ECE 452: Computer Organization and Design

Spring 2017

Homework 6: Architectural Exploration with Sniper

Assigned: 25 Apr 2017
Due: 4 May 2017

Instructions:

- Please submit your assignment solutions (no code needs to be attached) via Canvas in a word or pdf file.
- 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

Sniper is a next generation parallel, high-speed and accurate x86 simulator. This multi-core simulator is based on the interval core model and the Graphite simulation infrastructure, allowing for fast and accurate simulation and for trading off simulation speed for accuracy to allow a range of flexible simulation options when exploring different homogeneous and heterogeneous multi-core architectures. The Sniper simulator allows one to perform timing simulations for both multi-program workloads (multiple applications running on different cores) and multi-threaded, shared-memory applications (one multi-threaded application running on multiple cores) with 10 to 100+ cores, at a high speed when compared to existing simulators.

This assignment will provide you with hands on experience working on Sniper to perform architectural exploration for processor/memory subsystems. 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 for More Information:

Home-page for Sniper: http://www.snipersim.org/

Sniper Manual: http://snipersim.com/documents/sniper-manual.pdf

Installing Sniper platform with SPLASH-2 benchmarks:

The following steps have been tested on Ubuntu 14.04 64-bit machine (www.ubuntu.com/download/desktop) and Linux Servers of ENS (www.engr.colostate.edu/ens/how/connect/servers.html#linuxcs). Some specific instructions for ENS servers are written in red text. It is highly recommended that you perform the installation on your own Ubuntu Linux environment rather than on ENS servers.

Download and extract Sniper package to your home directory (or directory of your choice, but note that instructions assume you have extracted Sniper to your home directory).

```
cd ~
wget  http://snipersim.org/download/a8daff6a4e1f3841/packages/sniper-6.0.tgz
tar -xvzf sniper-6.0.tgz
cd sniper-6.0
```

Download and extract Pin front-end.

```
> mkdir pin_kit
> cd pin_kit
> wget http://software.intel.com/sites/landingpage/pintool/downloads/pin-2.14-67254-gcc.4.4.7-linux.tar.gz
> tar -xvzf pin-2.14-67254-gcc.4.4.7-linux.tar.gz --strip 1
```

Place two of the patch files attached with this assignment, *sniper-pinplay-controller-67254.patch* and *xed_format_fix_pin_67254.patch*, into sniper-6.0 directory. Apply patches to resolve compatibility issue between Sniper and Pin front-end.

```
> cd ~/sniper-6.0
> patch -p 1 <sniper-pinplay-controller-67254.patch
> patch -p 1 <xed_format_fix_pin_67254.patch
```

On ENS servers use scp command to copy patches from any computer:

```
sudo scp /path/to/sniper_assignment/*.patch ens-username@ens-computer-name.engr.colostate.edu:path/to/sniper-6.0/in/ens/server
```

Compile Sniper. Some extra dependencies (a pre-compiled copy of the Python interpreter environment) will be downloaded automatically, so make sure you have a working internet connection when you make Sniper for the first time.

For personal computers, you may need to install additional packages before compiling.

```
sudo apt-get install libbz2-dev libboost-dev libsqlite3-dev gfortran gnuplot
```

If you are using Ubuntu 15 and above, install additional gcc components shown below

```
sudo apt-get install gcc-4.8 g++-4.8
```

Here is how to compile:

```
make
```

If you are using Ubuntu 15 and above, set the following before the 'make' command

```
CC=/usr/bin/gcc-4.8 CXX=/usr/bin/g++-4.8 make
```

Download and extract SPLASH2 benchmark suite.

```
wget http://snipersim.org/packages/sniper-benchmarks.tbz
tar -xvjf sniper-benchmarks.tbz
cd benchmarks
```

Open "Makefile" and comment out benchmarks that will not be used in this assignment, as shown in following snippet (On ENS servers you can use vim to edit Makefile):

```
gedit Makefile &
```

