CS/ECE 561: Hardware/Software Design of Embedded Systems
Fall 2015

Homework 2: Architectural Exploration with gem5

Assigned: 22 September 2015
Due: 6 October 2015

Instructions:

- Please submit your solutions via RamCT. Submissions should include source code files or text/word/pdf files in a single zip file, with separate folders designated for every problem.
- Some questions might not have a clearly correct or wrong answer. In such cases, grading is based on your arguments and reasoning for arriving at a solution.

Introduction

gem5 is a modular discrete event computer system simulator platform. This means that:

1. gem5's components (processors, memories, networks, controllers) can be rearranged, parameterized, extended or replaced easily to suit your exploration needs.

2. It simulates the passing of time as a series of discrete events.

3. Its intended use is to simulate one or more computer systems in various ways.

4. It's more than just a simulator; it's a simulator platform that lets you use as many of its premade components as you want, to build up your own simulation system.

gem5 is written primarily in C++ and python and most components are provided under a BSD style license. It can simulate a complete system with devices and an operating system in full system mode (FS mode), or user space only programs where system services are provided directly by the simulator in syscall emulation mode (SE mode). There are varying levels of support for executing Alpha, ARM, MIPS, Power, SPARC, and 64 bit x86 application binaries on CPU models including two simple single CPI models, an out of order model, and an in order pipelined model. A memory system can be flexibly built out of caches and interconnects. Recently the Ruby simulator has been integrated with gem5 to provide even better, more flexible memory system modeling.

This assignment will provide you with hands on experience working on gem5 to perform architectural exploration for embedded processor/memory subsystems. We will primarily be focusing on the SE mode, and work with the Alpha and ARM processors. The following pages will first go through the process of helping you to set up the environment, before presenting you with problems to solve.

USEFUL LINKS:

Home-page for gem5: http://www.m5sim.org/

gem5 Documentation: http://www.m5sim.org/Documentation

Installing gem5 platform with ALPHA processors:

These steps are adapted (for ALPHA) from the video: http://www.m5sim.org/Introduction. You must perform the following installation on your own linux environment rather than on Engineering servers. The instructions below have been tested and validated on Ubuntu 14.04.

cd into your home directory and first install the supplementary tools needed by gem5

> sudo apt-get install mercurial scons gcc m4 swig python python-dev libgoogle-perftools-dev g++ libpcre3-dev

Then get gem5 and build the simulation environment for ALPHA processors (Note: the build will take a while):

> hg clone http://repo.gem5.org/gem5-stable

> ls (you should have a gem5-stable directory if everything went well. Rename the directory to gem5)
> cd gem5
> scons build/ALPHA/gem5.opt

Now to test the installation, run the 'hello world' program on the built ALPHA platform:

> cd gem5
> build/ALPHA/gem5.opt configs/example/se.py -c tests/test-progs/hello/bin/alpha/linux/hello

NOTE: pre-compiled binary hello already exists at location gem5/tests/test-progs/hello/bin/alpha/linux/

If the simulation completed correctly, you should see a "Hello World!" somewhere in the output. The file stats.out in the m5out directory contains details of your simulation, such as number of instructions executed, cache misses, etc.

>gedit m5out/stats.txt &

Downloading and running SPLASH2 benchmark suite

We will be using workloads from the SPLASH2 benchmark suite to run simulations on our ALPHA platform. The necessary code (with pre-compiled binaries) for this purpose is available on the gem5 website. Download the file: http://www.gem5.org/dist/m5_benchmarks/v1-splash-alpha.tgz De-compress the file. Copy the folders paramacs.upc.3 and splash2 and paste them in your gem5 directory. . Test the simulator with SPLASH-2 benchmarks. One example is the following:

> build/ALPHA/gem5.opt configs/example/se.py --cmd=splash2/codes/kernels/lu/contiguous_blocks/LU

The simulation will take some time but once it eventually finishes you can look at the stats.txt file for simulation statistics. Some benchmarks must be simulated with inputs or command line options. Here are two examples of this (in the second case replace "/home/john/…" with the absolute path to the input.2048 file in your environment):

> build/ALPHA/gem5.opt configs/example/se.py --cmd=splash2/codes/kernels/cholesky/CHOLESKY -i splash2/codes/kernels/cholesky/inputs/d750.O

> build/ALPHA/gem5.opt configs/example/se.py --cmd=/splash2/codes/apps/fmm/FMM --input="/home/john/work/gem5/splash2/codes/apps/fmm/inputs/input.2048" -o "-o"

You can learn more about splash2 benchmarks and their usage from the C source files and README files in the directories for various apps and kernels in SPLASH-2. You should also look up options to configure the simulation (e.g., simulate for a fixed number of instructions, change cache size, etc) in the python file configs/common/Options.py and read up on the gem5 documentation at http://www.m5sim.org/Documentation. Note: do not alter the Options.py file. Rather, just make note of the options in the file, which you can use on the command line (with your own unique values) when running your simulations.

Using McPAT for area and power analysis

The stats.out file provides performance statistics for a benchmark running on a specific configuration of a processor core. McPAT is a tool that takes the output provided by gem5 as an input and provides detailed power and area information. This is very useful during architectural exploration. For example, the tool can be used when you change the cache configuration by increasing cache size, and would like to find out not just the performance benefits of your modification, but also the power and area overhead as a result of the additional cache memory you added.

