Inf-2200 Computer Architecture and Organization 2018 – Mandatory Assignment 3: Memory simulator

Handout: Friday, October 12th 2018

Deadline: Friday, November 2nd 2018 @ 12:00 PM (noon)

Introduction

In this assignment, you will implement a cache simulator for a memory system with a level-1 instruction cache, a level-1 data cache and a unified level-2 cache. Your goal is to use the simulator to find the best cache design for your benchmark from mandatory assignment 1. To do this you will measure the cache hit and miss ratios, and experiment with different parameters for the caches.

Cache specifications

Some specifications for the cache are fixed:

As a starting point, you should use the following specifications:

- The CPU – L1 bus width is 32 bits.
- The L1 – L2 bus width is the L1 block size
- The L2 – RAM bus width is the L2 block size.

In addition the table below suggests initial parameters for each of the caches. Note that you design should make it easy to change the different parameters.

	L1 Instruction cache	L1 data cache	L2 unified cache
Cache size	32 KB	32 KB	256 KB
Cache associativity	4-way associativity	8-way associativity	8-way associativity
Replacement policy	Approximated LRU	Approximated LRU	Approximated LRU
Block size	64 bytes	64 bytes	64 bytes
Bus width	64 bytes	64 bytes	64 bytes
Write policy	Write-back	Write-back	Write-back

Cache simulator implementation

As a starting point for your simulator implementation, we provide pre-code that implements the API of a memory subsystem and a CPU using this API. The simulator executes instruction fetch, data load and data store memory operations based on a given memory trace file. The trace is stored in a binary format specified by the *p2AddrTr* structure in the *byurt.h* header file. You are required to write an implementation of a memory hierarchy over the memory subsystem API containing modules for the caches. It is not necessary to make any changes to the pre-code, except memory.c, which is your starting point.

Your cache design and implementation should satisfy the following requirements:

1. The caches should be possible to implement in real hardware, that is, the cache parameters must be realistic according to your textbook.
2. Cache parameters such as size, block size, write-back policy and associativity should be easily changeable, either as runtime arguments or using compile time flags (defines).
3. The L1 data cache and L2 caches should support both reads and writes, and the L1 instruction cache must be read-only (you may want to use a generic cache implementation that you can use for all three caches, so you don't need to enforce a mechanism for preventing writes to the L1 instruction cache instance, just make sure you never write to it).
4. The caches should support the parameters and policies specified above, in addition to the parameters and policies specified in the evaluation section below.

You can make some simplifications:

1. You can assume that each data access is within a cache line.
2. For simulating instruction fetch, you can assume that all instructions are of a fixed size (32 bit) and that the addresses are word-aligned (note that the IA-32 instructions have variable length and may not be word aligned, so effectively you may ignore the last two bits of the accessed memory addresses to pretend that accesses are aligned).

Methodology

You will evaluate the cache by creating a memory access trace that contains all the memory accesses of a program. You should create at least two traces:

1. A trace used to test correctness of the simulator. For this trace you should know in advance the number of cache hits and misses the trace is going to produce, given a set of cache parameters.
2. A trace of memory addresses collected for you benchmark from assignment 1. You will use this to evaluate the cache parameters in order to find the best cache design for your benchmark.

The correctness trace may need to be created manually be starting from a known cache state, and then adding one-by-one memory operations to the trace.

To create a trace file for your benchmark, you must first produce a log file over memory accesses that you will convert to the given binary trace format using the traceconverter.py script from the pre-code. To produce the log file, you can either use a memory tracing tool such as Valgrind/ lackey, or by doing an analysis of your program to find the memory access pattern and then create a trace that reflects this (for example by writing a script that creates the logfile). To run valgrind on your benchmark, you may execute the following command:

```
valgrind --log-file=logfile --tool=lackey --trace-mem=yes [program-name]
```

This will produce a file logfile that may be used as input to the *traceconverter.py* script to create a trace file *trace.tr*.

You should make sure that the dataset size used to create the trace is realistic. In case it is not practical to create a memory trace, due to time or storage size constraints, for a realistic dataset you need to discuss in your report how the reduced size influences the measured results

Evaluation

Your goal is to find the cache configuration that gives the best performance for your benchmark at the lowest hardware implementation cost. To do this, you will start with the default parameters listed above. Then you should change one parameter at a time and measure the changes in hit and miss ratio for each cache. The parameters you will change for each cache are:

- Total cache size
- Block size
- Associativity
- Write-policy

However, you are also allowed to change additional parameters. Note that the number of sets in a given cache is a dependent variable given from the total cache size, block size and associativity (number of ways = number of blocks per set). Your report should present the results from you measurements and provide a discussion about the cost of implementing your final cache design in hardware.

Deliverables

You are allowed to either work alone or in groups of two. Each group is required to submit a single code and report, but we recommend that both persons do all parts of the project independently. The goal of the report is to convince a computer architecture expert that your cache design is the best for your particular benchmark. You should assume that the expert may want to repeat your experiments to verify your results. The report should therefore contain all the information required to do this, including the following parts:

1. Introduction: where you describe your benchmark, its realistic input data, and its memory access pattern.
2. Cache simulator implementation and design: where you provide the details for the expert to re-implement your cache simulator.
3. Experiment methodology: where you describe how to do the experiments, including:
   a) How you count cache hits and cache misses for each cache.
   b) The design of your correctness test trace, including how to use them to test for correctness.
4. Experiment results: where you present the evaluation results.
5. Discussion: where you discuss the most interesting result with respect to hardware cost.
6. Conclusion: where you write a short summary about the best cache design.

The assignment will be given a pass / not pass grade. The grade is based on the simulator implementation and on the written report