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 aplicaciones *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 18.04).

NOTA: Las prácticas se pueden realizar en los equipos del laboratorio, en tu propio equipo o incluso en el en los equipos del clúster de ARQO si está disponible. Se debe indicar el equipo empleado.

Si no dispone del framework de herramientas *valgrind* en su equipo personal, puede instalarlo ejecutando el siguiente comando como superusuario (asumiendo que su distribución de Linux es Ubuntu):

```
> (sudo) apt-get install valgrind
```

Si no dispone de GNUplot en su equipo personal, puede instalarlo ejecutando el siguiente comando como superusuario (asumiendo que su distribución de Linux es Ubuntu):

```
> (sudo) apt-get install gnuplot
```

Si no dispone de lstopo en su equipo personal, puede instalarlo ejecutando el siguiente comando como superusuario (asumiendo que su distribución de Linux es Ubuntu):

```
> (sudo) apt-get install hwloc
```

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 de código fuente de la librería de manejo de matrices.
- `arqo3.h` – fichero de cabeceras de la librería de manejo de matrices.
- `slow.c` – fichero de código fuente del programa de ejemplo que calcula la suma de los elementos de una matriz.
- `fast.c` – fichero de código fuente del 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.
- `slow_fast_time.sh` – fichero que contiene un script Bash de ejemplo para ejecutar los programas slow y fast.

IMPORTANTE: el script `slow_fast_time.sh`, tal como se entrega, **NO** genera los resultados y gráficos tal y como se piden en el enunciado. Si decidís utilizarlo, deberéis modificarlo de acuerdo con lo que se pide en el enunciado. Después de compilar los programas fast y slow, podéis lanzar el script introduciendo el siguiente comando:

```
> source slow_fast_time.sh
```

IMPORTANTE

A lo largo de la práctica se hará referencia a un número P que afectará a los experimentos y resultados que se obtendrán. El valor de P que deberá utilizar, y anotar en la memoria, se calcula como: P = (num_pareja mod 7). Ejemplos:

- pareja PT08: P = (8 mod 7) = 1
- pareja PM13: P = (13 mod 7) = 6

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 éste. También se deberá incluir una leyenda que permita diferenciar las diferentes curvas representadas sobre una misma gráfica y una explicación de los resultados y conclusiones que se extraen de la misma.

RECOMENDACIÓN:

Se recomienda encarecidamente a los alumnos emplear scripts de Bash y de GNUplot similares a los presentados en clase para lanzar las ejecuciones, obtener los resultados y generar los gráficos de resultados. El uso de dichos scripts les ahorrará tiempo ejecutando pruebas, puesto que un script bien diseñado puede lanzar las pruebas de forma desatendida y en el orden que establezca. Así mismo, la generación de scripts de GNUplot le permitirá generar rápidamente los gráficos de resultados de forma muy fácil.

En caso de que los alumnos entreguen los scripts que empleen (y que estos funcionen adecuadamente) junto con la memoria y los ficheros de resultados, se les valorará positivamente en la nota de la práctica.

Si no tiene conocimientos de Bash scripting y desea adquirir nociones básicas, se recomienda que realice los sencillos tutoriales referenciados en [4] y [5]. El tiempo que invierta en ellos le compensará en tiempo ahorrado realizando la práctica.

Del mismo modo, si no tiene conocimientos previos de GNUplot y desea adquirir nociones básicas, se recomienda que realice el sencillo tutorial referenciado en [6].

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:

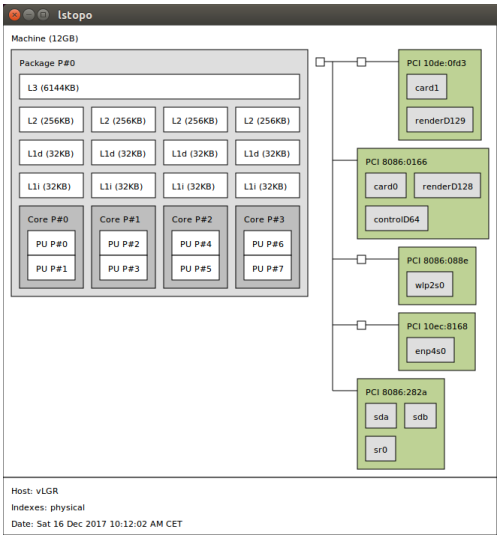
```
> getconf -a | grep -i cache
```

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 caché 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.

También puede obtener información complementaria ejecutando:

```
> lstopo
```
(recuerda de instalar `apt-get install hwloc`)

Analiza la arquitectura de tu equipo en particular.