Download McPAT version 0.8 from http://www.hpl.hp.com/research/mcpat/, extract the gzipped tar file (tar -xzvf mcpat*.tar.gz), and run the following commands to create the tool.

> sudo apt-get install gcc-4.8-multilib g++-4.8-multilib

> make

Also download and extract mcpat_scripts_new.zip into the mcpat directory and extract it. The way to use this tool is to first run and complete a gem5 simulation so that a stats.txt is created. Then use the perl script m5-mcpat.pl to take the output from gem5 and create an xml file that is the input for mcpat. Make sure that you copy the m5-mcpat.pl and **mcpat-template-alpha.xml files from mcpat_scripts_new directory to the mcpat directory**. Then use the following command (assumes that you have created the mcpat directory inside the gem5 directory):

> perl m5-mcpat.pl ../m5out/stats.txt ../m5out/config.ini mcpat-template-alpha.xml > output.xml

Once this xml file has been created, run the mcpat tool as follows:

> perl mcpat-exec.pl output.xml (If you get any errors, check the note below)

After some time, a power and area breakdown of your processor while running a particular benchmark is dumped by the tool to the output.

Note: For McPAT to work, you must simulate gem5 using the following options: --cpu-type="detailed", --caches, and --l2cache. For example the following command executes the RADIX benchmark for a duration of 1 million instructions and generates an output that is usable by mcpat:

> build/ALPHA/gem5.opt configs/example/se.py -I 1000000 --cpu-type="detailed" --caches --l2cache --cmd=splash2/codes/kernels/radix/RADIX

Q1 [150 points]. In this problem you will explore the cache hierarchy in the ALPHA processor using gem5. Consider a baseline 2 GHz ALPHA processor subsystem running the RADIX benchmark, with an L1 data cache size of 4 kB and associativity of 2, L1 instruction cache size of 4 kB and associativity of 2, and a unified L2 cache size of 128 kB with associativity of 4. The following command would simulate such a system for 5 million instructions (refer to Options.py to see other possible options you can use during gem5 simulation; many parameters take default values if not specified during simulation – e.g., clock frequency by default is 1 GHz):

> build/ALPHA/gem5.opt configs/example/se.py -I 5000000 --cpu-type="detailed" --sys-clock=2GHz --cpu-clock=2GHz --caches --l2cache --l1d_size=4kB --l1i_size=4kB --l2_size=128kB --l1d_assoc=2 --l1i_assoc=2 --l2_assoc=4 --cmd=splash2/codes/kernels/radix/RADIX

(a) Suppose you are allowed to increase the total L1 cache size (instruction+data cache size) to a maximum of 64kB and L2 cache size to a maximum of 4MB. You are also allowed to increase maximum L1 associativity to 4 and maximum L2 associativity to 8. Find the cache configuration that gives the best performance results (i.e., smallest execution time represented by system.cpu.numCycles in the stats.txt file) with the minimum overall cache size and associativity. Mention the cache size and associativity values (for L1 data, L1 instruction, and L2) of your best configuration and improvement in execution time over the baseline case configuration shown in the example for this question.

(b) Find the area and power dissipation (total leakage + runtime dynamic power) of the processor core (core + L1 caches) and L2 cache for your best configuration and compare it with the baseline configuration. The power data you obtain from mcpat is for the 65nm library. How does this power change for 45nm and 32nm technology nodes? Mention any trends you notice for leakage and dynamic power, and area (Hint: you need to change technology node parameter in the xml template file to recalculate results for different technology nodes).

Q2 [100 points]. In this problem you will work with the ARM A9 processor subsystem, that has been designed for a small form factor and low power embedded processing. The processor does not have many of the functional units as the ALPHA processor does but due to its simple design can still operate at 2GHz. Start by building the ARM simulator, similar to the ALPHA build (> scons build/ARM/gem5.opt).

Now you must compile the SPLASH-2 benchmarks for the ARM processor, so you can execute ARM binaries of the benchmarks on the ARM simulator. The following command will download and install the ARM cross-compiler on your linux system.

> sudo apt-get install gcc-arm-linux-gnueabi

Let us cross-compile the radix benchmark. Download splash2-modified.tgz and go to the splash2-orig/codes/kernels/radix directory and edit the makefile. You need to use the gcc ARM cross compiler to create ARM compatible binaries instead of your native gcc compiler (that creates x86 binaries). Therefore modify the following variable in the makefile:

CC = arm-linux-gnueabi-gcc

Save and make the ARM binary for RADIX (go to splash2-orig/codes/kernels/radix, change the CC field in Makefile, and build it). Execute it on the gem5 ARM simulator for the baseline configuration as you did in the previous question. This is how you can cross-compile for the ARM processor.

(a) Cross compile the Raytrace application and the FFT kernel. If the ARM processor does not have an L2 cache and only has a budget of 32kB that must be divided up between the L1 instruction and data caches, what cache configuration provides the best performance for each of the benchmarks, assuming you must execute 10 million instructions? Determine the L1 data and instruction cache sizes and associativity of the caches for your best configurations.

Note: Here is an example of the command line to execute the FFT kernel:

> build/ARM/gem5.opt configs/example/se.py --cpu-type="detailed" --sys-clock=2GHz --cpu-clock=2GHz --caches --l1d_size=16kB --l1i_size=16kB --l1d_assoc=2 --l1i_assoc=2 --cmd=splash2-orig/codes/kernels/fft/FFT -o “-m8 -o” -I 10000000