CMU 18-741: Advanced Computer Architecture

Handout 2: Trace-Driven Cache Simulator Project

**** Due 9/27/2005 ****

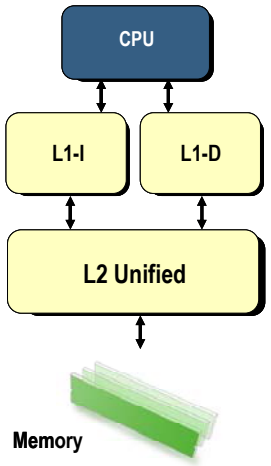
1. Introduction

In modern microprocessor designs, caches are essential for improving instruction execution time by mitigating long-latency accesses to main memory. Caching of instructions and data exploits temporal and spatial locality in order to create the illusion of fast and large memory. When designing caches, trace-based simulation is used to acquire useful cache statistics without implementing a complete processor model. This project is designed to 1) help you understand the structure and operation of a multi-level cache hierarchy, 2) exercise trace-driven methodology, and 3) characterize cache statistics.

This project must be completed in groups of 2 or 3 students. After forming a group, one member should email the TAs with the names and email addresses of all members by September 15th. Use the subject heading "Project 1 Group".

2. Cache Design

Your task is to model an inclusive two-level cache hierarchy using C or C++. There will be a split L1 instruction and data cache and a unified L2 cache. The figure below illustrates the high-level hierarchy. (Note: you will not be modeling the CPU or Memory.) Both the data L1 and unified L2 caches will be write-back, write-allocate, and all the caches will use LRU as the replacement policy. Other parameters (cache size, block size, associativity) will be configurable and specified in a configuration file. **All PC and memory references are 32 bits wide.** As a simplification, you may assume the block sizes in all the caches are the same.



Configuration File Format

The configuration file will have 4 lines, with the L1I configuration specified first, followed by the L1D, L2U configurations. The last line will specify the number of memory references to skip in the beginning of the trace before starting to collect cache statistics. (Note: you still simulate the skipped references.) The purpose of this last parameter is to allow the caches to "warm" so that initial "cold" misses do not affect your statistics. The default configuration file used is named "**config.default**".

Each of the cache configurations will be a single line with the following format:

[Cache Size] [Line/Block Size] [Associativity]

[Cache Size] is the total cache size in units of bytes (power of 2)

[Line/Block Size] is in units of bytes (power of 2)
[Associativity] is a power of 2

Here is one **possible** example configuration file:

```
8192 32 1
8192 32 4
65536 32 2
1000000
```

This example configuration has an 8 KB direct mapped L1I, 8 KB four-way set associative L1D, and a 64KB two-way set associative L2U. The block size for all three caches is 32 bytes, and statistics will begin counting after 1000000 memory references (instructions + data). We will be testing your simulator with a number of different configurations.

3. Tracing

A memory address trace is a sequence of memory references generated by the execution of a benchmark or program. Thankfully, you will not need to generate these traces on your own since we provide a set of traces sampled from the SPEC2000 benchmark suite (www.spec.org).

Each trace file, which represents a specific benchmark such as GCC, contains references due to 100 million instructions. Given that on average 30% of all instructions are loads or stores, there will be approximately 100 million PC addresses + 30 million load or store addresses per benchmark.

All traces are in ASCII format, compressed as .gz files and stored under:

```
/afs/ece/class/ece741/.vol1/specint        (integer benchmarks)
/afs/ece/class/ece741/.vol2/specint        (integer benchmarks)
/afs/ece/class/ece741/.vol3/specfp         (floating point benchmarks)
```

Since each trace (when uncompressed) occupies gigs of storage, it is infeasible for you to process the traces in an uncompressed format under AFS. Instead, we will require you to "pipe" compressed traces directly into your simulator (via "stdin").

The following command is an example of how we expect your program to run:

```
zcat tracefile.gz | ./yourCacheSim
```

If you had a simple trace file in uncompressed ASCII format, your program should execute as:

```
./yourCacheSim < mytrace.ascii.txt
```

Each line in an uncompressed trace file contains two fields separated by a space. The first field is a number signifying the reference type, and the second field is the memory address (in hex string). There are three possible reference types:

"0" represents a PC address
"1" represents a load address
"2" represents a store address

Remember that all PC and memory addresses are 32 bits wide.

An example of a trace file is shown:

```
0 200264d4
1 1ff952d0
0 200264d8
```

```
1 1ff952d0
0 200264dc
2 1ff952d0
0 200264e0
0 20026560
```

4. Cache Statistics

Your cache simulator must output (via stdout) the following statistics after processing a trace:

- Total number of accesses (fetch PC, Load, Store) for L1I, L1D, L2U
- Read miss rate for L1I, L1D, L2U
- Write miss rate for L1D, L2U
- Clean evictions for L1I, L1D, L2U (not including forced evictions due to inclusion)
- Writebacks for L1D, L2U (not including forced writebacks due to inclusion)
- Forced clean evictions for L1I, L1D
- Forced dirty writebacks for L1D
- A checksum (printed in hex) for L1I, L1D, L2U

The checksum will merely be the XOR of {tag, valid, dirty} from all valid blocks in the cache. In pseudocode:

```
unsigned long checksum = 0;
for (all valid cache blocks)
  checksum = checksum ^ ((tag << 2) | (validbit << 1) | (dirtybit));
```

Note that this checksum is order independent.