The snippet with only splash-2 and hooks enabled is as shown:

```
-----------------------
all: dependencies
	make -C tools/hooks
	make -C splash2
#	make -C parsec
#	make -C cpu2006
#	make -C npb
#	make -C local
-----------------------
```

Setup environment variables and compile

```
>export SNIPER_ROOT= /path/to/your/home/sniper-6.0
>export BENCHMARKS_ROOT=$(pwd)
```

On ENS servers use setenv command instead:

```
>setenv SNIPER_ROOT /path/to/your/home/sniper-6.0
>setenv BENCHMARKS_ROOT /path/to/your/home/sniper-6.0/benchmarks
```

Then compile the benchmarks using the following command:
>make

Place two configuration files (*small.cfg* and *big.cfg*) attached with this assignment under ~/sniper-6.0/config/.

For ENS servers to above configuration files (*small.cfg* and *big.cfg*) use the following scp command:

```
>sudo scp /path/to/sniper_assignment/*.cfg ens-username@ens-computer-
name.engr.colostate.edu:path/to/sniper-6.0/in/ens/server/config
```

For ENS servers place additional patch file, *tool-run-sniper.patch*, enclosed with this assignment under ~/sniper-6.0/tools/. Then patch *run_sniper.py*:

```
>sudo scp /path/to/sniper_assignment/tool-run-sniper.patch ens-username@ens-
computer-name.engr.colostate.edu:path/to/sniper-6.0/in/ens/server/tools
>cd ~/sniper-6.0/tools/
>patch run_sniper.py < tool-run-sniper.patch
```

Q1 [70 points]. In this problem you will explore the cache hierarchy in an x86 processor using Sniper. Consider an x86 multiprocessor subsystem of 4 in-order cores running at 2 GHz, with L1 data cache size of 4 KB and associativity of 1, L1 instruction cache size of 4 KB and associativity of 1, and unified L2 cache size of 32 KB with associativity of 4. (See *small.cfg* in attachment.) The following command simulates a 4-core system running *lu* benchmark with large input size using architectural parameters specified in *small.cfg*:

```
>cd ~/sniper-6.0/benchmarks
>./run-sniper -p splash2-lu.cont -i small -n 4 -c small
```

After simulation ends, you should find performance summary in *sim.out* under current folder.

```
cat sim.out
```

(a) Suppose you are given a budget of 64KB to distribute between L1 instruction and data caches, and a maximum L2 cache size budget of 2MB. You are also allowed to increase maximum L1 associativity to 4 and maximum L2 associativity to 8. Modify *small.cfg* and simulate to find the cache configuration for *lu* that gives the best performance results (i.e., smallest execution time in the *sim.out* file) while respecting the constraints on associativity and size for caches. Present the cache size and associativity values of your best configuration and improvement in execution time over the base case.

(b) Repeat the analysis for the *cholesky* benchmark with small input size, using the command below:
>./run-sniper -p splash2-cholesky -i small -n 4 -c small

Q2 [80 points]. In this problem you will compare simulation results between in-order processor (*small.cfg*) and out-of-order processor (*big.cfg*). The following command simulates a system using out-of-order cores with McPAT power model enabled.
>./run-sniper -p splash2-cholesky -i small -n 4 -c big --power
After simulation, you should find total power summary displayed in terminal and performance summary saved in *sim.out* under current folder.

(a) For both *lu* and *cholesky* benchmarks, create a table to compare simulation results (average power, energy, IPC, and execution time), between in-order configuration and out-of-order configuration. What conclusions can you draw from these results?

(b) For the *cholesky* benchmark, find a parameter in *big.cfg* to modify so that the IPC increases without changing the cache configuration. Discuss the modified parameter in the configuration file and the IPC improvement obtained. Then apply the modified configuration file to the *lu* benchmark and report IPC improvement. Try to explain your method and results.