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.

```
--L2=<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 2.

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 **Makefile**.

Para compilar todos los programas a evaluar, se usará el comando **make**:

```
make
```

Si se desea compilar sólo uno de los programas, se puede ejecutar:

```
make programa
```

NOTA: Todas las configuraciones de caché que se utilizarán en las siguientes tareas tienen una asociatividad por conjuntos de 8 líneas.

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    double z[maxsize], t[maxsize], u[maxsize], v[maxsize];
4
5    for (int i=0; i<maxsize; ++i) {
6      u[i] = z[i] + t[i];
7    }
8    for (int i=0; i<maxsize; ++i) {
9      v[i] = u[i] + t[i];
10   }
11
12   return 0;
13 }
```

Listing 2: loop_merge_opt.cpp

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

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 el programa **loop_merge** con el programa **valgrind** y la herramienta **cachegrind** para las siguientes configuraciones:

   - L1 de 16 KiB con tamaño de linea de 32 B, L2 de 128 KiB con tamaño de linea de 64B
   - L1 de 32 KiB con tamaño de linea de 32 B, L2 de 256 KiB con tamaño de linea de 64B
   - L1 de 32 KiB con tamaño de linea de 64 B, L2 de 256 KiB con tamaño de linea de 64B

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 a las líneas 6 y 9.

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

   - L1 de 16 KiB con tamaño de linea de 32 B, L2 de 128 KiB con tamaño de linea de 64B
   - L1 de 32 KiB con tamaño de linea de 32 B, L2 de 256 KiB con tamaño de linea de 64B
   - L1 de 32 KiB con tamaño de linea de 64 B, L2 de 256 KiB con tamaño de linea de 64B

4. Observe los resultados obtenidos e inspeccione el código con la herramienta **cg_annotate**. Anote los resultados globales y observe los resultados prestando especial atención a las líneas 6 y 7.

5. Compare ambos resultados, ¿cuál es la diferencia en el ratio de aciertos y fallos en ambos niveles de cache?, ¿cuáles pueden ser las razones de estos cambios?, ¿es el resultado esperado?.

3.2. Tarea 2: Acceso secuencial

En esta tarea se pretende analizar dos programas: **access_seq.cpp** y **access_strided.cpp**.

Listing 3: access_seq.cpp

```
1  int main() {
2    constexpr int maxsize = 200;
3    double a[maxsize][maxsize] {}; // Default init
4    double b[maxsize][maxsize] {}; // Default init
5    double c[maxsize][maxsize]; // No init
6
7    for (int i=0; i<maxsize; ++i) {
8      for (int j=0; j<maxsize; ++j) {
9        c[i][j] = a[i][j] + b[i][j];
10     }
11   }
12
13   return 0;
14 }
```

Listing 4: access_strided.cpp

```
1  int main() {
2    constexpr int maxsize = 200;
3    double a[maxsize][maxsize] {}; // Default init
4    double b[maxsize][maxsize] {}; // Default init
5    double c[maxsize][maxsize]; // No init
6
7    for (int j=0; j<maxsize; ++j) {
8      for (int i=0; i<maxsize; ++i) {
9        c[i][j] = a[i][j] + b[i][j];
10     }
11   }
12
13   return 0;
14 }
```

Ambos programas implementan la misma funcionalidad: suma de dos matrices de dos dimensiones en una tercera matriz. El programa no imprime resultado.

Se pide:

1. Ejecute el programa **access_seq** con el programa **valgrind** y la herramienta **cachegrind** para las siguientes configuraciones:
   - L1 de 16 KiB con tamaño de linea de 32 B, L2 de 128 KiB con tamaño de linea de 64B
   - L1 de 32 KiB con tamaño de linea de 32 B, L2 de 256 KiB con tamaño de linea de 64B
   - L1 de 32 KiB con tamaño de linea de 64 B, L2 de 256 KiB con tamaño de linea de 64B
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 a la línea 9.

3. Ejecute el programa **access_strided** con el programa **valgrind** y la herramienta **cachegrind** para las siguientes configuraciones:

   - L1 de 16 KiB con tamaño de linea de 32 B, L2 de 128 KiB con tamaño de linea de 64B
   - L1 de 32 KiB con tamaño de linea de 32 B, L2 de 256 KiB con tamaño de linea de 64B
   - L1 de 32 KiB con tamaño de linea de 64 B, L2 de 256 KiB con tamaño de linea de 64B

4. Observe los resultados obtenidos e inspeccione el código con la herramienta **cg_annotate**. Anote los resultados globales y observe los resultados prestando especial atención a la línea 9.

5. Compare ambos resultados, ¿cuál es la diferencia en el ratio de aciertos y fallos en ambos niveles de cache?, ¿cuáles pueden ser las razones de estos cambios?.

3.3. 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 5: 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
16   return 0;
17 }
```

Listing 6: 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
15   return 0;
16 }
```

Se pide:

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

   - L1 de 16 KiB con tamaño de linea de 32 B, L2 de 128 KiB con tamaño de linea de 64B
   - L1 de 32 KiB con tamaño de linea de 32 B, L2 de 256 KiB con tamaño de linea de 64B

- L1 de 32 KiB con tamaño de linea de 64 B, L2 de 256 KiB con tamaño de linea de 64B

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 a las líneas 12 y 13.

