CS152 Laboratory Exercise 2 (Version 0.1.1)

Professor: Krste Asanović
TA: Donggyu Kim and Howard Mao
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley

February 16, 2018

1 Introduction and goals

The goal of this laboratory assignment is to conduct memory hierarchy experiments by running realistic workloads on RocketChip [1]. To enable RTL-based simulation using FPGAs, we have already generated performance simulators by automatically transforming RocketChip [2, 3]. We will run these performance simulators in Amazon EC2 F1 instances (`https://aws.amazon.com/ec2/instance-types/f1`) to collect RocketChip's memory system stats for a subset of the SEPC2006int benchmark suite (`https://www.spec.org/cpu2006/`). You will also make architectural design decisions based on the results.

The lab has two sections, a directed portion and an open-ended portion. Everyone will perform 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.

For both the directed potion and the open-ended portion of this lab, you must work in *a group of two (not three)*. You are also encouraged to discuss solutions to the lab assignments with other groups, but must run through the lab by yourselves and turn in *a hard copy of your own lab report in the beginning of the lecture on March 5th.*

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 instructor or the TAs.

1.1 Graded Items

You will turn in a hard copy of your results to the instructor or TA. Please label each section of the results clearly. The directed items need to be turned in for evaluation. You only need to turn in *one* of the problems found in the open-ended portion.

1. (Directed) Section Problem 3.5: Analysis on cache statistics
2. (Directed) Section Problem 3.6: Performance prediction using cache statistics
3. (Open-ended) Coming soon
4. (Directed) Feedback on this lab

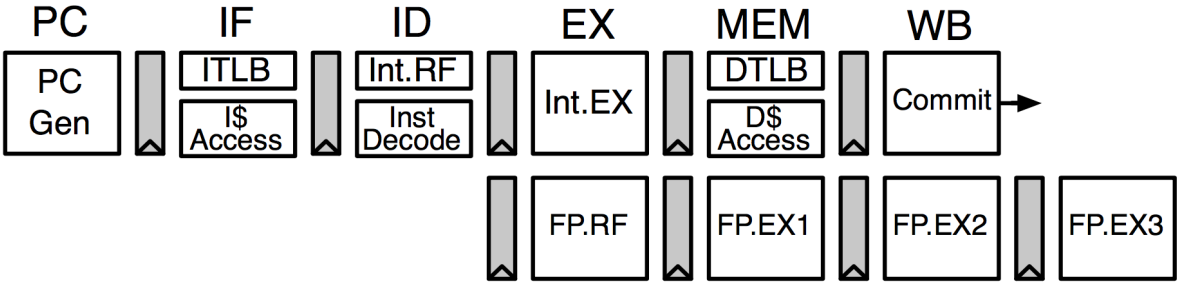

Figure 1: The Rocket Core Pipeline.

Lab reports must be in *readable* English and not raw dumps of log-files. It is *highly* recommended that your lab report be typed. Charts, tables, and figures - when appropriate - are great ways to succinctly summarize your data.

2 Background

2.1 Rocket Chip

RocketChip [1] is an open-source SoC generator suitable for research and industrial purposes. Rather than being a single instance of an SoC design, RocketChip is a hardware design generator, capable of producing many design instances from a single piece of Chisel [4] source code. Multiple industry products as well as silicon prototypes are manufactured using RocketChip. A RocketChip instance generally consists of three major components: processors, a cache hierarchy, and an uncore.

Rocket Chip instantiates an in-order processor, Rocket, by default, but also supports various core implementations. Rocket is a 5-stage in-order processor (Figure 1) that implements the RISC-V ISA [5, 6]. Its cache hierarchy includes L1 instruction caches, L1 blocking data caches, and fully associative L1 TLBs and a direct-mapped L2 TLB with configurable sizes, associativities, and replacement policies. In this lab, we provide pre-built FPGA images for various configurations. For now, RocketChip does not provide an L2 cache implementation yet, so we adopt an abstract L2 cache model instead [3]. This cache model is runtime configurable, so we do not need different FPGA images for different L2 cache parameters.

2.2 The SPEC CPU2006 Benchmark Suite

The SPEC CPU2006 benchmark suite (https://www.spec.org/cpu2006/) *was* widely used to evaluate real systems as well as design ideas in computer architecture research. This benchmark suite includes real-world application which execute *trillions of instructions* with their reference inputs. In this lab, we will only use a subset of benchmarks with their test inputs as follows:

- `400.perlbench`: cut-down version of Perl v5.8.7, the popular scripting language.
- `403.gcc`: C language compiler gcc version 3.2, which generates code for an AMD Opteron processor.

