EEL4713 Assignment #6
Spring 2012
Assigned: 4/12/12
Demo: May 2, 2012 in class during schedule final time 7:30-9:30 am
Due: 5/4/12 (Note this is Friday!) @ 11:55pm Via Sakai

Section 1: Setup
Download the latest version of the Grid appliance from http://www.gridappliance.org. Also, install the Cacti and SESC simulators by following the instructions in the "Archer" project Wiki: go to http://archerproject.org, follow the "Educational modules" link, then the link "An assignment on cache modeling with Cacti and SESC".

Section 2: Textbook Questions
Questions from chapter 6: 6.3.1-2, 6.15.1-4, 6.18.1-2

Section 3: Laboratory

3.1. Introduction
Today's computers (including memory hierarchies) are designed with the aid of extensive simulations that provide quantitative data to justify design choices. In this problem, you will use a cache hierarchy simulator to investigate tradeoffs in the design of a level-1 cache. In this problem, you will use the "SESC" and "cacti" tools available for execution from the Grid appliance. SESC is a microprocessor simulator (that also simulates the behavior of two-level caches), while Cacti is a program that calculates the speed of caches based on parameters such as size and associativity. Note that even though you are not using the latest version of Cacti, the basic fundamentals you will learn do not changed based on the version of Cacti you use. In this assignment you will consider a base configuration with two levels of caches: the L1 cache is split into instruction+data caches; the base configuration has both I-cache and D-cache configured with the same parameters: 8Kbytes, 32-byte blocks, direct-mapped. Use default parameters for the L2 cache.

Note: each SESC simulation may take 10 or more minutes, so start working on this assignment early! The Grid appliance allows you to run multiple simulations concurrently on remote computers, so you can have more than one simulation running at the same time.

Warming up Cacti 3.2 simulator: Cacti is a tool that helps designers determine cache access times, power/energy consumption, among other parameters based on relevant design points entered as inputs to the simulator. It is a fast simulator that uses analytical models to estimate its outputs. Let's use the Cacti tool to determine the access time of the L1 cache configured as above as an example. Use a 0.09µm (90nm) technology. The command for a Cacti simulation is:

```
cd cacti
./cacti C B A TECH Nsubbanks
```

Where C is the cache size, B is the block size, A is the associativity (e.g., 1=direct-mapped), TECH is the transistor technology, and Nsubbanks is the number of sub-banks (a .pdf document with the full description of the CACTI simulator version 3.2 is included). For example, typing the command "./cacti 8192 32 1 0.09 1 > cacti.out" writes the output of cacti to cacti.out, and the command "grep Access cacti.out" filters the output cacti.out for lines containing the string "Access". Before moving to the next step, double-check that the access time you obtain is approximately 479ps. Assume a processor clock rate of 3.252GHz throughout this lab (the cache above would have a 2-cycle latency).

Warming up SESC simulator: SESC (sourceforge.net/projects/sesc) is an open-source execution-driven simulator of superscalar and multi-core processors. It takes as inputs a MIPS executable and a configuration file, which determines the computer architecture parameters to be used in the simulation (e.g., cache size, block size, number of processors). It simulates the execution of the MIPS executable instruction by instruction, and at the end of simulation, provides a summary of various simulated

parameters (e.g., cache miss rate). Because it simulates every instruction at great detail, SESC simulations can take a very long time to finish. For many benchmarks, simulation times of hours/days are not uncommon. For this assignment, the simulations have been chosen to take a few minutes.

To run your first SESC simulation type:

```
cd sesc_demo
./sesc.smp –itest.in –csesc32.conf crafty.mipseb
```

This command runs a simulation of the application "crafty.mipseb" (a chess simulator benchmark from SPEC CPU 2000, http://www.spec.org/cpu/CINT2000/186.crafty/docs/186.crafty.html), with input file test.in, and SESC configuration parameters stored in sesc32.conf. As the simulation progresses, you will see outputs from the crafty application coming out line by line in the X terminal. At the end of the simulation, a file named "sesc_crafty.mipseb.XYZABC" will be created (XYZABC is replaced by a random string). This output file summarizes all the inputs that were configured in the sesc32.conf file (see below) as well as shows summary results from the simulation. For example, if you type:

```
grep DL1:readMiss sesc_crafty*
grep IL1:readMiss sesc_crafty*
grep DL1:readHit sesc_crafty*
grep IL1:readHit sesc_crafty*
```

You will see the total number of cache hits and misses for the Iand Dcaches, for processors P(0) and P(1). You only need to care about processor P(0) in this assignment. Typing "grep clockTicks sesc_crafty*" shows the total number of simulated clock cycles. In this example, the simulation covers about 113E6 clock cycles.

