Práctica3: Memoria caché y rendimiento

NOTA: Leer detenidamente el enunciado de cada ejercicio antes de proceder a su realización.

El objetivo de esta práctica es experimentar y controlar el efecto de las memorias caché en el rendimiento de un programa de usuario. Para ello, se hará uso del framework de herramientas de código abierto llamada *valgrind* [1], concretamente nos centraremos en el uso de dos apliacaciones *cachegrind* [2] y *callgrind* [3].

Todo el trabajo asociado a esta práctica se realizará en un entorno Linux (en particular, asumiremos que se utiliza la versión de Linux instalado en los laboratorios: Ubuntu 14.04).

NOTA: Si bien puede preparar las prácticas en su propia instalación de Linux, los valores numéricos deben ser los obtenidos en los laboratorios de la escuela.

MATERIAL DE PARTIDA

Junto con el enunciado de la práctica, se entrega un fichero "`.zip`" que contiene los siguientes ficheros que serán referenciados a lo largo del enunciado:

- `arqo3.c` – fichero con el código fuente de la librería de manejo de matrices de números en coma flotante.
- `arqo3.h` – fichero de cabeceras para la librería de manejo de matrices.
- `slow.c` – fichero con un programa de ejemplo que calcula la suma de los elementos de una matriz.
- `fast.c` – fichero con un programa de ejemplo que calcula la suma de los elementos de una matriz (más eficiente que el anterior).
- `Makefile` – fichero utilizado para la compilación de los ejemplos aportados.

IMPORTANTE

A lo largo de este documento se hará referencia a un número P que afectará a los experimentos y resultados que se obtendrán. El valor de P que debe utilizar debe ser igual al número de pareja dentro de su grupo de prácticas (es decir, para los alumnos de la pareja 8 del grupo 1362, P=8; para la pareja 3 del grupo 1301, P=3; …).

A lo largo de la práctica, se pide generar varias gráficas. Todas las gráficas generadas deberán tener claramente indicado qué se representa en cada eje, y se deberá marcar de forma legible la escala de los mismos. También se deberá incluir una leyenda que permita diferenciar las diferentes curvas representadas sobre una misma gráfica.

Ejercicio 0: información sobre la caché del sistema (1 p)

En primer lugar, deberíamos de ser capaces de obtener la información relativa a la memoria caché de la máquina sobre la que estamos trabajando. Esto es sencillo en un sistema Linux y podemos hacerlo ejecutando cualquiera de los siguientes comandos:

```
> cat /proc/cpuinfo
```

```
> dmidecode
```

NOTA: este comando requiere permisos de superusuario, por lo que no podremos utilizarlo en la instalación de los laboratorios.

```
> getconf -a | grep -i cache
```

A partir de la información expuesta, indique en la memoria asociada a la práctica los datos relativos a las memorias caché presentes en los equipos del laboratorio. Se ha de resaltar en qué niveles de cache hay separación entre las cachés de datos e instrucciones, así como identificarlas con alguno de los tipos de memoria caché vistos en la teoría de la asignatura.

Ejercicio 1: Memoria caché y rendimiento (3 p)

En este ejercicio, vamos a comprobar de manera empírica como el patrón de acceso a los datos puede mejorar el aprovechamiento de las memorias caché del sistema. Para ello, se utilizarán los dos programas de prueba que se aportan con el material de la práctica.

Un ejemplo de salida de uno de estos programas es:

```
>./slow 100
Word size: 32 bits
Execution time: 0.000057
Total: 49713.222100
```

Estos programas indican, en primer lugar, el tamaño de dato que utilizan (que se corresponde con el tamaño de palabra del equipo), dato que será importante tener en cuenta a la hora de traducir las matrices utilizadas a tamaño de memoria ocupado.

En segundo lugar, imprimen el tiempo requerido para hacer la suma de todos los elementos de una matriz cuadrada cuyo tamaño ha sido pasado como argumento al programa (en este caso, la matriz es de 100x100 elementos de 4 bytes = 40.000 bytes).

En último lugar, los programas imprimen el resultado de la suma. Puesto que la inicialización de las matrices se realiza utilizando una semilla fija para la inicialización con números pseudo-aleatorios, el resultado debería ser el mismo si el tamaño se mantiene fijo (nótese que distintas versiones del programa pueden obtener resultados diferentes debido a los redondeos llevados a cabo al cambiar el orden de las operaciones).

