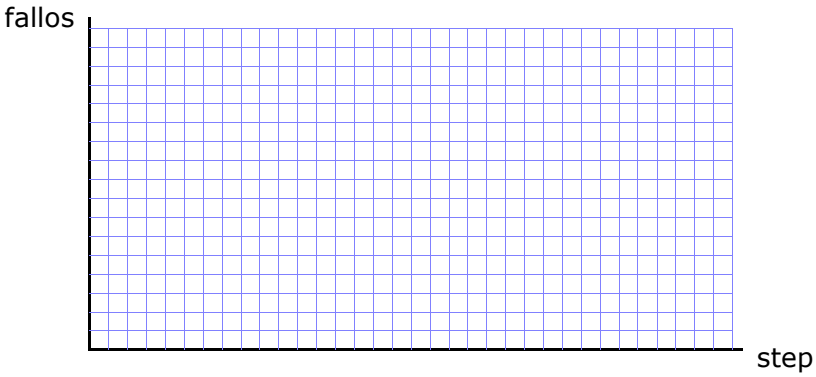
En la siguiente página se muestra el modelo de cuestionario que os entregará el profesor de laboratorio y que deberéis rellenar. Tendréis que repetir el cuestionario para 2 caches diferentes.

Nombre 1:	Fecha:
Nombre 2:	Grupo:

CÓDIGO CACHE:

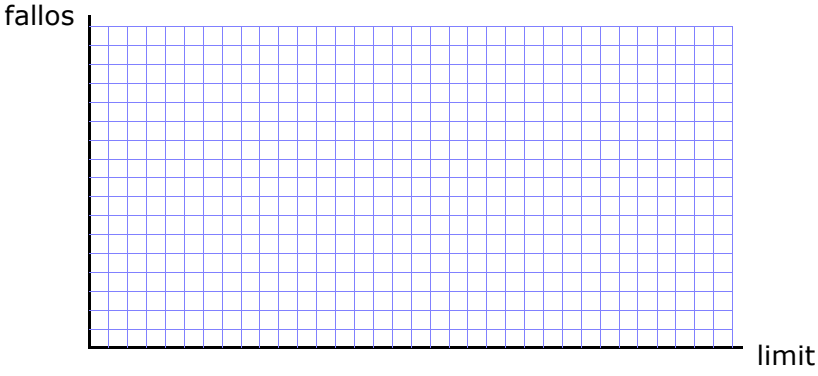
TAMAÑO LÍNEA (Justificad la respuesta):

Rellenad la siguiente tabla donde se represente el número de fallos que se producen (eje y) en función de la variable step (eje x). Esta gráfica es similar a la del apartado 2 del trabajo previo.



TAMAÑO CACHE (Justificad la respuesta):

Rellenad la siguiente tabla donde se represente el número de fallos que se producen (eje y) en función de la variable limit (eje x). Esta gráfica es similar a la de los apartados 4 y 5 del trabajo previo.

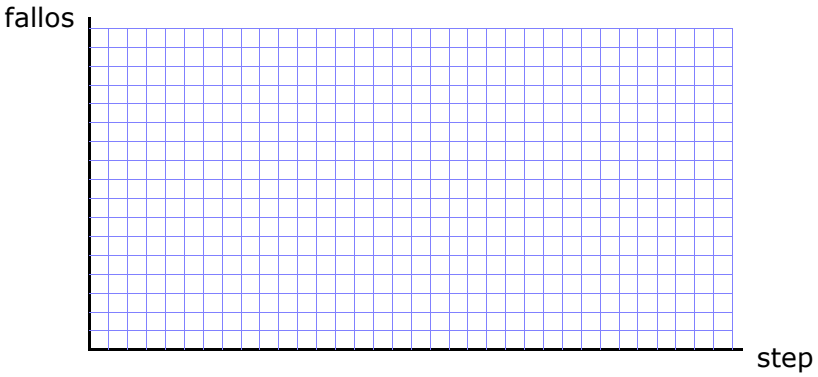


ASOCIATIVIDAD (Justificad la respuesta):

CÓDIGO CACHE:

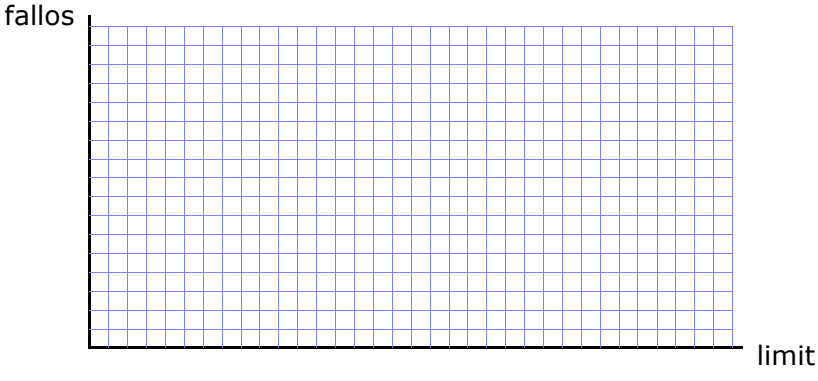
TAMAÑO LÍNEA (Justificad la respuesta):

Rellenad la siguiente tabla donde se represente el número de fallos que se producen (eje y) en función de la variable step (eje x). Esta gráfica es similar a la del apartado 2 del trabajo previo.



TAMAÑO CACHE (Justificad la respuesta):

Rellenad la siguiente tabla donde se represente el número de fallos que se producen (eje y) en función de la variable limit (eje x). Esta gráfica es similar a la de los apartados 4 y 5 del trabajo previo.



ASOCIATIVIDAD (Justificad la respuesta):