NOTA: Si se usa lstopo con máquinas virtuales, como puede ser el caso de la cola mv.q del cluster, existe una incoherencia en los tamaños de la caché L2 entre getconf y lstopo. Para responder a las respuestas se deben coger los datos de getconf, aunque se puede poner la figurita de lstopo.

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 1000
Word size: 64 bits
Execution time: 0.007926
Total: 5000066.116831
```

La salida de estos programas nos indica, en primer lugar, el tamaño de dato que utiliza (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 1000x1000 elementos de 8 bytes cada uno = 8.000.000 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).

Nótese, además, que, si ejecuta el programa múltiples veces, el tiempo de ejecución del mismo variará. Es por este motivo que las pruebas de rendimiento deben realizarse múltiples veces y de forma intercalada cómo se indica a continuación:

```
for tamaño_matriz
    for iteración
        slow
        fast

  slow 1024
  fast 1024
  ... (10 veces)
  slow 1024
  fast 1024
  slow 2048
  fast 2048
  ... (10 veces)
  slow 2048
  fast 2048
  ...
```

Existen técnicas estadísticas para determinar la cantidad necesaria de repeticiones de los experimentos, pero, puesto que ese tipo de estudio no es objetivo de esta asignatura, se pide que el alumno determine de forma experimental (realizando múltiples pruebas) qué cantidad de repeticiones es adecuada. El criterio para determinar que los resultados son adecuados será que las gráficas resultantes no presenten “grandes” picos en las medidas.

Para la realización de este ejercicio se pide:

1) Tomar datos de tiempo de ejecución de los dos programas de ejemplo que se proveen (`slow` y `fast`) para matrices de tamaño NxN, con N variando entre 1024 y 16384 con un incremento en saltos de 1024 unidades. Deberá tomar resultados para todas las ejecuciones múltiples veces (se recomiendan al menos 10) y de forma intercalada, como se ha indicado anteriormente, y calcular la media de las mismas.

2) Indique en la memoria una explicación razonada del motivo por el que hay que realizar múltiples veces la toma de medidas de rendimiento para cada programa y tamaño de matriz.

3) Guardar para la entrega el fichero con los resultados obtenidos. Por criterios de unificación, se pide que el fichero se llame `time_slow_fast.dat` y siga el formato:

```
<N> <tiempo "slow"> <tiempo "fast">
```

4) Pinte una gráfica con GNUplot en formato PNG que muestre los resultados obtenidos para los programas `slow` y `fast`, llámela `time_slow_fast.png`, e inclúyala en la memoria.

5) 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 fila están consecutivos) o bien por columnas (los elementos de una columna son consecutivos)?

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 que puede ser configurado 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 de 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 o bien `cachegrind` o 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 caché 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 que muestre el ejecutable objetivo, se volcaran tanto por pantalla, como a un archivo cuyo nombre tiene el formato: `cachegrind.out.<pid>` o bien `callgrind.out.<pid>`, dependiendo de la herramienta utilizada. Se puede modificar el nombre del fichero que contenga el volcado de la salida mediante los argumentos siguientes:

```
--callgrind-out-file=<file>   Output file name [callgrind.out.%p]

--cachegrind-out-file=<file>  Output file name [cachegrind.out.%p]
```

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 ejecutar:

```
> valgrind --tool=cachegrind --cachegrind-out-file=ls_out.dat ls
     ...
> cg_annotate ls_out.dat | 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:        ls_out.dat
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,
- una caché de nivel superior de 8MB, tamaño de línea 64B y 16 vías.

NOTA: Para poder usar los resultados de cg_annotate es necesario eliminar las comas que indican "miles".

Esta configuración por defecto coincide con la del sistema sobre el que se ejecute `valgrind`. También podemos observar que en nuestro caso se han producido 548.211 lecturas de la memoria de instrucciones (columna *`Ir`*), de las cuales 1.696 han producido fallos en la caché de primer nivel (columna *`I1mr`*) y 1.569 han producido fallos de lectura en la caché de nivel superior (columna *`ILmr`*). Datos similares se obtienen para las lecturas y escrituras de datos (columnas *`D*r`* y *`D*w`* respectivamente).