- `429.mcf`: Combinatorial optimization for single-depot vehicle scheduling in public mass transportation.

SPEC CPU2006 will retire soon. Why not SPEC CPU2017? (Un)fortunately, SPEC CPU2017 is more realistic, requiring longer execution times. We just wanted to save time and money. However, we selected benchmarks that also exist in SPEC CPU2017.

2.3 FPGA-based Performance Simulation with Amazon EC2 F1 Instances

We evaluated various pipelines for very small benchmarks using software-based performance simulators for Lab1. However, software-based simulation does not provide sufficient performance to evaluate complex hardware designs for realistic software applications.

Instead, any RTL designs can be directly mapped and emulated in the FPGA at speed. However, We need more accurate and runtime configurable timing models for memory systems and devices that will be implemented as an abstract RTL model or a software model. For this reason, RocketChip RTL implementations are automatically transformed and instrumented to generate performance simulators running in the FPGA [2, 3], enabling efficient communications between FPGA and software.

There are increasing interests in using FPGAs for application-specific accelerators. As a result, cloud service providers decided to offer FPGA cloud instances such as Amazon EC2 F1 instances. We use these services to quickly evaluate real-world hardware designs for real-world software applications.

We pre-built FPGA-based performance simulators for various cache parameters and provide them as Amazon FPGA images (AFIs) that can be loaded into any F1 instances. We also provide the Amazon machine image (AMI) that contains necessary software binaries and scripts to run simulations.

3 Directed Portion (30%)

3.1 Launching an F1 Instance

We will launch an F1 instance to start this lab. First, log in to an instructional machine (`icluster{6-9}.eecs.berkeley.edu`). Then, to enable commands to control your F1 instances, run:

```
# You can add this line in ~/.bash_profile
inst$ source ~cs152/sp18/cs152.lab2.bashrc
```

Next, let's launch an F1 instance:

```
inst$ launch-f1 | tee <file name>
wait until running
running now!
Instance ID: i-07ab2ee7a526ccacb
IP Address : 34.227.49.244
Keep Instance ID & IP Address
Please shut the instance down in 8 hours.
```

```
wait for initialization
initializing
...
initializing
ok
It's time to SSH into your instance
```

It will take for a while (3 min) for initialization, so please be patient. Also, note that you will need the instance id and the IP address across this lab. Most importantly, **each student** is allowed to launch **single instance at a time** as many times as possible, but the total instance-hours should not exceed 40 hours. (Therefore, *80 hours are allowed for each group in total.*) If you violate any rule, you will get a severely penalty. (50 % with the first violation and 100 % with the second violation. Therefore, you don't have to do this lab if you violated the rules twice.)

3.2 Linux Boot and Hello World

Now, let's SSH into the F1 instance:

```
inst$ ssh centos@<IP Address>
```

Once you SSH into your F1 instance, move into `cs152-lab2`:

```
$ cd cs152-lab2
```

This directory contains necessary files to conduct this lab. First of all, we should load an AFI image:

```
# 16KiB L1$, L1 TLB reach = 128KiB, L2 TLB reach = 4MiB
$ ./load-fpga.sh agfi-0aa6f0423f0ff7843
```

This will load a RocketChip simulator for 16KiB L1 I/D caches, fully-associative L1 I/DTLBs with 32 entries, direct-mapped L2 TLB with 1024 entries.

We provide you a make file command to conveniently run RISC-V binary images:

```
# Default argument values
# L1_SIZE ?= 16KB (design parameter)
# L1_TLB_REACH ?= 128KB (design parameter)
# L2_TLB_REACH ?= 4MB (design parameter)
# L2_WAY_BITS ?= 2 (2^2 = 4 ways, runtime parameter)
# L2_SET_BITS ?= 12 (2^12 = 4096 sets, runtime parameter)
# L2_BLOCK_BITS ?= 6 (2^6 = 64 Bytes, runtime parameter)
$ make BIN=<RISCV binary image>
```

This command creates a directory named `output/<cache parameters>`, runs the image in the simulation, pipes `stdout` and `stderr` to `output/<cache parameters>/<RISCV binary image>.{out, err}`, and dumps the cache (and branch) statistics to `output/<cache parameters>/<RISCV binary image>.stat`. (Read `Makefile` for more information.) This will be useful when you run simulations for various cache configurations.

