Lab: Cache memory evaluation

J. Daniel Garcia (coordinator)
Computer Architecture
Computer Science and Engineering Department
University Carlos III of Madrid

2022

1 Objective

The goal of this laboratory is to increase the understanding by the student of the concepts related to cache memory and the different optimizations that can be done in order to better exploit this memory.

2 Description of the laboratory

In this laboratory the you will evaluate several C++ programs with **cachegrind** the tool, which is part of the tools available inside **valgrind** (`http://valgrind.org/`). This tool is capable of simulating the execution of a program and evaluating the impact of the cache performance.

If the evaluated program is compiled with the debugging information flag activated(**-g** in **gcc**), **cachegrind** can also provide the cache utilization for every line in the source code. For this purpose, the executable must be generated using the commands:

```
cmake -S . -B debug -DCMAKE_BUILD_TYPE=Debug
cmake --build debug
```

ATTENTION: Keep in mind that for executing jobs in the **avignon** cluster you must first create a *script* to launch the compilation.

This way, it is possible to obtain the information of the execution of the program using **valgrind** and **cachegrind**:

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

ATTENTION: Keep in mind that for executing jobs in the **avignon** cluster you must first create a *script* to launch **valgrind** executions.

If no additional parameter is provided, **cachegrind** simulates code execution on the real hardware where it is being executed. Otherwise, it can be used to evaluate a given cache configuration. When the execution finishes it shows basic statistics of cache usage. It also generates a file named **cachegrind.out.<pid>**, where **pid** is the process identifier. This file contains additional information that can be analyzed using the **cg_annotate** utility. (The name of the file can be modified with the option **–cachegrind-out-file=<file_name>**).

The following option can be used to simulate a given cache configuration:

```
--I1=<size in bytes>,<associativity>,<line size in bytes>
```

They specify the size (in bytes), the associativity (number of ways) and the size of the cache line of the instruction cache of level 1.

```
--D1=<size>,<associativity>,<line size>
```

They specify the size (in bytes), the associativity (number of ways) and the size of the cache line for the data cache of level 1.

```
--LL=<size>,<associativity>,<line size>
```

They specify the size (in bytes), the associativity (number of ways) and the size of the cache line of the last level cache.

The tool **cg_annotate** receives as parameter the file to be analyzed and the abslute path of the files that will be annotated line by line. If the path or the files to be annotated are not known the option **–auto=yes** can be used to annotate every file that can be of interest.

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

ATTENTION: Keep in mind that for executing jobs in the **avignon** cluster you must first create a *script* to launch **cg_annotate** executions.

Additionally, if only one file is going to be annotated:

```
cg_annotate cachegrind.out.XXXXX <absolute path to file .cpp>
```

For more information about the usage of **cachegrind** and **cg_annotate**,the documentation can be found in the webpage of Valgrind: http://valgrind.org/docs/manual/cg-manual.html.

3 Tasks

To complete this laboratory different small programs are given with their source code. These programs must be evaluated (see next sections). A **CMakeLists.txt** file for their compilation is also provided.

Create a directory named **lab3** in your home directory at **avignon**.

```
usuario@avignon-frontend:~$ mkdir lab3
usuario@avignon-frontend:~$
```

From your local machine, transfer all lab support files, that you have previously donwloaded:

```
usuario@mimaquina:~/Downloads/lab3$ scp * usuario@avignon.lab.inf.uc3m.es:~/lab3/
usuario@avignon.lab.inf.uc3m.es password:
aos.cpp 100% 237 20.6KB/s 00:00
CMakeLists.txt 100% 369 35.7KB/s 00:00
loop_merge.cpp 100% 254 22.7KB/s 00:00
loop_merge_opt.cpp 100% 216 17.2KB/s 00:00
soa.cpp 100% 249 24.1KB/s 00:00
usuario@mimaquina:~/Downloads/lab3$
```

To compile the programs to be evaluated, you may use the following commands in a script:

```
cmake -S . -B debug -DCMAKE_BUILD_TYPE=Debug
cmake --build debug
```

NOTE: Your lab group will be assigned one of the following configurations:

- Configuration 1: Line size of 32 B and caches are all 2-way set associative.
- Configuration 2: Line size of 32 B and caches are all 4-way set associative.
- Configuration 3: Line size of 64 B and caches are all 4-way set associative.
- Configuration 4: Line size of 64 B and caches are all 8-way set associative.

3.1 Task 1: Loop merging

In this task two programs must be analyzed: **loop_merge.cpp** and **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 }
```

Both programs implement the same functionality: given two vectors $\vec{z}$ and $\vec{t}$, they compute another two vectors $\vec{u}$ y $\vec{v}$:

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

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

For this purpose the programs use 4 fixed size arrays. The programs do not print any result. The student is asked to:

1. Run **loop_merge** and **loop_merge_opt** with the program **valgrind** and the tool **cachegrind** for the following configurations of cache:
   - Last level cache is fixed to 128 KiB.
   - Evaluate L1D cache sizes of 16 KiB, 32 KiB and 64 KiB.

2. Get the results obtained and inspect the code with the tool **cg_annotate**. Annotate the global results and pay special attention to the results on loop bodies

3. Compare results for both programs. Discuss in your report the results for Dr, D1mr, DLmr, Dw, D1mw y DLmw.

3.2 Task 2: Structures and Arrays

In this task two programs will be analyzed: **soa.cpp** and **aos.cpp**. Both programs implement the same functionality: sum of the coodinates of two sets of points (**a** and **b**) with coordinates in a bidimensional space. The second one uses three arrays of structures which represent the points and the first one uses three structures with two array inside each one. The programs do not print any result.

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 }
```

The student is asked to:

1. Run **soa** and **aos** with the program **valgrind** and the tool **cachegrind** for the following configurations of cache:
   - Last level cache is fixed to 256 KiB
   - Evaluate L1D cache sizes of 8KiB, 16KiB and 32 KiB
2. Get the results obtained and inspect the code with the tool **cg_annotate**. Annotate the global results and pay special attention to the loops.

3. Compare both results. Discuss in your report the results for Dr, D1mr, DLmr, Dw, D1mw y DLmw.

4 Submission

The deadline for the submission of the results of this laboratory will be published in Aula Global.
The student must follow the following rules:

- All submissions must be done through Aula Global.
- The only format for your lab report is PDF.
- Please, do not include in your report any screenshot. Use tables with data and graphs or charts made by your team.
- The content of the report depends on your lab group ID. Please, pay attention.
    1. Configuration 1: Groups 1, 5, 9, 13, 17, 21, ...
    2. Configuration 2: Groups 2, 6, 10, 14, 18, 22, ...
    3. Configuration 3: Groups 3, 7, 11, 15, 19, 23, ...
    4. Configuration 4: Groups 4, 8, 12, 16, 20, 24, ...