C152 Laboratory Exercise 4

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

March 28, 2011

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 Simics models of VLIW and multithreaded machines, 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.

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 Overview of the new machines

For this lab you will make use of two new target machines that Simics is capable of simulating. These machines are not as fully featured as the machines we have been using in previous sections, making complicated studies more difficult to conduct. However, this lab will serve as an introduction to the machines.

The new first machine is an Intel Itanium processor called Vasa running Red Hat Linux. This processor uses the IA-64 architecture (VLIW). It is dependent on the compiler to statically exploit ILP by packaging instructions into 128-bit bundles.

The second new machine is a Sun Microsystems UltraSPARC T1 processor called Niagara-Simple running Solaris. This processor uses the SPARCv9 architecture. It has 8 cores, each of which has 4 hardware thread contexts that appear to the OS as separate cores. This machine employs fine–grained multithreading, meaning that each core switches between one of the available threads on every cycle. Available configurations of this machine in Simics have 1 (1x1), 2 (2x1), or 32 (8x4) thread contexts.

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. Make sure you name your open–ended section partners in your individual portion, and that the group results name you as a participating member. The following items need to be turned in for evaluation:

1. Problem 2.1: VLIW statistics for each benchmark and answers
2. Problem 2.2: Loop unrolling statistics and answers
3. Problem 2.3: Niagara statistics and answers
4. Problem 3.1/3.2 statistics and evaluations (include source code if required)

2 Directed Portion

2.1 Exploring VLIW assembly

The Intel Itanium processor target that we will use to study VLIW performance uses the IA-64 instruction set. The format of this bundle is displayed in Figure 1. Each bundle contains 3 41-bit instructions and a 5-bit template field that specifies the grouping of this bundle with adjacent bundles. Groups contain instructions that can execute in parallel. The first instruction in a bundle is said to be in slot 0 of the bundle, the second instruction in slot 1, and the last instruction in slot 2. The Simics `step-*` commands execute one slot at a time from a bundle.

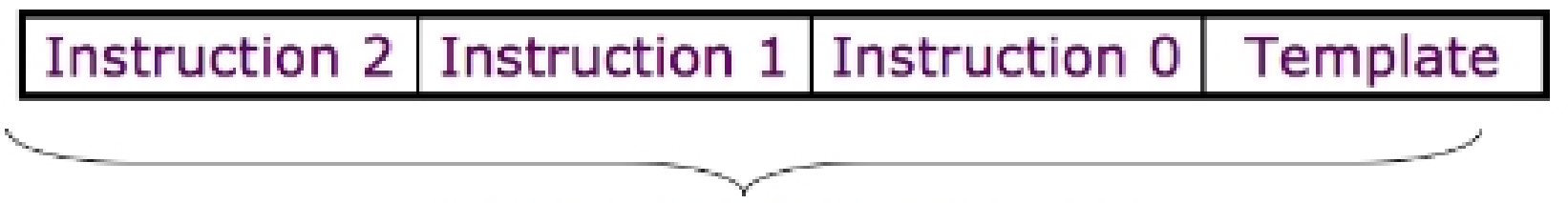

Figure 1: IA-64 instruction bundle

Simics is normally set up to process and report one instruction per processor at a time, so the way it disassembles instruction bundles is worth explaining. Since individual instructions do not have well-defined addresses, Simics uses the encoding scheme (bundle address + slot number) when dissassembling instructions. Since bundle addresses are always 16-byte aligned, the lower 4 bits in the bundle address are always zero and can be used to encode the slot number. For example, the following instructions are in the three slots of the bundle located at address `0xe000000004497f30`:

```
[cpu0] v:0xe0000000004497f30 p:0x0000000004497f30        ld8.acq r14 = [r15] ;;
[cpu0] v:0xe0000000004497f31 p:0x0000000004497f31        cmp.eq p6, p7 = 0, r14
[cpu0] v:0xe0000000004497f32 p:0x0000000004497f32        nop.i 0x0
```

The Linux disk image used by Vasa is rather sparse in terms of available features. It lacks the software that normally allows us to mount the host machine's file system on the target machine's file system. This means the the only way to transfer files between the machines is to create ISO images and mount them on the target machine's simulated cdrom drive.

More importantly, there is no compiler installed on the simulated machine. Since CS152 students don't have access to the department's Itanium cluster, we could not compile IA-64 binaries from source even if we could copy over the source files. For these reasons, in this version of the lab we will only ask you to examine code from several programs already installed on the target machine.

First, boot up the machine: `host$ ./simics targets/ia64-460gx/vasa-common.simics` The machine will take a few minutes to boot up. You may create a checkpoint if you wish, though you will probably not need it. Sometimes the X11 terminal of the simulated machine goes blank, but nothing is wrong and if you type the screen will refresh.

Run the following three commands one at a time on the target machine. For each command, once it begins executing quickly return to the Simics command line and pause execution with `control-C`. While this method is imprecise, it is acceptable for the analysis you will do in this section. The commands to run the mini-benchmarks are:

```
target# bzip2 /var/log/dmesg
target# yes
target# grep Swap /var/log/dmesg
```

Step through the code 30 'instructions' (i.e. 10 bundles) at a time with the command `simics> si 30`. Do this for at least 50 bundles. For each of the mini-benchmarks, report on:

1. The incidence of bundles that fill all/two/one/no slots with useful instructions.

2. The theoretical IPC of this machine assuming two bundles are fetched and executed per cycle (i.e. the way a real Itanium operates).

3. Whether there seems to be any correlation between the presence of noops in the bundles and the presence of any other instruction types. Give examples of code sequences you observe.

2.2 Investigating the effects of loop unrolling

In this section you will investigate the compiler's effectiveness at loop unrolling. We are unable to use the VLIW Itanium machine since it lacks a compiler, so instead you will compile and run code on the Bagle machine we have used in previous labs.

1. The code in the file `loop_unrolling.c` is a simple scalar–vector addition loop. After booting the machine, copy the source code into the target machine's file system. Compile the code:
   ```
   target#gcc -I/host/share/instsww/pkg/virtutech/simics-3.0.30/src/include
    -O3 loop_unrolling.c -o unroll
   ```

2. Enable magic break points and begin executing the loop benchmark.