Lab0 EE156/CS140 **Introduction to Sniper Simulator**
Estimated time 7 - 10 hours
TAs: Parnian Mokri, Robert Costa and Bharat Kesari
See Canvas for Office Hours and Zoom Links Due Date: 2/3/23

1 Overview

This assignment introduces the student to microprocessor simulators and explains how computer architects use these simulation models to quantitatively evaluate different CPU designs. The assignment will show you how to do the following:

- Run sniper with a test application.
- Conduct an experiment that will explore the effect of L2 cache size on the system's performance and power.

This lab is an introduction to the following general skills:

- File management with bash/python
- Post-processing files with bash/python scripts
- Writing a technical report in LaTex

2 Getting Started

In this class, we will use the Sniper Multi-Core Simulator. This simulator contains some of the tools such as pin and McPAT to profile workloads behavior and estimate power consumption. We will use these tools to study workloads behavior on different computer architecture topologies.

1. Using your EECS username, login to linux.eecs.tufts.edu via SSH **and then log in to red-giant via SSH**. For example:

   $ ssh ⟨your-utln⟩@linux.eecs.tufts.edu
   $ ssh ⟨your-utln⟩@red-giant.eecs.tufts.edu

2. Check your quota with the 'quota' command. Make sure 'space' is big enough to hold to hold 634 MB. In the example below, the user doesn't have enough space:

   Disk quotas for user mbauer01 (uid 1000):

Filesystem	space	quota	limit	grace	files	quota	limit	grace
vs-home:/cs_ugrad2/mbauer01	534M	922M	1024M		5657	19k	132k	

   - If you don't have enough space, fill out the quota request form at https://www.eecs.tufts.edu/userguide/forms/quota.php.
     You'll need to log in with your ECE username and password. Fill in ECE as your department, '/ece' as the file system, a requested final total quota size of 5 GB (or more as needed, if you've already had a quota increase in the past), a faculty sponsor of Mark Hempstead, and specify EE156/Comp140 as the reason.

3. Make a subdirectory on your linux.cs.tufts.edu home directory to hold a copy of sniper called 'sniper-container':

   $ cd
   $ mkdir sniper-container

   Note: Your home directory structure is shared between the Tufts red-giant, Linux, and homework server. **You must be on red-giant to run simulations with sniper**. If you run sniper on any other server, your simulations will likely be killed by IT administration.

4. cd to your ‘sniper-container’ subdirectory and grab a copy of sniper 7.3 for yourself by running:

   $ cd ~/sniper-container
   $ cp -r /usr/sup/sniper-7.3 .

   Note: Disregard any error messages about ‘.snapshots’, you don’t need it.

   Once it finishes – it will take a while depending on your connection, 10 to 30 minutes at least– you should have your own copy of Sniper 7.3 in the ‘sniper-container’. If that copy ever gets mangled, delete it and re-run the cp command to grab a new copy.

   Note: Once downloaded, check if you have the write permission to the sniper-7.3 folder by running

   $ ls -l

   from within the ‘sniper-conatiner’ directory. If not, run

   $ chmod +w sniper-7.3

5. Set the following environments. You can find the absolute path using the pwd command.

```
setenv GRAPHITE_ROOT YOUR_PATH_TO_COPY_OF_SNIPER

setenv SNIPER_ROOT YOUR_PATH_TO_COPY_OF_SNIPER

setenv BENCHMARKS_ROOT YOUR_PATH_TO_COPY_OF_SNIPER/benchmarks/

setenv PIN_HOME  YOUR_PATH_TO_COPY_OF_SNIPER/pin_kit
```

   Note: An example '**YOUR_PATH_TO_COPY_OF_SNIPER**' is "/h/[your-utln]/sniper-container/sniper-7.3"

6. Setup python virtual environment

   $ virtualenv --python=/usr/sup/bin/python2 venv

   Note: If you get a permission denied error, rename the 'venv' within the sniper-7.3 directory to something other than 'venv' and rerun the command.

7. Activate the virtual environment **(you need to do this any time you want to run sniper)**

   $ source venv/bin/activate.csh

8. Test sniper (run this command inside the 'sniper-7.3' directory)

   $ cd sniper-7.3
   $ ./run-sniper -- /bin/ls

   Note 1: that is two separate dashes.
   Note 2: you may see an error which includes: "Pin app terminated abnormally due to signal 11." That is okay as long as the file 'sim.out' is created in the 'sniper-7.3' directory and is non-empty.

