Laboratorio de caché

J. Daniel García Sánchez (coordinador) Arquitectura de Computadores

Grupo ARCOS Departamento de Informática

Universidad Carlos III de Madrid

# 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.

# Descripción de la práctica

En esta práctica se realizar 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 l.

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>.

# 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 CMake- Lists.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.

* 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(){

constexpr int maxsize = 100000;

float u[maxsize];

float v[maxsize];

float z[ maxsize];

float t[ maxsize];

for (int i=0; i<maxsize; ++i) { u[ i ] = z[i ] + t[ i ];

}

for (int i=0; i<maxsize; ++i) { v[ i ] = u[i] + t[ i ];

}

}

2

3

4

5

6

7

8

9

10

11

12

13

14

15

Listing 2: loop\_merge\_opt.cpp

1

int main(){

constexpr int maxsize = 100000;

float u[maxsize];

float v[maxsize];

float z[ maxsize];

float t[ maxsize];

for (int i=0; i<maxsize; ++i) { u[ i ] = z[i ] + t[ i ];

v[ i ] = u[i] + t[ i ];

}

}

2

3

4

5

6

7

8

9

10

11

12

13

Ambos programas implementan misma funcionalidad: dados dos vectores *z* y *t*, calculan otros

dos vectores u y v:

* u = z + t
* v = u + 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.

* + 1. Observe los resultados obtenidos e inspeccione el c digo con la herramienta cg\_annotate. Anote los resultados globales y observe los resultados prestando especial atención al cuerpo de los bucles.
    2. Compare ambos resultados. Discuta en su informe los resultados para Dr, D1mr, DLmr, Dw, D1mw y DLmw.
  1. 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;

struct points { double x[maxsize]; double y[maxsize];

};

int main() {

points a{}, b{}, c{}; // Default init

for (int i=0; i<maxsize; ++i) { a.x[ i ] = b.x[i ] + c.x[ i ];

a.y[ i ] = b.y[i ] + c.y[ i ];

}

}

2

3

4

5

6

7

8

9

10

11

12

13

14

15

Listing 4: aos.cpp

1

struct point { double x; double y;

};

int main() {

constexpr int maxsize = 100000;

point a[maxsize], b[maxsize], c[ maxsize];

for (int i=0; i<maxsize; ++i) { a[ i ]. x = b[i ]. x + c[i ]. x;

a[ i ]. y = b[i ]. y + c[i ]. y;

}

}

2

3

4

5

6

7

8

9

10

11

12

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.

* + 1. Observe los resultados obtenidos e inspeccione el código con la herramienta cg\_annotate. Anote los resultados globales y observe los resultados prestando especial al cuerpo de los bucles.
    2. Compare ambos resultados. Discuta en su informe los resultados para Dr, D1mr, DLmr, Dw, D1mw y DLmw

# 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 ser 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, . . .