Para la realización de este ejercicio se pide:

1) Tomar datos de tiempo de ejecución de los dos programas de ejemplo que se proveen (“fast” y “slow”) para matrices de tamaño NxN, con N variando entre P y 2048+P y con un incremento de 16.

2) Guardar para la entrega el fichero con los resultados obtenidos. Por criterios de unificación, se pide que el fichero siga el formato:

   `<tamaño de la matriz> <tiempo “slow”> <tiempo “fast”>`

3) Pintar una gráfica que muestre los resultados obtenidos, e incluirla en la memoria.

4) Además de la gráfica mencionada, se pide en la memoria explicar el método seguido para la obtención de los datos, y una justificación del efecto observado. ¿Por qué para matrices pequeñas los tiempos de ejecución de ambas versiones son similares, pero se separan según aumenta el tamaño de la matriz? ¿Cómo se guarda la matriz en memoria, por filas (los elementos de una filas están consecutivos) o bien por columnas (los elementos de una columna son consecutivos)? ¿qué efecto tiene el tamaño de bloque de la cache?

NOTA: para contestar a esta pregunta se debe entender el funcionamiento del código fuente de los programas que se proporcionan.

Ejercicio 2: Tamaño de la caché y rendimiento (3 p)

El framework de herramientas *valgrind* nos permite ejecutar programas en un entorno cuyas características nos deja modificar en ciertos aspectos. Concretamente, utilizando las herramientas *cachegrind* ó *callgrind* (como una extensión de la anterior) podemos obtener información sobre el comportamiento de los accesos a la caché de nuestro programa, así como modificar las características la jerarquía de las cachés.

Para utilizar cualquiera de las herramientas que nos ofrece *valgrind* sólo tenemos que ejecutar:

```
>valgrind --tool=<herramienta> [opciones _herr] <ejecutable>
      [args_ejecutable]
```

En nuestro caso, utilizaremos como herramienta ó bien *cachegrind* ó bien *callgrind*. Estas herramientas simulan la ejecución del programa deseado sobre un sistema con unos parámetros determinados. Ambas herramientas nos ofrecen una serie de argumentos que nos permiten configurar hasta dos niveles de cache en el sistema sobre el que se simula la ejecución de nuestro programa:

```
--I1=<size>,<associativity>,<line size>
Specify the size, associativity and line size of the level 1
instruction cache.

--D1=<size>,<associativity>,<line size>
Specify the size, associativity and line size of the level 1
data cache.

--LL=<size>,<associativity>,<line size>
Specify the size, associativity and line size of the last-level
cache.
```

Nótese que los tamaños de la caché y de la línea son tamaños en bytes, y que el parámetro *associativity* hace referencia al número de vías que tendrá cada una de las memorias caché.

La salida de estas herramientas (además de la salida por pantalla que muestre el ejecutable objetivo así como la propia salida de la por pantalla de la herramienta) es una archivo cuyo nombre sigue el formato: *cachegrind.out.<pid>* ó bien *callgrind.out.<pid>* . Estos ficheros contienen los resultados de la simulación, en particular contienen contadores y estadísticas del uso de las cachés del sistema. Para acceder a estos datos de una forma cómoda, se dispone de la herramienta *cg_annotate* (si la salida se generó con *cachegrind*) y *callgrind_annotate* (si la salida se generó con *callgrind*).

Por ejemplo, si queremos obtener la información relativa al uso de caché cuando ejecutamos el comando ls, tenemos que:

```
> valgrind --tool=cachegrind ls
      …
>cg_annotate cachegrind.out.8170 | head -n 30
--------------------------------------------------------------------------------
I1 cache:         32768 B, 64 B, 8-way associative
D1 cache:         32768 B, 64 B, 8-way associative
LL cache:         8388608 B, 64 B, 16-way associative
Command:          ls
Data file:        cachegrind.out.8170
```

```
Events recorded:  Ir I1mr ILmr Dr D1mr DLmr Dw D1mw DLmw
Events shown:     Ir I1mr ILmr Dr D1mr DLmr Dw D1mw DLmw
Event sort order: D1mr
Thresholds:       0.1
Include dirs:
User annotated:
Auto-annotation:  off

--------------------------------------------------------------------------------
     Ir  I1mr  ILmr      Dr  D1mr  DLmr     Dw  D1mw  DLmw
--------------------------------------------------------------------------------
548,211 1,696 1,569 142,123 4,342 2,819 59,621 1,152 1,042  PROGRAM TOTALS
```