9. **$ cd benchmarks**

10. Test one of sniper's benchmarks

    $ ./run-sniper -p splash2-fft -i test -n 4 -c gainestown

    Note: For running benchmarks you have to run the 'run-sniper' executable that is in the 'benchmarks' directory.

3 Sniper Multi Core Simulator

Sniper is a cpu simulator for x86 architectures [1]. The simulator is fast because it's based on a higher level of abstraction than cycle accurate simulators, there are both interval and ROB models available. http://snipersim.org/w/Interval˙Simulation.

3.1 Running Sniper without Configuration

$ cd *YOUR_PATH_TO_COPY_OF_SNIPER*

$ source venv/bin/activate.csh

$ ./run-sniper -- /bin/ls

Note: *YOUR_PATH_TO_COPY_OF_SNIPER* is the path to the 'sniper-7.3' directory.
For example: '/h/[your-utln]/sniper-container/sniper-7.3'

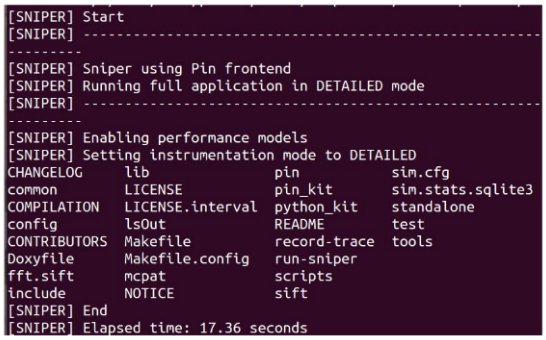


Figure 1: running sniper on /bin/ls without any config files

3.2 Sniper Configuration

Sniper allows computer architects to test different computer architecture topologies (configurations); see Figure 4. Sniper allows you to configure your architecture using command-line parameters and/or configuration files, located at **sniper/config/**. The default configuration is base.cfg; for which you have to set the parameters such as number of cache levels. To use configuration file use -c.

./run-sniper -c gainestown /bin/ls

Configuration files are evaluated from left to right on the command-line, i.e., newer values override older ones. Take a look at some of the configuration files in this directory and see how architectural features are specified for different processor configurations. The default configuration is supposed to model the microarchitecture of the Intel Gainestown processor https://en.wikipedia.org/wiki/Xeon#Gainestown.

3.3 Creating a Configuration File

By adding a configuration file such as **YourCFG.cfg** in the **YOUR_PATH_TO_COPY_OF_SNIPER/config** directory system characteristics can be modified. Figure 2 shows a custom config file for cache level 3; setting its size to 16 MB. The following command runs sniper on ls program with YourCFG overwriting the original value of cachesize.

$ **./run-sniper -c gainestown -c YourCFG /bin/ls**

```
[perf_model/l3_cache]
cache_size = 16384 #KB
```

Figure 2: Example of YourCFG to reset l3 cache size

- Run sniper on /bin/ls as a test; see subsection 3.1
- Look for **[perf_model/l3_icache]** in sim.cfg to see an example of cache configuration
- Open YourCFG.cfg in /path/to/sniper/config; specify which cache you are customizing, and edit the parameter that you want as shown in Figure 2 .

3.4 Sniper Output

Each time you run sniper, it creates at least 5 files: sim.cfg, sim.info, sim.out, sim.scripts.py, sim.stats.sqlite3. The complete list of sniper's output can be seen in Figure 3; these files can be used as sample for your experiments.

Note: you cannot use this configuration directly but you are encouraged to refer to it when you

are designing your configuration file later on in this class. The **sim.cfg** shows the configuration of the current simulation. The cache policies can be changed based on **sniper/common/core/memory_subsystem/cache**.
You are encouraged to look at .cc files and understand policies for later assignments.

```
fft     fft.o     sim.cfg    sim.out          sim.stats.sqlite3
fft.c   Makefile  sim.info   sim.scripts.py   viz
```

Figure 3: Sniper Output Files

The **sim.out** has the simulation results, including IPC, Instruction, time, cache summary for each core. While **sim.stat.sqlite3** is the output in SQL format.

Figure 4: Topology for default quad-core Nehalem-based Xeon and FFT test application

Sniper provides scripts to visualize your output, you can run the following commands to create corresponding images.

