C152 Laboratory Exercise 2

Professor: Krste Asanovic
TA: Scott Beamer
Department of Electrical Engineering & Computer Science
University of California, Berkeley

February 17, 2009

1 Introduction and goals

The goal of this laboratory assignment is to allow you to conduct some simple memory hierarchy experiments in the Simics simulation environment. Using the g-cache cache simulator module, you will collect cache statistics and make some architectural recommendations based on the results.

The lab has two sections, a directed portion and an open-ended portion. Everyone will do the directed portion the same way, and grades will be assigned based on correctness. The open-ended portion will allow you to pursue more creative investigations, and your grade will be based on the effort made to complete the task or the arguments you provide in support of your ideas.

Students are encouraged to discuss solutions to the lab assignments with other students, but must run through the directed portion of the lab by themselves and turn in their own lab report. For the open-ended portion of each lab, students can work individually or in groups of two or three. Any open-ended lab assignment completed as a group should be written up and handed in separately. Students are free to take part in different groups for different lab assignments.

You are only required to do one of the open ended assignments. These assignments are generally starting points or suggestions. Alternatively, you can propose and complete your own open ended project as long as it is sufficiently rigorous. If you feel uncertain about the rigor of a proposal, feel free to consult the TA or professor.

1.1 Tools and conventions

This lab assumes you have completed the first laboratory assignment. While we will re-include all the files used for both labs in this lab's distribution bundle for your convenience, there will be points where you can save time by reusing checkpoints and such created in the first lab.

Furthermore, we will assume that you remember all the commands used in the first lab for controlling Simics simulation. If you feel any confusion about these points, feel free to consult the first lab guide or the Simics User Guide.

Several new tools will be introduced in this lab, mainly having to do with the g-cache module. We will also introduce a new mode of Simics operation (`-stall`). For a more thorough explanation of these items, you can refer to the Simics User Guide chapter 18.

1.2 Graded Items

You will turn a hard copy of your results to the professor or TA. Please label each section of the results clearly. The following items need to be turned in for evaluation:

1. Problem 2.2: simple cache statistics for each benchmark and answers
2. Problem 2.3: suggested working sets and evidence
3. Problem 2.4: statistics and answers
4. Problem 2.5: complex cache statistics and answers
5. Problem 3.1/3.2/3.3/3.4 modifications and evaluations (include source code if required)

2 Directed Portion

2.1 General methodology

At its core, the Simics software is an instruction set architecture simulator, not a machine performance simulator. While Simics presents an interface to the OS and programs running within it that makes the target machine appear to be a normal machine, in actuality many of the simulated machine's functions are idealized far beyond the capabilities of a real machine. For example, in normal execution mode, main memory accesses in Simics appear to take zero cycles.

However, it is possible to attach a cache simulator module to Simics and use it to gauge a program's cache performance or the effectiveness of a particular cache hierarchy. The cache simulation module included with Simics is called `g-cache`. These modules can be linked into hierarchies, connected to multiple processors and have a variety of adjustable parameters. `g-caches` are configured and attached to the memory hierarchy by means of simple Simics scripts containing python commands (the @ prefix tells Simics to interpret a statement as python).

One of these `g-cache` parameters is the delay accrued by a memory request as it passes down through each level of the hierarchy to the simulated memory interface. By running Simics in `-stall` mode, users can force the delayed execution of instructions according to whether or not the data the instructions require is contained in caches or memory. Simultaneously, the cache modules record statistics about the memory requests which have accessed them. `-stall` mode is a more detailed mode of simulation, and therefore noticeably slows down the operating speed of the target machine.

A further methodological detail is that when the caches are first attached to the simulation, they are empty. Any statistics recorded from them initially will only reflect the compulsory misses encountered as they fill up. For this reason, it is necessary to 'warm' the caches by running the intended benchmark for millions of instructions before the true cache performance statistics can be collected.

Thus, the general methodology of this lab takes the following form:

1. Run the simulation at full speed to get the file system loaded, the benchmarks set up, etc.
2. Checkpoint the simulation
3. Restart simulation in `-stall` mode

4. Load the caches using a .simics script

5. Warm the caches by executing benchmark code

6. Pause or breakpoint the simulation

7. Reset all the caches' statistics

8. Continue to run the benchmark and collect real cache data

Data from the cache modules can take several forms. `<cache name>.info` reports the configuration of the cache, `<cache name>.status` displays the current value of every cache line, `<cache name>.statistics` reports cache performance statistics, and `<cache name>.add-profiler` adds a profiler module which tracks cache misses on a per instruction or address basis.

You can use any of the `ilinux{1,2,3}.eecs.berkeley.edu` instructional servers to complete this lab assignment. Do not wait until the night before the assignment is due, because you will face resource contention that may significantly increase the time it takes to complete the assignment.

2.2 Collecting statistics from a simple cache

You should untar the file `benchmarks-1.tar` distributed with this Lab (`tar -xvf benchmarks-1.tar`) and make sure the three binary and three input files are available in your `simics-workspace` directory. The binary files for this lab are the same as the ones used in Lab 1 (`bzip2_sparc`, `mcf_sparc`, `soplex_sparc`).

Start Simics, using either the `targets/sunfire/bagle-common.simics` script or a checkpoint you created in the previous lab. Make sure the host filesystem or workspace is mounted and that the 3 benchmark binaries and 3 benchmark input files are copied into the target machine. To save time in the following sections, you should create a checkpoint that has all the files loaded, or possibly a checkpoint at each of the initial magic breakpoints in the benchmark programs.

For each benchmark:

- Start Simics in stall mode and load the proper configuration file
  ```
  host$ ./simics -stall -c <configuration name>
  ```

- Execute the following commands in order to ensure the proper operation of the benchmarks and caches:
  ```
  simics> magic-break-enable
  simics> istc-disable
  simics> dstc-disable
  ```

- Load the cache module into the simulation (a .simics file is simply a sequence of Simics commands).
  ```
  simics> run-command-file add-1cache-bagle-2.2.simics
  ```

- Look at the configuration of the cache you have loaded, and compare this to the contents of the .simics file you just executed. Observe how the cache is configured with simple declarative statements in Python.
  ```
  simics> cache.info
  ```