Laboratorio de caché

J. Daniel García Sánchez (coordinador)

Arquitectura de Computadores
Grupo ARCOS
Departamento de Informática
Universidad Carlos III de Madrid

1. Objetivo

Esta práctica tiene como objetivo fundamental mejorar la comprensión por parte del estudiante de los conceptos relacionados con la memoria caché y las diversas optimizaciones que se pueden realizar en el software para un mejor aprovechamiento de la misma.

2. Descripción de la práctica

En esta práctica se realizará la evaluación de la ejecución de diversos programas en C++ con la herramienta **cachegrind**, que forma parte de las herramientas disponibles dentro de **valgrind** (http://valgrind.org/). La herramienta es capaz de simular la ejecución de un programa y el impacto que tiene la memoria caché sobre él.

Si el programa evaluado está compilado con el flag de información de depuración activado (**-g** en **gcc**), **cachegrind** puede además indicar cuál es la utilización de la memoria caché con nivel de detalle de línea de código fuente. Para ello, se debe generar primero el código ejecutable mediante el comando:

```
gcc -g test.cpp -o test
```

De este modo, se puede obtener la información de ejecución del programa utilizando valgrind:

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

Si no se le indica ningún parámetro adicional, **cachegrind** simula la ejecución del código sobre la configuración real de memoria caché del procesador de la máquina sobre la que se está ejecutando. Al finalizar la ejecución, muestra estadísticas básicas sobre el uso de la memoria caché y genera un fichero de nombre **cachegrind.out.<pid>** donde **pid** es el identificador del proceso. Este fichero contiene información adicional que puede ser analizada mediante la utilidad **cg_annotate**. (El nombre del fichero puede ser modificado mediante la opción **–cachegrind-out-file=<file_name>**).

Si se desea modificar la configuración de caché se deben utilizar las opciones:

```
--I1=<tamaño>,<asociatividad>,<tamaño línea>
```

Especifica el tamaño (en bytes), la asociatividad (número de vías) y el tamaño de línea (en bytes) de la memoria caché de instrucciones de nivel 1.

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

Especifica el tamaño (en bytes), la asociatividad (número de vías) y el tamaño de línea (en bytes) de la caché de datos de nivel 1.

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

Especifica el tamaño (en bytes), la asociatividad (número de vías) y el tamaño de línea (en bytes) de la caché de último nivel.

La herramienta **cg_annotate** recibe por parámetro el fichero que se desee analizar y la ruta de los ficheros que se quieren anotar línea por línea. Si no se sabe la ruta o cuáles son los ficheros de interés, se puede utilizar la opción **–auto=yes** para que anote todos los ficheros que puedan ser de interés.

```
cg_annotate cachegrind.out.XXXXX --auto=yes
```

Adicionalmente, si sólo se quiere anotar un fichero:

```
cg_annotate cachegrind.out.XXXXX <ruta absoluta al fichero .cpp>
```

Si se desea más información sobre el funcionamiento o la utilización de **cachegrind** y **cg_annotate**, se puede visitar la documentación en la página web de Valgrind: http://valgrind.org/docs/manual/cg-manual.html.

3. Tareas

Para la realización de este laboratorio se suministra código fuente para un conjunto de pequeños programas que se deben evaluar (ver siguientes secciones). También se suministra un archivo **CMakeLists.txt**.

Para compilar todos los programas a evaluar, se pueden usar los siguientes mandatos:

```
mkdir build
cd build
cmake .. -DCMAKE_BUILD_TYPE=Debug
make
```

NOTA: A tu grupo de laboratorio se le asignará una de las siguientes configuraciones:

- Configuración 1: Tamaño de línea de 32 B y todas las cachés son asociativas de 2 vías.
- Configuración 2: Tamaño de línea de 32 B y todas las cachés son asociativas de 4 vías.
- Configuración 3: Tamaño de línea de 64 B y todas las cachés son asociativas de 4 vías.
- Configuración 4: Tamaño de línea de 64 B y todas las cachés son asociativas de 8 vías.

3.1. Tarea 1: Fusión de bucles

En esta tarea se pretende analizar dos programas: **loop_merge.cpp** y **loop_merge_opt.cpp**.