3. Ejecute el programa **aos** con el programa **valgrind** y la herramienta **cachegrind** para las siguientes configuraciones:

   - L1 de 16 KiB con tamaño de linea de 32 B, L2 de 128 KiB con tamaño de linea de 64B
   - L1 de 32 KiB con tamaño de linea de 32 B, L2 de 256 KiB con tamaño de linea de 64B
   - L1 de 32 KiB con tamaño de linea de 64 B, L2 de 256 KiB con tamaño de linea de 64B

4. Observe los resultados obtenidos e inspeccione el código con la herramienta **cg_annotate**. Anote los resultados globales y observe los resultados prestando especial atención a las líneas 12 y 13;

5. Compare ambos resultados, ¿cuál es la diferencia en el ratio de aciertos y fallos en ambos niveles de cache?, ¿cuáles pueden ser las razones de estos cambios?.

3.4. Tarea 4: Producto de matrices

En esta tarea se pretende analizar dos códigos fuente: **product.cpp** y **product_block.cpp**. Ambos códigos implementan la misma funcionalidad: producto de dos matrices bidimensionales.El programa no imprime resultado.

Listing 7: product.cpp

```
1  int main() {
2    constexpr int maxsize = 100;
3
4    double a[maxsize][maxsize] {}; // Default init
5    double b[maxsize][maxsize] {}; // Default init
6    double c[maxsize][maxsize]; // No init
7
8    for (int i=0; i<maxsize; ++i) {
9      for (int j=0; j<maxsize; ++j) {
10       double r=0;
11       for (int k=0; k<maxsize; ++k) {
12         r += a[i][k] * b[k][j];
13       }
14       c[i][j] += r;
15     }
16   }
17
18   return 0;
19 }
```

Listing 8: product_block.cpp

```
1  int main() {
2    constexpr int maxsize = 100;
3
4    double a[maxsize][maxsize] {}; // Default init
5    double b[maxsize][maxsize] {}; // Default init
6    double c[maxsize][maxsize]; // No init
7
8    constexpr int bsize = 20;
9    static_assert(maxsize % bsize == 0,
10       "size must be multiple of blocksize");
11
12   for (int bj=0; bj<maxsize; bj+=bsize) {
13     for (int bk=0; bk<maxsize; bk+=bsize) {
14       for (int i=0; i<maxsize; ++i) {
15         for (int j=bj; j<bj+bsize; ++j) {
16           double r=0;
17           for (int k=bk; k<bk+bsize; ++k) {
18             r += a[i][k] * b[k][j];
19           }
20           c[i][j] += r;
21         }
22       }
23     }
24   }
```

```
25
26    return 0;
27  }
```

Se pide:

1. Ejecute el programa **product** con el programa **valgrind** y la herramienta **cachegrind** para las siguientes configuraciones:
   - L1 de 16 KiB con tamaño de linea de 32 B, L2 de 128 KiB con tamaño de linea de 64B
   - L1 de 32 KiB con tamaño de linea de 32 B, L2 de 256 KiB con tamaño de linea de 64B
   - L1 de 32 KiB con tamaño de linea de 64 B, L2 de 256 KiB con tamaño de linea de 64B
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 a la línea 12 y 14.
3. Ejecute el **programa product_block** con el programa **valgrind** y la herramienta **cachegrind** para las siguientes configuraciones:
   - L1 de 16 KiB con tamaño de linea de 32 B, L2 de 128 KiB con tamaño de linea de 64B
   - L1 de 32 KiB con tamaño de linea de 32 B, L2 de 256 KiB con tamaño de linea de 64B
   - L1 de 32 KiB con tamaño de linea de 64 B, L2 de 256 KiB con tamaño de linea de 64B
4. Observe los resultados obtenidos e inspeccione el código con la herramienta **cg_annotate**. Anote los resultados globales y observe los resultados prestando especial atención a las líneas 18 y 20.
5. Compare ambos resultados, ¿cuál es la diferencia en el ratio de aciertos y fallos en ambos niveles de cache?, ¿cuáles pueden ser las razones de estos cambios?.

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 la entrega será rellenando el cuestionario a través de Aula Global.
- La entrega y realización de los cuestionarios se hará de forma individual. Tenga en cuenta que las preguntas a contestar podrán ser distintas a la de otros de estudiantes.
- Una vez iniciado un cuestionario el estudiante dispondrá de un tiempo máximo de 30 minutos para completarlo.
- Cada estudiante dispondrá de un único intento para completar el cuestionario.
- El número máximo de cuestiones que tendrá que contestar cada estudiante será de 10.
- Se recomienda que el estudiante tenga preparadas en distintos archivos separados las soluciones a distintas tareas así como los datos asociados (ratio de acierto, ratio de fallo,número de aciertos, ...), puesto que se le podrá requerir que entregue estos archivos o que conteste alguna cuestión sobre la solución.