En este caso podemos observar la configuración de cachés que el programa coge por defecto (en una instalación distinta de la de los laboratorios): una caché de datos e instrucciones de primer nivel en ambos casos de 32KB de tamaño y un tamaño de línea de 64B con 8 vías, así como una caché de nivel superior de 8MB, tamaño de línea 64B y 16 vías. Esta configuración por defecto coincide con la del sistema sobre el que se ejecute *valgrind*. También podemos observar en nuestro caso se han producido 548.211 lecturas de la instrucciones (columna *Ir*), de las cuales 1.696 han producido fallos en la caché de primer nivel (columna *I1r*) y 1.569 han producido fallos en la caché de nivel superior (columna *ILmr*). Datos similares se obtienen para las lecturas y escrituras de datos (columnas *Dr?* y *Dw?* Respectivamente).

Para este ejercicio, se pide:

1) Tomar datos de la cantidad de fallos de caché producidos en la lectura de datos al ejecutar las dos versiones (*fast* y *slow*) del programa que calcula las suma de los elementos de una matriz para matrices de tamaño NxN, con N variando entre P y 2048+P. Utilizar tamaños caché de primer nivel (tanto para datos como para instrucciones) de tamaño variable: P, ,2*P, 4*P y 8*P Kbytes, y una caché de nivel superior de tamaño fijo igual a 8 Mbytes. Para todas las cachés asumir que son de correspondencia directa (es decir, con una única vía) y con tamaño de línea igual a 64 bytes.

2) Guardar para la entrega el fichero con los resultados obtenidos. Por criterios de unificación, se pide que el fichero se pide que los datos se almacenen en ficheros llamados `<tamCache>.dat` (esto es, se generará un fichero para cada tamaño de caché) siguiendo el formato:

   `<tam matriz> <D1mr "slow"> <D1mw "slow"> <D1mr "fast"> <D1mw "fast">`

3) Se han de representar sobre dos gráficas (una para los fallos de lectura de datos y otra para los fallos en escritura de datos) el comportamiento de las 5 medidas realizadas para cada una de las dos versiones e incluirla en la memoria.

4) Justifique el efecto observado. ¿Se observan cambios de tendencia al variar los tamaños de caché al cambiar a la versión *fast* o *slow* del programa? ¿A qué se debe este efecto?

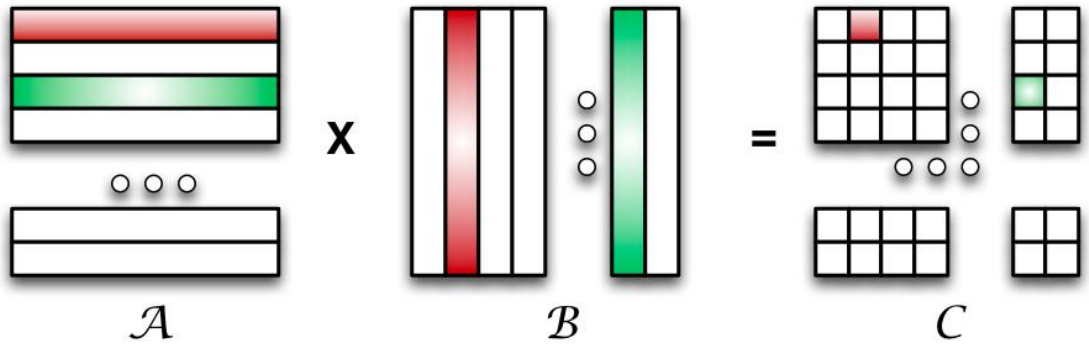
Ejercicio 3: Caché y multiplicación de matrices (3 p)

Se pide que los alumnos confeccionen un programa que lleve a cabo la multiplicación de dos matrices cuadradas del mismo tamaño. Este programa recibirá como único argumento el número N (=nº de filas = nº de columnas de las matrices).