Now, let's boot Linux and print hello world in the simulator. Run:

AFI	L1 $ assoc.	L1 $ set size	L1 $ block size	L1 TLB reach	L2 TLB reach
agfi-09161206020478060	4	32	64 bytes	128 KiB	4MiB
agfi-0aa6f0423f0ff7843	4	64	64 bytes	128 KiB	4MiB
agfi-04da881d8fea1585c	8	64	64 bytes	128 KiB	4MiB
agfi-0aa6f0423f0ff7843	4	64	64 bytes	128 KiB	1MiB
agfi-09dfe6ac51b81c7f6	4	64	64 bytes	128 KiB	No L2 TLB
agfi-0815d9f8a9ed5ef16	4	64	64 bytes	32 KiB	No L2 TLB

Table 1: AFIs for various cache configurations

```
$ make BIN=images/bblvmlinux-hello
```

You can see linux boot messages and `Hello CS152!` in the screen. It also prints performance counter values, which are also saved in `outputs/16KiB-128KiB-4MiB-2-12-6/bblvmlinux-hello.stat`:

```
##  cycles              = 9522836
##  instret             = 5002693
##  loads               = 253655
##  stores              = 151974
##  L1 I$ misses        = 22999
##  L1 D$ misses        = 29558
##  L2$ misses          = 9233
##  ITLB misses         = 606
##  DTLB misses         = 1204
##  L2 TLB misses       = 1596
##  branches            = 144598
##  branche mispredicts = 27078
```

Can you compute the CPI and the misses per kilo instructions (MPKIs) of miss events for `bblvmlinux-hello`?

Warning: `make claen` will delete *all* generated output files. So, ask yourself carefully if you really want to delete all the files before you run into a disaster.

3.3 Cache Parameter Sweep for SPEC CPU2006

Note: do not execute the script with Version 0.1. But, this will be available very soon after the sections.

We will collect cache and branch stats for SPEC CPU2006 by running `images/bblvmlinux-{400.perlbench, 403.gcc, 429.mcf}`. Note that these benchmarks run on top of Linux where TLBs play an important role for system performance.

To automate simulation runs for various cache configurations, we will run a script, `sweep.py`. This scripts loads AFI images for each cache configuration and runs simulations for each benchmark sequentially by varying L2 cache parameters. Please take a look and figure out how this script runs simulations using the makefile command. Table 1 shows AFIs for different cache configurations.

Once you understand what is going to happen, it is time to run the script to sweep cache parameters:

```
# To prevent simulations gone while you are out
$ tmux
$ ./sweep.py
```

It will take for a while. You can hang out while the machine is working, but make sure you come back on time. (Otherwise, you will blow credits out and have Donggyu go bankrupt.) If you want to return to the screen, run in an F1 instance:

```
$ tmux a
```

3.4 Terminating Your Instance

Once, you finish the experiments. Copy the output files to the instructional machine as follows:

```
inst$ scp -r centos@<IP Address>:~/cs152-lab2/outputs ./
```

You will have the whole output files in the current directory. Next, shut down your instance with the following command in **an instructional machine**:

```
inst$ terminate-f1 <Instance Id>
```

Once again, you should terminate an instance on time to avoide a (brutal) penalty.

3.5 Analysis on Cache Statistics

3.6 Performance Prediction with Cache Statistics

4 Open-ended Portion (70%)

Stay tuned. We will let you know through Piazza when it's ready.

References

[1] K. Asanović, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, D. Dabbelt, J. Hauser, A. Izraelevitz, S. Karandikar, B. Keller, D. Kim, J. Koenig, Y. Lee, E. Love, M. Maas, A. Magyar, H. Mao, M. Moreto, A. Ou, D. A. Patterson, B. Richards, C. Schmidt, S. Twigg, H. Vo, and A. Waterman, "The Rocket Chip Generator," Tech. Rep. UCB/EECS-2016-17, EECS Department, University of California, Berkeley, apr 2016.

[2] D. Kim, A. Izraelevitz, C. Celio, H. Kim, B. Zimmer, Y. Lee, J. Bachrach, and K. Asanović, "Strober : Fast and Accurate Sample-Based Energy Simulation for Arbitrary RTL," in *ISCA*, 2016.

[3] D. Kim, C. Celio, D. Biancolin, J. Bachrach, and K. Asanović, "Evaluation of RISC-V RTL with FPGA-Accelerated Simulation," in *First Workshop on Computer Architecture Research with RISC-V*, 2017.

[4] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avižienis, J. Wawrzynek, and K. Asanović, "Chisel: constructing hardware in a scala embedded language," in *DAC*, 2012.

[5] A. Waterman, Y. Lee, D. Patterson, and K. Asanovi, "The RISC-V Instruction Set Manual: User-level ISA Version 2.1," Tech. Rep. UCB/EECS-2016-118, 2016.

[6] A. Waterman, Y. Lee, D. Patterson, and K. Asanovi, "The RISC-V Instruction Set Manual Volume II: Privileged Architecture Version 1.10," tech. rep., 2017.