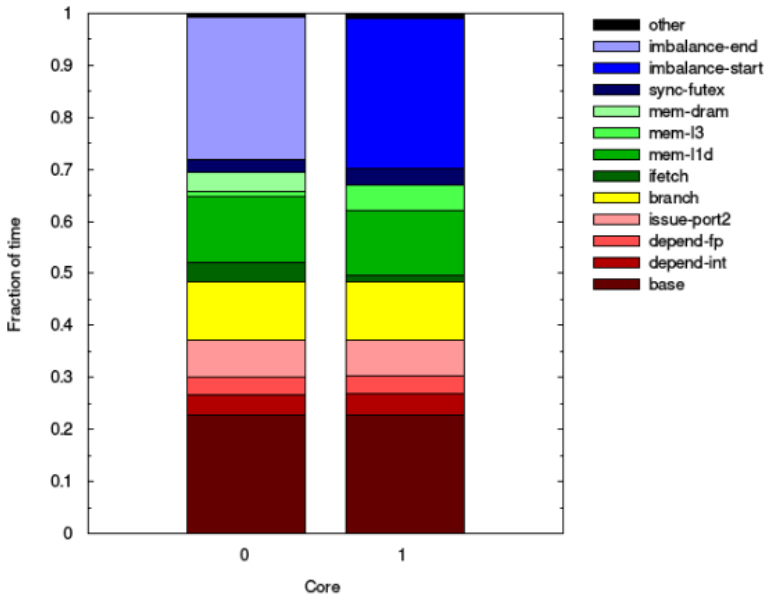


Figure 5: CPI for default quad-core Nehalem-based Xeon and FFT test application

- Run '**YOUR_PATH_TO_COPY_OF_SNIPER**/tools/gen_topology.py': system topology

- Run '**YOUR_PATH_TO_COPY_OF_SNIPER**/tools/cpistack.py' generates cpi-stack output
- Run '**YOUR_PATH_TO_COPY_OF_SNIPER**/tools/mcpat.py' : generates power output
- Run '**YOUR_PATH_TO_COPY_OF_SNIPER**/tools/dumpstats.py' : detailed statistics

The viz folder provides more detailed output. To generate the viz folder, add the **--viz** flag to the "run-sniper" command. The viz folder is generated as a child directory of the specified output directory. For example, **YOUR_PATH_TO_OUTPUT_DIR/viz/levels/level2/data** shows the data values in cache level 2. Python scripts are available to customize sniper, change physical parameters like frequencies of cores, or have it periodically collect data; take a look at **sniper/scripts** for guidance, pay especial attention to
periodic-stats.py.

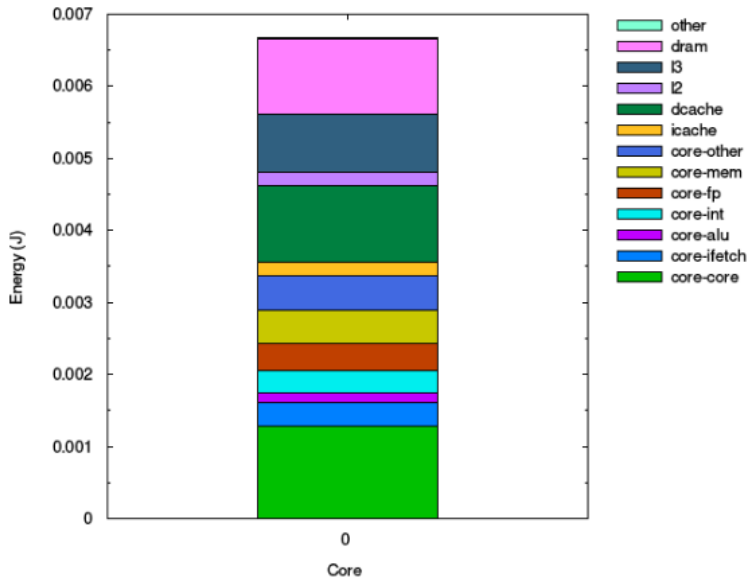


Figure 6: Power for default quad-core Nehalem-based Xeon and FFT test application

Figure 4 shows your system topology: you have two cores in the system and 3 levels of cache. It also shows the size of these caches. Figure 5 shows Cycle Per Instruction (CPI) distribution in each core. CPI is a metric to measure performance. It shows number of clock cycles it takes to finish an instruction. Other performance metrics can be calculated based on CPI or the **sim.out** entries.

$$CPI = \frac{\text{The number of clock cycles required to execute the program}}{\text{number of instructions executed in running the program}}$$

The **McPAT** (Multicore Power, Area, and Timing) model from HP can be used to model power, area, and timing of multithreaded, multicore, and manycore architectures [2]. After you run mcpat.py you will have 4 files that estimate power consumption; power.txt gives you core and processor power. Figure 6 shows McPAT result of running the fft benchmark on core 0.