Para este ejercicio, se pide:

1) Tomar datos de la cantidad de fallos de caché producidos en la lectura y escritura de datos al ejecutar las dos versiones (*fast* y *slow*) del programa que calcula las sumas de los elementos de una matriz para matrices de tamaño NxN, con N variando entre 1024+128*P y 5120+128*P y con un incremento en saltos de 1024 unidades. Utilizar tamaños de caché de primer nivel (tanto para datos como para instrucciones) de tamaño variable de 1024, 2048, 4096 y 8192 Bytes (note que deben ser potencias de 2) 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. Como valgrind es un emulador, solo es necesario **ejecutar 1 vez** el programa para cada configuración solicitada.

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

   `<N> <D1mr "slow"> <D1mw "slow"> <D1mr "fast"> <D1mw "fast">`

3) Se han de representar los resultados sobre dos gráficas generadas con GNUplot en formato PNG: una gráfica para los fallos de lectura de datos en un fichero `cache_lectura.png` y otra gráfica para los fallos de escritura de datos en un fichero `cache_escritura.png`. En

cada caso, en el eje de las abscisas se indicará el tamaño de la matriz, mientras que en eje de las ordenadas se indicará el número de fallos. Deberá pintar una serie de datos por cada tamaño de caché indicado.

4) Justifique el efecto observado. ¿Se observan cambios en la gráfica al variar los tamaños de caché para el programa `slow`? ¿Y para el programa `fast`? ¿Varía la gráfica cuando se fija en un tamaño de caché concreto y compara el programa `slow` y `fast`? ¿A qué se debe cada uno de los efectos observados?

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

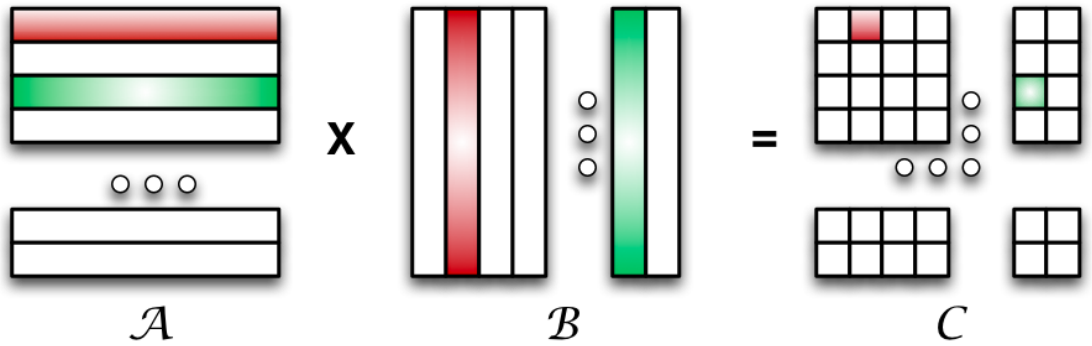
Se pide que los alumnos implementen un programa que lleve a cabo la multiplicación de dos matrices cuadradas del mismo tamaño. Este programa recibirá como único argumento el número N (=nº de filas = nº de columnas de las matrices).

Para simplificar la tarea, se han de usar las funciones de la 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, 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 el tiempo necesario para generar y liberar las matrices utilizadas).

IMPORTANTE: La impresión de datos por pantalla es una tarea más costosa de lo que pueda parecer, por lo que un programa que mida rendimiento de cálculo en términos de tiempo de cálculo nunca debe imprimir datos por pantalla durante el contaje de tiempo.

Recordatorio: al multiplicar dos matrices A y B, obtenemos una matriz C cuyos elementos se calculan según la expresión siguiente:

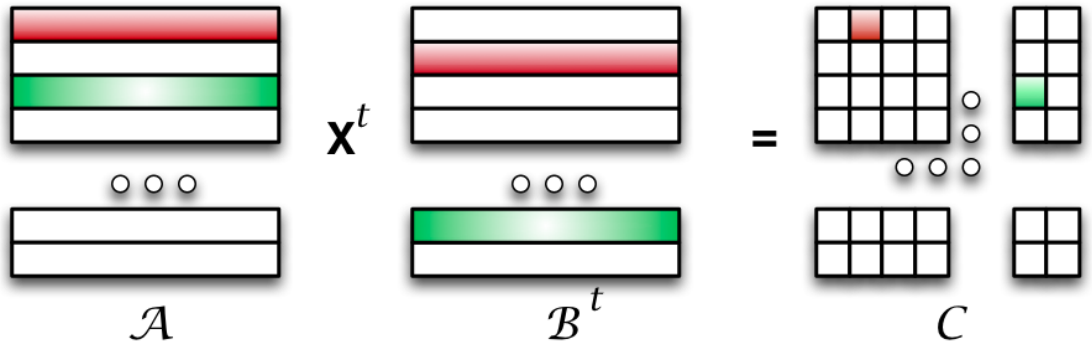
$$c_{ij} = \sum_{k=1}^{N} a_{ik} \cdot b_{kj}$$



