Computer Architecture 2012/2013
Assignment 3a

Date: November 13th, 2012
Deadline: November 25th 2012, 23:59

Contents

1 Overview

Reminder: The purpose of assignment series 3 is to evaluate the cache behavior of programs using MGSim, using MIPS code as made possible by assignment series 2.

The goal of assignment 3a is to:

- construct infrastructure to measure memory latencies;
- construct infrastructure to sample cache hit and miss rates;
- automate the computation of minimum, maximum, median, average latencies;
- automate the construction of latency frequency histograms.

2 Instructions

- For this assignment, you can work in groups of 2.
- Read this entire document before you start.

- Your must submit a compressed tarball[1], named after your last name and student ID, containing:
  - the files that you have produced during the assigment.
  - a file `report.rst` containing your write ups to open questions using reStructured Text. This must also contain your full name and student ID. Ensure that `report.rst` is valid by using `rst2html`.
- Your submission must be sent by e-mail before the deadline, at the e-mail address given by the assistants. Do not send your submission to the mailing list!

3 Prerequisites

You will need the MIPS infrastructure from assignment series 2.

4 Memory traces

You have previously used `mgsim` to run programs with the simulator. For example:

```
mgsim -c minisim.ini hello.bin
```

1. Select a test program which performs load and store operations, then run `mgsim` on this program with the command-line argument `-i`. This starts the simulator in interactive mode. First enter `trace mem` at the prompt, then `run` to start the program. Count how many events are reported. Compare with the dynamic load/store instruction count.

2. You can automate the capture of all memory events to a file as follows:

   ```
   echo "trace mem; run; quit" | \
       mgsim -c minisim.ini test.bin -i 2>&1 | \
       grep '^\[' >trace.log
   ```

   Use this command on your test program and include the resulting trace in your submission.

3. Using regular expressions, write a program `gather` in a scripting language of your choice (eg. shell, Python, Perl) which reads as input a file produced by step #2 and which, for each *load* (ie not stores) in this trace, reports the access latency.

   The output of your program should be composed by one line of text for each load. Each line should be composed of three columns. The first column should report the simulation cycle at which the load was issued in decimal. The second column should report the target address of the load in hexadecimal. The third column should report the load latency in the same unit and base as the first column.

   You can use the following information:

[1]A compressed tarball is created with `tar -czf xxxx.tgz ....`

- the current time in simulation cycles is indicated at the beginning of each trace event.
- a message of the following form is a load hit:

  ```
  ... load *0x000200f8/2 -> R0001 int:[F:00008000]
  ```
- a message of the following form is a load miss, followed later by the the load completion:

  ```
  ... load *0x000200fc/1 -> delayed R0001
  ... Completed load: 0x0000000000000002 -> R0001
  ```
- a message of the following form is a store, followed later by the store acknowledgement:

  ```
  ... store *0x000200f8/2 <- R0000 int:[F:0000f000]
  ... T0 completed store
  ```

5 Memory statistics

1. Write another program `stats` which reads as input the output of `gather` above, and displays the following values on one line separated by spaces:
   - the minimum load latency,
   - the maximum load latency,
   - the median load latency,
   - the average load latency.
2. Write another program `freq` which automatically produces a histogram of the access latencies. For example, you can reuse the information given here:

   http://stackoverflow.com/questions/2471884/histogram-using-gnuplot

Note
In both cases, feel free to reuse existing third-party tools that simplify the task, as long as these tools are available on the LIACS workstations.

3. Using your programs above, report the memory statistics from all the test programs you have used in series 1 & 2. For the sake of the experiments, try to find (or construct) at least one test program which performs more than 1000 loads.

6 Cache hits and misses

At the interactive prompt of MGSim, it is possible to print the current cache hits and misses using the following commands:

```
read cpu0.dcache
read cpu0.icache
```

Moreover, if MGSim is run with `-o MemoryType=FLATCOMA`, a L2 cache is included as well and its statistics can be printed with:

```
read memory.cache0
```

Finally, regardless of the combination of caches selected, the number of load requests that arrive to the memory can be displayed as follows:

```
read memory:nreads
```

You can then automate the gathering of these statistics with:

```
echo "...; read ...; read ...; quit" | mgsim ...
```

1. Write a program `cachestats` which reports, given the output of the `read` commands above, a single line with 4 values separated by spaces:
   - the number of load requests issued to the L1 cache;
   - the L1 hit rate;
   - the number of load requests issued to the memory (after all caches);
   - if applicable, the L2 hit rate.
2. Run `stats` and `cachestats` for all the test programs you have isolated above.

   Then build a graph with one point per program in the x-axis, which shows in the y-axis the results from `stats` and `cachestats`. If possible, try to sort the programs by their cache hit rates.

7 Summary of submission contents

Your final submission archive should contain the following files:

- `report.rst` (your report with explanations);
- your `gather` , `stats` and `freq` programs;
- the `bin` files for the test programs you have used;
- the latency histograms;
- the latency/cache graph.

8 Grading

You will be evaluated as follows:

- whether your `gather` program works as requested (3pt);
- whether your `stats` program works as requested (1pt);
- whether your `freq` program works as requested (1pt);
- whether your `cachestats` program works as requested (2pt);
- whether and how you have evaluated your test programs as requested (3pt).