Cache Laboratory

J. Daniel García Sánchez (coordinator)

Computer Architecture
ARCOS Group
Computer Science and Engineering Department
University Carlos III of Madrid

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 student is asked to evaluate several C++ programs with the tool **cachegrind**, which is part of the tools available inside **valgrind** (http://valgrind.org/). This tool is capable of simulating the execution of a program and evaluation the impact of the cache memory over it.

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 command:

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

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

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

If no additional parameter is provided, **cachegrind** simulates code execution over the real processor cache configuration in which it is being evaluated. When the execution finishes it shows basic statistics of the usage of the cache and generates a file with the name **cachegrind.out.<pid>** where **pid** is the identifier of the process. This file contains additional information that can be analyzed using the utility **cg_annotate**. (The name of the file can be modified with the option **–cachegrind-out-file=<file_name>**).

For changing the configuration of the caches the following option should be used:

```
--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>
```


ARCOS@uc3m

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

```
--L2=<size>,<associativity>,<line size>
```

They specify the size (in bytes), the associativity (number of ways) and the size of the cache line of the data cache of level 2.

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
```

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 **Makefile** file for their compilation is also provided.

To compile the student can use the following command **make**:

```
make
```

If only one program must be compiled, the student can also execute:

```
name>
```

NOTE: All cache configurations that will be used in the next sections will be 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    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 }
```

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 pront any result. The student is asked to:

1. Run **loop_merge** with the program **valgrind** and the tool **cachegrind** for the following configurations of cache:
   - L1 of 16 KiB with line size of 32 B, L2 of 128 KiB with line size of 64B.
   - L1 of 32 KiB with line size of 32 B, L2 of 256 KiB with line size of 64B.
   - L1 of 32 KiB with line size of 64 B, L2 of 256 KiB with line size of 64B
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 of the line 6 and 9.

3. Run **loop_merge_opt** with the program **valgrind** and the tool **cachegrind** for the following configurations of cache:

   - L1 of 16 KiB with line size of 32 B, L2 of 128 KiB with line size of 64B.
   - L1 of 32 KiB with line size of 32 B, L2 of 256 KiB with line size of 64B.
   - L1 of 32 KiB with line size of 64 B, L2 of 256 KiB with line size of 64B

4. Get the results obtained and inspect the code with the tool **cg_annotate**. Annotate the global results and pay special attention to the results of the line 6 and 7.

5. Compare both results, what is the difference between the hit and miss rates of both cache levels?. What could be the reason of these changes?. Is the result the one expected?.

3.2 Task 2: Sequential Access

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

Both programs implement the same functionality: the sum of two bidimensional matrices in a third matrix. The programs do not print any result.

The student is asked to:

1. Run **access_seq** with the program **valgrind** and the tool **cachegrind** for the following configurations of cache:
   - L1 of 16 KiB with line size of 32 B, L2 of 128 KiB with line size of 64B.
   - L1 of 32 KiB with line size of 32 B, L2 of 256 KiB with line size of 64B.
   - L1 of 32 KiB with line size of 64 B, L2 of 256 KiB with line size of 64B
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 of line 9.

3. Run **access_strided** with the program **valgrind** and the tool **cachegrind** for the following configurations of cache:

   - L1 of 16 KiB with line size of 32 B, L2 of 128 KiB with line size of 64B.
   - L1 of 32 KiB with line size of 32 B, L2 of 256 KiB with line size of 64B.
   - L1 of 32 KiB with line size of 64 B, L2 of 256 KiB with line size of 64B

4. Get the results obtained and inspect the code with the tool **cg_annotate**. Annotate the global results and pay special attention to the results of line 9.

5. Compare both results, what is the difference between the hit and miss rates of both cache levels?. What could be the reason of these changes?. Is the result the one expected?.

3.3 Task 3: Structures and Arrays

In this task two programs must be analyzed: **soa.cpp** and **aos.cpp**. Both programs implement the same functionality: sum of the coodinates of two sets (**a** and **b**) of points 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 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 }
```

The student is asked to:

1. Run **soa** with the program **valgrind** and the tool **cachegrind** for the following configurations of cache:

   - L1 of 16 KiB with line size of 32 B, L2 of 128 KiB with line size of 64B.

- L1 of 32 KiB with line size of 32 B, L2 of 256 KiB with line size of 64B.
- L1 of 32 KiB with line size of 64 B, L2 of 256 KiB with line size of 64B

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 of lines 12 and 13.

3. Run **aos** with the program **valgrind** and the tool **cachegrind** for the following configurations of cache:

   - L1 of 16 KiB with line size of 32 B, L2 of 128 KiB with line size of 64B.
   - L1 of 32 KiB with line size of 32 B, L2 of 256 KiB with line size of 64B.
   - L1 of 32 KiB with line size of 64 B, L2 of 256 KiB with line size of 64B

4. Get the results obtained and inspect the code with the tool **cg_annotate**. Annotate the global results and pay special attention to the results of lines 12 and 13.

5. Compare both results, what is the difference between the hit and miss rates of both cache levels?. What could be the reason of these changes?. Is the result the one expected?.

3.4 Task 4: Matrix Product

In this task two programs must be analyzed: **product.cpp** and **product_block.cpp**. Both programs implement the same functionality: product of two bidimensional matrices. Theprograns do not print any result.

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   }
```


ARCOS@uc3m

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

The student is asked to:

1. Run **product** with the program **valgrind** and the tool **cachegrind** for the following configurations of cache:
   - L1 of 16 KiB with line size of 32 B, L2 of 128 KiB with line size of 64B.
   - L1 of 32 KiB with line size of 32 B, L2 of 256 KiB with line size of 64B.
   - L1 of 32 KiB with line size of 64 B, L2 of 256 KiB with line size of 64B
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 of lines 12 and 14.
3. Run **product_block** with the program **valgrind** and the tool **cachegrind** for the following configurations of cache:
   - L1 of 16 KiB with line size of 32 B, L2 of 128 KiB with line size of 64B.
   - L1 of 32 KiB with line size of 32 B, L2 of 256 KiB with line size of 64B.
   - L1 of 32 KiB with line size of 64 B, L2 of 256 KiB with line size of 64B
4. Get the results obtained and inspect the code with the tool **cg_annotate**. Annotate the global results and pay special attention to the results of lines 18 and 20.
5. Compare both results, what is the difference between the hit and miss rates of both cache levels?. What could be the reason of these changes?. Is the result the one expected?.

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 admissible for submission will be through the completion of the quiz in Aula Global.
- The submission and completion of the quizzes will be individual. The student must take into account that the questions to be answered could differ from person to person.
- Once the quiz is started the student will have a maximum of 30 minutes to complete it.
- Every student will have only one opportunity to complete the quiz.
- The maximum of questions that the student will have to answer will be 10.
- Students are recommended to have ready before starting the quiz (in different files) solutions to every task as well as their associated data (hit rate, miss rate, number of hits, ...), since they could be asked to submit those files or answer questions related to the data and solutions.