- 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. Los elementos de la multiplicación traspuesta se calculan según la expresión siguiente:

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



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

IMPORTANTE: La versión del programa que realiza la "multiplicación traspuesta" debe tener en cuenta el tiempo de cálculo requerido para realizar la trasposición de la matriz B, pero no el tiempo de reservar y liberar dicha matriz, por lo que la matriz debe ser reservada antes de iniciar el contaje de tiempo y liberada después de finalizar dicho contaje.

Una vez codificadas ambas versiones de la multiplicación de matrices y verificado que reportan la misma matriz resultante C, se pide:

1) Tomar datos de los tiempos de ejecución obtenidos al ejecutar las dos versiones del programa que calcula la multiplicación de dos matrices de tamaño NxN, con N variando entre 128+16*P y 2176+16*P y con un incremento en saltos de 256 unidades. Deberá tomar resultados para todas las ejecuciones múltiples veces y de forma intercalada (como se indica en el ejercicio 1) y calcular la media de las mismas.

2) Tomar datos de la cantidad de 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 128+16*P y 2176+16*P y con un incremento en saltos de 256 unidades. En este caso se usará con *cachegrind* con la configuración de cachés por defecto. Recuerda que como valgrind es un emulador, solo es necesario **ejecutar 1 vez** el programa para cada configuración solicitada.

3) Guardar para la entrega el fichero con los resultados obtenidos. Por criterios de unificación, se pide que el fichero se llame `mult.dat` y siga el formato:

```
<N> <tiempo "normal"> <D1mr "normal"> <D1mw "normal">
<tiempo "trasp"> <D1mr "trasp"> <D1mw "trasp">
```

Nota: Es posible ejecutar los apartados 1) y 2) en un mismo script, lo que permitirá unificar la

salida de ambos programas de forma más sencilla que si se ejecutan en scripts separados. Pero recordad que solo se requiere 1 ejecución con valgrind.

4) Se han de representar los resultados obtenidos de las 3 medidas realizadas para cada una de las dos versiones del programa de multiplicación de matrices sobre tres gráficas generadas con GNUplot en formato PNG: una para los fallos de caché en lectura en un fichero llamado `mult_cache_read.png`, otra para los fallos de caché en escritura en un fichero llamado `mult_cache_write.png` y otra para los tiempos de ejecución en un fichero llamado `mult_time.png` e incluirlas en la memoria.

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

Ejercicio 4 (Opcional): Configuraciones de Caché en la multiplicación de matrices (2 p)

Este ejercicio no es de entrega obligatoria, y permite compensar errores de ejercicios anteriores para obtener la máxima puntuación de la práctica. Se pide a los alumnos que, para afianzar los conocimientos adquiridos en esta práctica, realicen un pequeño estudio sobre cómo afectan las distintas configuraciones de las cachés a la ejecución de los programas de multiplicación de matrices y multiplicación traspuesta de matrices.

El alumno puede "jugar" con los diferentes parámetros de configuración de las cachés estudiadas a lo largo de la práctica (pudiendo incluir, si así lo desean, la asociatividad y el tamaño de línea de la caché) así como el tamaño de las matrices multiplicadas. Los resultados que reporte pueden contemplar errores de lectura y escritura, así como tiempos de ejecución, y cualquier otro dato significativo que se le ocurra obtener.

Todos los experimentos que decida realizar, así como los resultados que obtenga deberán ir acompañados de una explicación que justifique el motivo por el que ha decidido realizar el experimento y las conclusiones que ha extraído del mismo. También se valorará en este ejercicio que el alumno proporcione los scripts de Bash y GNUplot o, en su defecto, los comandos Linux que haya empleado para obtener los resultados.

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/comandos 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 límite según lo indicado en el calendario de las prácticas, hasta las 23:59 de la noche.

EVALUACIÓN

Esta práctica, en total, suma 10 puntos la parte obligatoria más (hasta) 2 puntos el ejercicio opcional.

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

[4] **Bash Scripting tutorial.**
https://linuxconfig.org/bash-scripting-tutorial

[5] **Introduction to bash shell redirections.**
https://linuxconfig.org/introduction-to-bash-shell-redirections

[6] **GNUPLOT 4.2 - A Brief Manual and Tutorial.**
http://people.duke.edu/~hpgavin/gnuplot.html