Advanced Computer Architecture
Lab 1 — Cache Simulation with Pin

1 Introduction

The purpose of this assignment is to give insight into:

- How a cache works.
- How different cache designs affect program execution.
- How a program can be tuned for a specific cache configuration.

You will extend a simple cache model and perform experiments with different programs and cache configurations.

You should team up and work in groups of two. This lab assignment is examined in the computer lab. During the examination, you will be asked to demonstrate and explain your solutions.

2 Brief overview of `Pin`

Pin[1] is a dynamic binary instrumentation framework. It allows you to easily insert code at specified parts of the program.

Lets consider the following example. A programmer would like to find out the total number of memory instructions that his application triggers. In this case, it is enough to write a simple C/C++ function that increments a variable and then set up Pin to execute it before or after every memory instruction it detects. Pin will then execute the application's binary and insert your code at the right moment, and by the end of the execution the variable will store the total number of memory instructions triggered.

The tool is provided by Intel[2] and is free to use.

3 Getting started

3.1 Linux servers

In this assignment we will be using the IT-departments linux servers[3]. To login on, connect with *SSH* to *tussilago.it.uu.se* or *vitsippa.it.uu.se*, e.g. using:

[1] `http://en.wikipedia.org/wiki/Pin_(computer_program)`

[2] `http://www.pintool.org/`

[3] Note that the linux machines are only available from within the university. If you want to log on from home, you first need to ssh to a sun server. See https://www.it.uu.se/datordrift/faq/unixinloggning for complete list of servers.

host$ ssh -Y *user-name*@tussilago.it.uu.se.

You should now be connected to one of the linux servers.

3.2 Setting up the environment

First we need to download Pin and install the necessary lab files. To do so, get the `install_lab1.sh` file from Piazza, and run the script by

host$ sh install_lab1.sh

It will install files in your home directory, ~/avdark.

4 Modifying the cache model

Edit the file `avdark-cache.c` in the source directory (`~/avdark/lab1/avdark-cache`) to modify the cache model. To rebuild the cache model run **host$ make**. If you prefer not to change the working directory use **host$ make -C ~/avdark/lab1/avdark-cache**. Remember to use the test cases to check that your modifications work. There are separate test cases for direct mapped caches (these should *always* work) and associative caches.

The internals of the cache model are fairly simple, most of the code is just Pin *glue code*, and it is only necessary to set up Pin. Your assignment boils down to rearranging the data line array or/and fix the tag and index computations.

All important functions are explained in the source code. It is recommended that you spend some time with the original cache model to get a basic understanding of it before you start to modify it.

5 The Assignments

5.1 Simulating an associative cache

At the moment the cache model is only direct mapped. Modify the cache model so that it can be configured as both a direct mapped (i.e. 1-way) and a 2-way associative data cache. The 2-way associative cache should use the LRU-replacement policy. Note that the cache model never handles actual data. The cache model only contains tags and valid bits.

To test the existing direct mapped cache you can use the `pin-avdc.sh` script to launch applications with the simulator. For example:

host$ ./pin-avdc.sh -a 2 -s 65536 -l 64 -- ls

The parameters before the double dashes (**--**) are passed to Pin and the simulator glue code. The simulator will output its results in `avdc.out` by default. The simulator glue code takes the following parameters:

-a ASSOC Set associativity. Default: 1

-s SIZE Set cache size. Default: 8388608 B

`-l BLOCK_SIZE` Set block size. Default: 64 B

`-o FILE` Set output file. Default: `avdc.out`

5.1.1 Evaluation

- Describe the modifications made to the cache model (by detailing the source code).
- Run 2 or 3 examples with different cache parameters to show those modifications (i.e. run "make test" and check that it works).

5.2 Miss Ratio measurements

Test the miss ratios for the RADIX program for the following cache settings:

Size [kB]	Block Size [B]	Associativity
16	32	1
16	32	2
32	16	1
32	32	1
32	64	1
32	16	2
32	32	2
32	64	2
64	32	1
64	32	2

Table 1: Cache configurations

Use RADIX with the -n 100000 option, i.e. run

```
host$ ./pin-avdc.sh -s <SIZE> -l <BLOCK_SIZE> -a <ASSOC>
-- ../radix/radix -n 100000
```

on the target machine. This makes RADIX sort 100000 keys. The RADIX binary is located in `~/avdark/lab1/radix`.

5.2.1 Evaluation

1. Live-run some examples with different cache parameters.
2. Examine and draw some conclusions from the cache-behavior of RADIX.

Note: You don't need to test all configurations in Table 1, 6 different configurations should be enough to be able to draw some conclusion. Explain how selected your cache configurations in that case.

5.3 Improving Cache Performance (optional, bonus)

In this assignment you should try to improve the cache performance of a given program example. The program is a simple matrix multiplication. Try to rewrite the program so that the cache miss ratio is decreased. Use some of the methods discussed in the course book to reduce the miss ratio of the program.

You may not change the size of the matrix or use less precision, but otherwise you are free to experiment with loop-indices, matrix-transponation, blocking etc. The multiplication should be correct, though. Also, you must use *gcc* when compiling it.

During the measurements use a *2-way* associative, *8 MB* cache with a block size of *64 B*.

To play with the optimizations, change the matrix parameter `SIZE` so that the unoptimized version runs for about 20 seconds. Then try to minimize the execution time without changing the `SIZE` parameter.

The original program is located in `~/avdark/lab1/multiply`.

Copy `multiply.c` and `Makefile` to a working directory and hack away!

The original code has built in support for verifying the results. You may activate the verification code using the **`-v`** option to the binary. It is a good idea to run the verifications on the host machine instead of the simulated machine to save some time.

5.3.1 Evaluation

1. Show the optimizations done to the multiplication program and explain them.
2. Verify all your numbers by running simulations and the different versions of the multiplication program.
3. Show that your implementation is an improvement.

6 Revisions

2011 — Andreas Sandberg
2012 — Andreas Sembrant
2013 — Andreas Sembrant, Mahdad Davari
2015 — Germán Ceballos, Moncef Mechri