To ensure everyone has the same checksum and statistics, please obey the following guidelines:

- All tags, valid bits, and dirty bits should be initialized to 0
- Forced evictions out of the L1 caches (due to inclusion) should not be counted in the L2 statistics
- Remember, the checksum is only aggregated over **VALID** blocks

Some small trace files and their corresponding outputs will be provided in /afs/ece/class/ece741/project1/trace* directories to help you get started with debugging.

Here is an example of performance statistics output:

```
---------------------------------------------------------------------------
[L1-Inst] Accesses: 7265898
[L1-Inst] Misses: 17076

[L1-Inst] Reads: 7265898
[L1-Inst] Read misses: 17076
[L1-Inst] Read miss rate: 0.00235016

[L1-Inst] Miss rate: 0.00235016
[L1-Inst] Clean evictions: 7343

[L1-Inst] Forced clean evictions: 9635
[L1-Inst] Checksum: 0
---------------------------------------------------------------------------
[L1-Data] Accesses: 2734102
[L1-Data] Misses: 220588

[L1-Data] Reads: 1966123
[L1-Data] Read misses: 213269
[L1-Data] Read miss rate: 0.108472
[L1-Data] Writes: 767979
[L1-Data] Write misses: 7319
[L1-Data] Write miss rate: 0.00953021
```

```
[L1-Data] Miss rate: 0.0806802
[L1-Data] Clean evictions: 162701
[L1-Data] Dirty writebacks: 51106

[L1-Data] Forced clean evictions: 3155
[L1-Data] Forced dirty evictions: 3372
[L1-Data] Checksum: 3007EC
--------------------------------------------------------------------------------
[L2-Unif] Accesses: 237664
[L2-Unif] Misses: 181947

[L2-Unif] Reads: 230345
[L2-Unif] Read misses: 178387
[L2-Unif] Read miss rate: 0.774434
[L2-Unif] Writes: 7319
[L2-Unif] Write misses: 3560
[L2-Unif] Write miss rate: 0.486405

[L2-Unif] Miss rate: 0.765564
[L2-Unif] Clean evictions: 135781
[L2-Unif] Dirty writebacks: 47490
[L2-Unif] Checksum: 2F6B5
```

5. Requirements

The following is a list of what we expect for a completed project:

- A working cache simulator tested and ran to completion on all of the benchmark traces
- Source files
- Makefile
- Readme file
- Design review answers (see section 7)

You may use any operating system for your project, but your project will be tested on the Greek cluster (e.g. alpha.ece.cmu.edu), so make sure your final submission works on these machines.

Simply typing "make" should build your cache simulator and the executable created should be named "**cache_sim**". When running your cache simulator, the configuration should be read from a file named "**config.default**" in the same directory as the executable.

The following will be the sequence of commands we will use when grading your project:

```
make
cp our_config_choice config.default .
zcat benchmark_we_pick.gz | ./cache_sim
```

Please include a README file with your code, which should contain the names of the group members, and e-mail addresses at which you can be contacted in case there are any problems. If there are any other comments about the project (e.g. project does not work), please include it in the README file. A sample README file can be found at /afs/ece/class/ece741/project1/.

6. Grading

Your project will be graded according to the following:

40%	**Cache simulator builds and runs without crashing**
20%	**Cache simulator works correctly on a small test case**
20%	**Statistics/checksum correct on full test cases**
20%	**Design review answers**

In the first criteria, your simulator must build, run to completion for 5 selected benchmarks of our choosing, and output statistics in the format specified. We shouldn't have to look at your source code to get it working on the Greek cluster. This portion accounts for 40% of your project grade. To earn the next 20%, you simulator must also do something reasonable against a small test case.

In the third criteria, we will be checking your cache against expected statistics and checksums using 5 selected benchmarks (and various cache configurations). For each benchmark that matches, you will earn 4% (4% * 5 benchmarks = 20%).

Finally, the remaining 20% of your grade will depend on your design review answers.

Please consult the TAs or the instructor well ahead of the due date if any part of this project specification is not clear. You only have 2 weeks to complete this project. Check Blackboard for updates and announcements.

7. Design Review

Please submit "brief" answers to the following questions in ASCII or PDF format. (Get to the point. Please don't write an essay for each.)

1. Why is a multi-level cache hierarchy desirable?
2. When is a split L1-D and L1-I cache more desirable than a unified L1 cache? When is it not desirable?
3. Discuss the tradeoffs between having an inclusive vs. non-inclusive cache hierarchy.
4. Give 3 examples why LRU may not be the optimal replacement policy. For those examples, what are the best replacement policies?
5. Give 2 shortcomings/limitations of trace-driving cache simulations.

8. Submission

The final submission will be done electronically. Copy all relevant files to the directory in /afs/ece/class/ece741/project1/submission/your_email which matches the email of the student who e-mailed the group names. The directory is unlocked until the deadline so you can make changes until then if you submit early.