C152 Laboratory Exercise 4

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

March 19, 2009

1 Introduction and goals

The goal of this laboratory assignment is to allow you to conduct some simple virtual experiments in the Simics simulation environment. Using the Simics Microarchitectural Interface and an out–of–order execution processor model, you will collect 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 in general 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.

It is also important to stress that how concise the report is and how the data is presented will be taken into account when grading. Problems usually are specific about what statistics they want, so there is no need to give them all. Tables and especially graphs are much more efficient and effective ways to communicate data.

This lab assumes you have completed the earlier laboratory assignments. However, we will re-include all the relevant files from past labs in this lab's distribution bundle for your convenience. Furthermore, we will assume that you remember all the commands used in earlier labs 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.

1.1 Simics MAI Overview

The Simics simulator by default assumes that every instruction completes instantaneously in a single cycle. This allows for speedy simulations that are useful for software/firmware correctness

testing. As we saw in previous labs, Simics can be extended with memory hierarchy timing modules to model realistic performance effects of a user-defined memory hierarchy. These extensions increase simulation realism at the expense of simulation speed.

In this lab, we will make use of further extensions which allow an instruction's execution to be delayed according to a microarchitectural model. This model can be programmed to simulate the timing behavior of the instructions as if they were being run on an in-order or out-of-order execution (OOE) processor. Microarchitectural models interact with Simics execution via the Micro-Architectural Interface.

Microarchitecturally accurate models are coded using C or the Simics Device Modeling Language. For this lab, we will use MAI modules included with Simics, rather than code our own. Specifically, we will use the MAI extension for the Sunfire UltraSPARC II processor (the Bagle machine). This extension allows for out-of-order execution, branch target speculation, and includes a memory hierarchy as well.

Several new variables are exposed to the user when working with MA models. The user can configure the width of the pipeline (the number of instructions allowed to fetch, execute, retire or commit in a single cycle), the size of the reorder buffer, whether instructions must retire in-order, and several memory hierarchy parameters related to OOE. The variables will be discussed as they are needed in the following lab sections.

When running in MA mode, Simics tracks dependencies of several varieties (register, control and memory) that exist in the instruction stream. It then places the instructions in a structure called an instruction tree — for the machine we are simulating, the instruction tree tracks the same state as the reorder buffer and associated structures would in an actual processor. This tree can be examined with the command `print-instruction-queue`. A load–store queue is also simulated.

The Simics microarchitecture simulator we will use in this lab speculates on branches by filling the instruction tree with instructions from both branch paths. Speculated instructions may only commit when their preceding branches have resolved. This behavior is notably different from the actual execution of many real out-of-order processors, but produces similar performance effects. The simulator we are using speculatively predicts branch target addresses.

The execution of instructions is divided into 6 stages (init, fetch, decode, execute, retire, commit) and instructions are only allowed to advance to the next stage when all applicable dependencies have been satisfied. In this way, instructions are allowed to execute out of order but may accumulate a multi-cycle delay appropriate to the underlying microarchitecture of the simulated processor.

A point worthy of note is that `steps` and `cycles` are no longer necessarily equivalent. A `step` occurs whenever an instruction commits. Advancing the simulation by one cycle may mean that multiple steps occur, or that none do. Similarly, advancing the simulation by one instruction may pause the simulation in the middle of a cycle. For this reason, it is advisable to use the `step-cycle` or `run-cycles` command to advance simulation, and then simply measure the number of steps that have occurred.

See the Simics MAI User Guide included with this lab for more information about any of these topics.

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 2.3: IPC statistics for each benchmark and answers
2. Problem 2.4: IPC statistics for all configurations and answers
3. Problem 2.5: IPC statistics for all memory configurations and answers
4. Problem 3.1/3.2/3.3 modifications and evaluations (include source code if required)

2 Directed Portion

2.1 General methodology

While you must ensure you are capturing a representative portion of the program's execution, you can measure instruction execution statistics whenever you like with `ptime` or logging.

For maximum efficiency, you should make sure that you are making use of all three operation modes of Simics. You only want to run in the slow, highly detailed modes when it is necessary to do so in order to collect accurate data.

The general methodology of this lab is:

1. Start Simics in fast mode, but use an MA-extended machine
2. Mount the host file system and load the appropriate files
3. Checkpoint the system
4. Restart Simics in stall mode and begin executing benchmark code to warm the caches
5. Checkpoint the system
6. Restart Simics in MA mode and collect OOE data

During this process Simics may report errors depending on which mode you are using and whether or not you are starting from a checkpointed simulation. If your simulation still runs after an error is reported, then simply disregard it.

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 Setup

Start Simics in `-fast` mode, using the `targets/sunfire/bagel-ma-common.simics` script. Make sure the host filesystem or workspace is mounted and that the benchmark binaries and 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, and probably a checkpoint at each of the initial magic breakpoints in the benchmark programs. Unfortunately, while the target machine being simulated is the same as in some previous labs, the underlying Simics modules used in the simulation are different (they use the MAI), so you will **not** be able to use checkpoints created in past labs.