Note: the main file of interest to you in this assignment is sesc32.conf. Check out the file sesc32.conf to help understand what the configuration has been set for this simulation, for example:

```
cacheLineSize : the block size (32 Bytes in this file)
frequency: processor frequency (5GHz in this file)
[IMemory]: parameters for L1 instruction cache
32Kbytes
2-way associative
Write-through (WT)
LRU replacement
2 ports
2-cycle hit delay
[DMemory]: parameters for L1 data cache
```

Condor simulation: the sesc configuration file is the only file you will need to configure in this assignment. You can create multiple configuration files with different names for all the simulations you will perform. You can then "batch" their execution through Condor in the Grid appliance. Check out the submit_sesc.condor file for an example on how to submit three jobs in parallel to Condor. The only difference among these sample jobs is the cache line size (32 Bytes in sesc32.conf, 64 and 128 in sesc64.conf and sesc128.conf). Feel free to use Condor or run simulations in your own machine for this assignment. For Condor to work, you need to be connected to the Internet and check that condor_status is responding correctly.

3.2. Block size tradeoffs

In this problem you will simulate caches with different block sizes to investigate tradeoffs between this cache parameter and performance.

a) Use SESC to simulate the system with the base cache configuration described above, and seven

other configurations with larger block sizes in the L1 instruction cache. Start with the file sesc32.conf; if needed, adjust the block size of the L2 cache in your simulations to be equal to or larger than the L1 block size. Make sure that for each cache configuration you select you simulate its access time using Cacti and take that into consideration when entering a value for the cache hit latency in SESC. Assume the hit latency is the ratio of cache access time to clock cycle, rounded up to an integer value. Plot two graphs: L1 I-cache miss rate (il1_miss_rate in the SESC simulation output) versus block size, and total execution time (sim_cycle) versus block size. Discuss your results with meaningful discussion (i.e. do not simply state what can be read off of the graphs themselves).

b) Repeat part a), but now changing the block size of the L1 data cache. Plot the same two graphs. Discuss your results, comparing to the results obtained in a).

3.3. Associativity tradeoffs

In this problem you will simulate caches with different associativities to investigate tradeoffs between this cache parameter and performance. Use SESC to simulate the system with the base cache configuration described above, and five other configurations with increased associativity in the L1 data cache. Make sure that for each cache configuration you select you simulate its access time using Cacti and take that into consideration when entering a value for the cache hit latency in SESC. Plot two graphs: L1 D-cache miss rate versus associativity, and total execution time (clock cycles) versus cache associativity. Discuss your results.

3.4. Cache size tradeoffs

In this problem you will simulate caches with different sizes to investigate tradeoffs between this cache parameter and performance. Use SESC to simulate the system with the base cache configuration described above, and five other configurations with increased size in the L1 instruction cache. Make sure that for each cache configuration you select you simulate its access time using Cacti and take that into consideration when entering a value for the cache hit latency in SESC. Plot two graphs: L1 I-cache miss rate versus associativity, and total execution time (clock cycles) versus cache size. Discuss your results.

3.5. Overall cache performance experiment

Given your results from 3.2-3.5, select three candidate cache configurations that you believe will show improved performance over the base configuration when you combine improvements in size, associativity and block size. Briefly explain the reasoning behind your selections. Obtain access times for these configurations using Cacti and simulate them using SESC. Summarize the relevant results from your simulation. Which cache configuration, among those you selected, yields the best performance (i.e. lowest execution time)?

3.6. Implementing a Simple Data Cache

For the final part of this laboratory you will be modifying the data memory of your pipelined processor from Assignment 5 in order to implement a simple data cache. You will be implementing an 8 block two-way set associative cache with a block size of 16 bytes, a random block replacement strategy, a write through write strategy, and a nowrite allocate strategy to deal with write misses. Obviously your cache hit time will be one clk cycle and due to the fact that your data memory bandwidth is 4 bytes/clk your miss penalty will be at the very least 4 clk cycles (there are other design issues that may cause you to increase this penalty). Read Chapter 5 in your textbook to see how this should be implemented.

Separate from your design create a counter that will count the number of read hits and misses that occur in the execution of a program. For your report, include a detailed section where you go through the design process (this should include a hand drawn schematic of your cache hardware). Include a section where you decode the lab6demo programs into MIPS assembly code along side the functional simulation output (in table form) from your processor. Make sure you annotate your output to show that your processor executed the program correctly. Make sure to include VHDL code for newly created components. Your report should still follow the report guidelines discussed in assignment

Note: in addition to submitting your reports and your design files electronically via Sakai, you must

demonstrate your components functionality in Lab on the DEMO date listed on the first page of this Assignment. Failure to submit your design files will cause you to receive a zero on the design sections of the Assignment. Please make sure that you only send your VHDL code, BDF files, mif files, and/or symbol files and don't send any other files generated by Quartus.