Para simplificar la tarea, se han de usar las funciones de las librería `arqo3` que se entrega como material de la práctica (la función `generateMatrix` se usará para generar cada una de las dos matrices a multiplicar, y la función `generateEmptyMatrix` se usará para generar la matriz que más tarde contendrá el resultado de la multiplicación. El programa deberá imprimir por pantalla el tiempo necesario para llevar a cabo la multiplicación de las matrices (sin tener en cuenta en tiempo necesario para generar y liberar las matrices utilizadas)

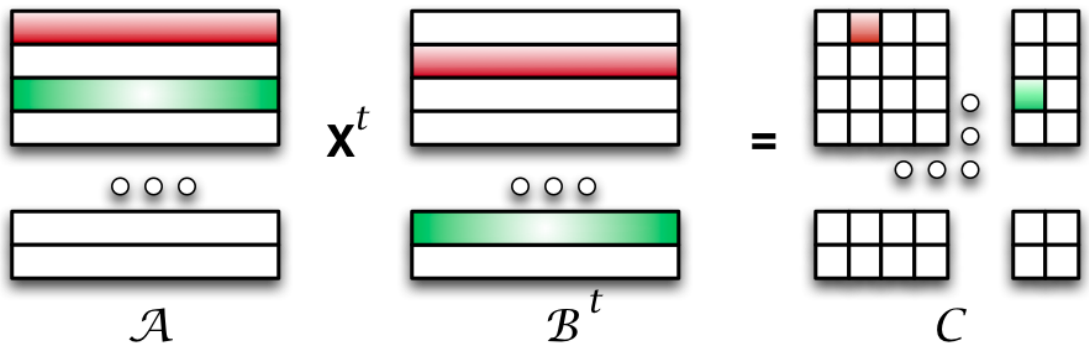
Recordatorio: al multiplicar dos matrices A y B, obtenemos una matriz C con elementos son:

$$c_{ij} = \sum_{k=1}^{N} a_{ik} \times b_{kj}$$



Nótese que en una multiplicación de matrices, la matriz A es accedida por filas, mientras que la matriz B es accedida por columnas.

Una vez desarrollado el programa que realiza la multiplicación de matrices, se pide crear un nuevo programa a partir del anterior que realice una "multiplicación traspuesta":



Nótese que en este caso el resultado debe ser equivalente al programa anterior, pero cambia el modo de almacenamiento de la matriz B, de la que pasamos a almacenar y utilizar su traspuesta. En este caso:

$$c_{ij} = \sum_{k=1}^{N} a_{ik} \times b^{t}_{jk}$$

Como puede apreciarse, cambia el patrón de accesos a los datos de la segunda matriz.

La versión del programa que realiza la "multiplicación traspuesta" debe tener en cuenta en el tiempo de ejecución que imprima por pantalla el tiempo requerido para realizar la trasposición de la matriz B (no es necesario contabilizar el tiempo de reserva de dicha matriz).

Una vez codificadas ambas versiones de la multiplicación de matrices, se pide:

1) Tomar datos de la cantidad de los tiempos de ejecución y fallos de caché producidos en la lectura y escritura de datos al ejecutar las dos versiones del programa que calcula la multiplicación de dos matrices de tamaño NxN, con N variando entre P y 512+P.

2) Guardar para la entrega el fichero con los resultados obtenidos. Por criterios de unificación, se pide que el fichero siga el formato:

```
<tam matriz> <tiempo "normal"> <D1mr "normal"> <D1mw "normal"> <tiempo "trasp"> <D1mr "trasp"> <D1mw "trasp">
```

3) Se han de representar sobre dos gráficas (una para los fallos de caché y otra para el tiempo de ejecución) el comportamiento de las 5 medidas realizadas para cada una de las dos versiones e incluirla en la memoria.

4) Justifique el efecto observado. ¿Se observan cambios de tendencia al variar los tamaños de las matrices? ¿A qué se deben los efectos observados?

MATERIAL A ENTREGAR

Memoria con las respuestas a las preguntas a lo largo del enunciado.
En subdirectorios independientes, los ficheros con los datos obtenidos en los experimentos de los ejercicios que así lo indiquen, y los scripts utilizados para la obtención de estos datos.
Códigos fuente desarrollados para la resolución del ejercicio 3.

La entrega se realizará a través de moodle, con fecha tope el día antes del comienzo de la siguiente práctica hasta las 23:59 de la noche.

MATERIAL DE REFERENCIA

[1] **Valgrind**. http://valgrind.org/

[2] **Cachegrind: a cache and branch-prediction profiler.**
http://valgrind.org/docs/manual/cg-manual.html

[3] **Callgrind: a call-graph generating cache and branch prediction profiler.**
http://valgrind.org/docs/manual/cl-manual.html