Apéndice de la Práctica 4: Manejo básico de cachegrind

Para ejecutar un código usando la herramienta cachegrind, deberemos ejecutar el siguiente comando

```
valgrind --tool=cachegrind ./ejecutable
```

En la ejecución de Valgrind puede usarse la opción `-v` para ver detalles adicionales tales como la configuración de las cachés simuladas, que por defecto son las que tiene el sistema donde se está realizando la ejecución. Valgrind solo considera 2 niveles de caché, que configura con la información de la caché de nivel 1 y la caché de último nivel (LL) del sistema (caché de nivel 3 en la mayoría de los procesadores modernos), a la que denomina Last Level Cache. [1] Asimismo, ten en cuenta en la interpretación de los resultados que las tasas de fallos que ofrece Cachegrind para la caché de último nivel (LL miss rate) **no son** las tasas de fallos locales de la caché de último nivel, es decir, el porcentaje de accesos realizados a la caché de último nivel que resultan en fallos, sino que son las tasas de fallos globales, es decir, el porcentaje de todos los accesos realizados por el procesador que resultan en fallos en la caché de último nivel. Para calcular las tasas de fallos locales en la caché de último nivel debes dividir manualmente el número de fallos obtenidos en dicha caché (LL misses) entre el número de fallos obtenidos en la caché del primer nivel (I1 misses + D1 misses) y multiplicar por 100 el resultado. Por otra parte, Cachegrind siempre imprime los números de accesos y fallos separando con comas los millares, millones, etc. lo que te dificultará la tarea de copiar sus datos de salida para operar o hacer gráficos con ellos. Una opción para evitar este problema es filtrar la salida de Cachegrind borrando estas comas. Esto puede hacerse por ejemplo de la siguiente forma:

```
valgrind --tool=cachegrind ./ejecutable  2>&1 | sed -e 's/,//g'
```

La configuración de la cache de datos de primer nivel se controla mediante la opción

`--D1=<tamaño de caché>,<asociatividad>,<tamaño de línea>`

donde todos los tamaños vienen dados en bytes. La herramienta tiene restricciones en cuanto a las combinaciones a usar, de forma que se restringe a la simulación de cachés con configuraciones realistas. Así, por ejemplo sólo se permiten tamaños de línea que sean potencias de 2 iguales o mayores a 16, que son de hecho los que se encuentran en la práctica. Análogamente el tamaño de la caché de último nivel se controla con la opción

`--LL=<tamaño de caché>,<asociatividad>,<tamaño de línea>`

Al realizar sus ejecuciones, Cachegrind guarda en fichero un información más detallada a la que se puede acceder mediante el comando `cg_annotate`. Por defecto este fichero se llama `cachegrind.out.<pid>` (donde `pid` es el identificador del proceso ejecutado), pero puedes elegir el nombre que quieras para el fichero con la opción `--cachegrind-out-file=<fichero>` de Cachegrind. La salida de cg_annotate es muy ancha, con lo que conviene redimensionar el terminal a una anchura de al menos 120 caracteres antes de ejecutarlo. Si por ejemplo el pid de nuestra última ejecución fue 13195, podemos leer el fichero correspondiente mediante el comando

```
cg_annotate cachegrind.out.13195
```

La salida está explicada en el manual. Básicamente muestra la configuración de la jerarquía de caché simulada, los eventos recogidos, y las estadísticas obtenidas tanto para todo el programa como para cada función individual. Las funciones están ordenadas por defecto de mayor a menor `Ir`, es decir en orden decreciente del número de instrucciones leídas (y por tanto ejecutadas); y aparecen tanto aquellas que hemos escrito en nuestra aplicación como aquellas que hemos invocado directa o indirectamente. Así, por ejemplo, pueden aparecer funciones invocadas por el sistemas operativo durante la carga e inicio de la ejecución de nuestro proceso, o funciones de la librería estándar que hemos empleado. Cada función está etiquetada de la forma `fichero:nombre_función`, siendo uno cualquiera de los dos campos sustituido por la cadena `???` si cg_annotate no es capaz de deducirlo

[1] Algunas versiones de Valgrind no reconocen correctamente la caché de último presente en algunos procesadores. El tamaño real de esta caché así como el tipo de procesador se encuentran en el fichero del sistema `/proc/cpuinfo`

de la información disponible. Así, verás que en el caso de nuestro programa de ejemplo, la lista de funciones está encabezada por la función `main` de nuestro programa, pero en lugar de aparecer como `traspuesta.c:main` aparece como `???:main`. El motivo es que debemos compilar nuestras aplicaciones con la opción `-g` para que éstas contengan la información de depurado que cg_annotate emplea. Puedes encontrar más información sobre la operativa de esta opción en el manual del compilador (man gcc). Otra ventaja de compilar con `-g` el programa antes de instrumentarlo con Cachegrind es que éste puede almacenar la información del número de accesos, fallos de cachés, etc. no sólo para cada función, sino para cada línea del programa. Por defecto cg_annotate no muestra la información de comportamiento del programa línea a línea; para obtenerla debe invocarse con la opción `--auto=yes`. Así, si ejecutamos

```
gcc -g -O3 -o traspuesta traspuesta.c
valgrind --tool=cachegrind ./traspuesta
cg_annotate --auto=yes  cachegrind.out.13618
```

suponiendo que el pid de nuestro proceso haya sido 13168, bajo la lista de estadísticas de las funciones, encontraremos un listado con el comportamiento de cada línea en el ejecutable para la que cg_annotate haya encontrado la información de depuración, entre las cuales se halla la rutina `main` que hemos escrito.

Obsérvese que gracias a la herramienta `cg_annotate` es posible ver el número de fallos que se producen en cada línea de código. Podemos incluso ver el número de fallos por referencia si estructuramos nuestro código de forma que haya una sola referencia a memoria por línea, lo cual puede hacerse mediante transformaciones del tipo

$$\texttt{a[i] = b[i] + c[i];} \Longrightarrow \begin{cases} \texttt{br = b[i];} \\ \texttt{cr = c[i];} \\ \texttt{a[i] = br + cr;} \end{cases}$$

Esto tiene el inconveniente de introducir variables adicionales. Con un mínimo nivel de optimización el compilador debería ubicar las variables escalares en registros, con lo que no incidirían en las estadísticas de Cachegrind. Desafortunadamente, cuando se activan las optimizaciones, el compilador suele fusionar el código de varias líneas, con lo que `cg_annotate` no es capaz de proporcionar las estadísticas de cada línea del código original. Por ello, si observamos este comportamiento de fusionado de estadísticas de líneas en la zona del código que nos interesa y queremos averiguar el número de fallos de cada referencia individual, es mejor no usar ningún flag de optimización. En esta situación también habrá accesos a memoria para las variables escalares, pero podremos asumir con casi total seguridad que todos los accesos que producen estas variables resultarán en aciertos en la caché de datos del primer nivel.