Listing 1: loop_merge.cpp

```
1  int main(){
2    constexpr int maxsize = 100000;
3
4    float u[maxsize];
5    float v[maxsize];
6    float z[maxsize];
7    float t[maxsize];
8
9    for (int i=0; i<maxsize; ++i) {
10     u[i] = z[i] + t[i];
11   }
12   for (int i=0; i<maxsize; ++i) {
13     v[i] = u[i] + t[i];
14   }
15 }
```

Listing 2: loop_merge_opt.cpp

```
1  int main(){
2    constexpr int maxsize = 100000;
3
4    float u[maxsize];
5    float v[maxsize];
6    float z[maxsize];
7    float t[maxsize];
8
9    for (int i=0; i<maxsize; ++i) {
10     u[i] = z[i] + t[i];
11     v[i] = u[i] + t[i];
12   }
13 }
```

Ambos programas implementan la misma funcionalidad: dados dos vectores $\vec{z}$ y $\vec{t}$, calculan otros dos vectores $\vec{u}$ y $\vec{v}$:

$$\vec{u} = \vec{z} + \vec{t}$$

$$\vec{v} = \vec{u} + \vec{t}$$

Para ello, los programas hacen uso de 4 arrays de tamaño fijo. El programa no imprime ningún resultado.

Se pide:

1. Ejecute **loop_merge** y **loop_merge_opt** con el programa **valgrind** y la herramienta **cachegrind** para las siguientes configuraciones:

   - Caché de último nivel fijada a 128 KiB.

- Evalúe con tamaños de caché L1D de 16 KiB, 32 KiB y 64 KiB.

2. Observe los resultados obtenidos e inspeccione el código con la herramienta **cg_annotate**. Anote los resultados globales y observe los resultados prestando especial atención al cuerpo de los bucles.

3. Compare ambos resultados. Discuta en su informe los resultados para Dr, D1mr, DLmr, Dw, D1mw y DLmw.

3.2. Tarea 3: Estructuras y arrays

En esta tarea se pretende analizar dos programas: **soa.cpp** y **aos.cpp**. Ambos programas implementan la misma funcionalidad: suma de las coordenadas de dos conjuntos (**a** y **b**) de puntos con coordenadas en el plano. Uno de ellos hace uso de tres arrays de estructuras que representan puntos y el otro hace uso de tres estructuras con dos arrays. El programa no imprime resultado.

Listing 3: soa.cpp

```
1  constexpr int maxsize = 100000;
2
3  struct points {
4    double x[maxsize];
5    double y[maxsize];
6  };
7
8  int main() {
9    points a{}, b{}, c{}; // Default init
10
11   for (int i=0; i<maxsize; ++i) {
12     a.x[i] = b.x[i] + c.x[i];
13     a.y[i] = b.y[i] + c.y[i];
14   }
15 }
```

Listing 4: aos.cpp

```
1  struct point {
2    double x;
3    double y;
4  };
5
6  int main() {
7    constexpr int maxsize = 100000;
8    point a[maxsize], b[maxsize], c[maxsize];
9
10   for (int i=0; i<maxsize; ++i) {
11     a[i].x = b[i].x + c[i].x;
12     a[i].y = b[i].y + c[i].y;
13   }
14 }
```

Se pide:

1. Ejecute **soa** y **aos** con el programa **valgrind** y la herramienta **cachegrind** para las siguientes configuraciones:
   - Caché de último nivel fijada a 256 KiB.
   - Evalúe con tamaños de caché L1D de 8 KiB, 16 KiB y 32 KiB.
2. Observe los resultados obtenidos e inspeccione el código con la herramienta **cg_annotate**. Anote los resultados globales y observe los resultados prestando especial la cuerpo de los bucles.

3. Compare ambos resultados. Discuta en su informe los resultados para Dr, D1mr, DLmr, Dw, D1mw y DLmw

4. Entrega

La fecha límite para la entrega de los resultados de este laboratorio se anunciará a través de Aula Global.

Se seguirán las siguientes reglas:

- Todas las entregas se realizarán a través de aula global.
- El único formato admisible para el informe será PDF.
- El contenido del informe depende de su identificador de grupo de laboratorio. Por favor preste atención.

    1. Configuración 1: Grupos 1, 5, 9, 13, 17, 21, ...
    2. Configuración 2: Grupos 2, 6, 10, 14, 18, 22, ...
    3. Configuración 3: Grupos 3, 7, 11, 15, 19, 23, ...
    4. Configuración 4: Grupos 4, 8, 12, 16, 20, 24, ...