3.5 Additional Workloads

To design a system, computer architects need to compare the performance of different machines. To do so computer architects run a small set of programs that represent certain behaviour seen in real applications. These programs are called benchmarks. SPLASH-2 is probably the most commonly used suite for scientific studies of parallel machines with shared memory. You can find the original paper on splash2 in this link.

This is a list of **Splash2 Workloads**:

barnes cholesky fft fft_O0 fft_O1 fft_O2 fft_O3 fft_forever fft_rep2 fmm lu.cont lu.ncont ocean.cont ocean.ncont radiosity radix raytrace raytrace_opt water.nsq water.sp barnes-scale fft-scale fmm-scale lu.cont-scale lu.ncont-scale ocean.cont-scale radix-scale water.nsq-scale

Your simulation commands will have the following form:

$ nice ./run-sniper -d YOUR_PATH_TO_OUTPUT_DIR -p splash2-fft -i small -n 4 -c gainestown --roi --viz

Note: You can run the simulation without the word "nice", but it is recommended so that other students using the machines won't see slow responses at the terminal.

- -d precedes the directory where the output files will be written
- -p proceeds the workload to be run (just replace 'fft' with the splash2 workload you would like to run)
- -i indicates input size; There are four options **-i test**, **tiny**, **-i small** and **-i large**.
- -n indicates the number of cores and will be 4 in this case

3.6 Scripting

Computer architects often have hundreds if not thousands of simulations to run for an experiment and therefore have a multitude of output files and data points to parse. Therefore, it is necessary to write scripts which will automate the process of beginning simulations and parsing the data. In general, it is ideal to just have one script which kicks off every simulation and parses/plots all the relevant resultant data. For kicking off simulations, it is recommended that you use *GNU parallels*, while parsing and plotting can be done using either a shell language such as tcsh or bash, or with a higher level scripting language such as python. The standard format for data to be stored is in a .csv file because of its simplicity, though larger analysis tools in industry might utilize databases and input/output in XML/JSON format.

3.7 Shell scripting

These scripts are run by the Unix shell. One way to write it is to run unix commands in a terminal to complete a task and then write a script that generalizes what you want to do. You can use bash script to run programs, file manipulation, printing, setting up the environment, and any necessary cleanup. Read more on shell programming in this link or other online tutorials.
Here are the steps to write a bash script:

- vim NAME_OF_THE_SCRIPT.sh
- write a script, see script 1 for an example
- set the permission on your script:

```
chmod +xw NAME_OF_THE_SCRIPT.sh
```

Listing 1: example of a bash script:test.sh

```
#!/bin/bash
# define your variables
benchmarkDir="YOUR_PATH_TO_SNIPER/benchmarks"
outputDir="YOUR_PATH_TO_SNIPER/benchmarks/outputDir"
simOut="YOUR_PATH_TO_SNIPER/benchmarks/outputDir/sim.out"
## go to benchmark folder inside sniper
cd "$benchmarkDir"
## make a oputputDir
mkdir -p "$outpukDir"
## You can run sniper here, use the same command as in Sec 3.5
##see if output (sim.out) was generated, echo "ok"
test -f "$simOut" || echo "ok"
```

3.8 python

Python is a good choice for manipulating files and post processing of raw data since implementing math functions is easier done in python than in bash. Here is how you do the same things as shown in bash script 1 in python.

Listing 2: example of a python script

```
import os
# define your variables
benchmarkDir="YOUR_PATH_TO_SNIPER/benchmarks"
outputDir="YOUR_PATH_TO_SNIPER/benchmarks/outputDir"
simOut = "YOUR_PATH_TO_SNIPER/benchmarks/outputDir/sim.out"
## go to benchmark folder inside sniper
os.chdir(benchmarkDir)
# define the name of the directory to be created
try:
    os.mkdir(outpukDir)
except OSError:
    print ("NOK" % outpukDir)
else:
    print ("OK %s " % outpukDir)
## You can run sniper here*
##see if output (sim.out) was generated, echo "ok"
if simOut.is_file():
```

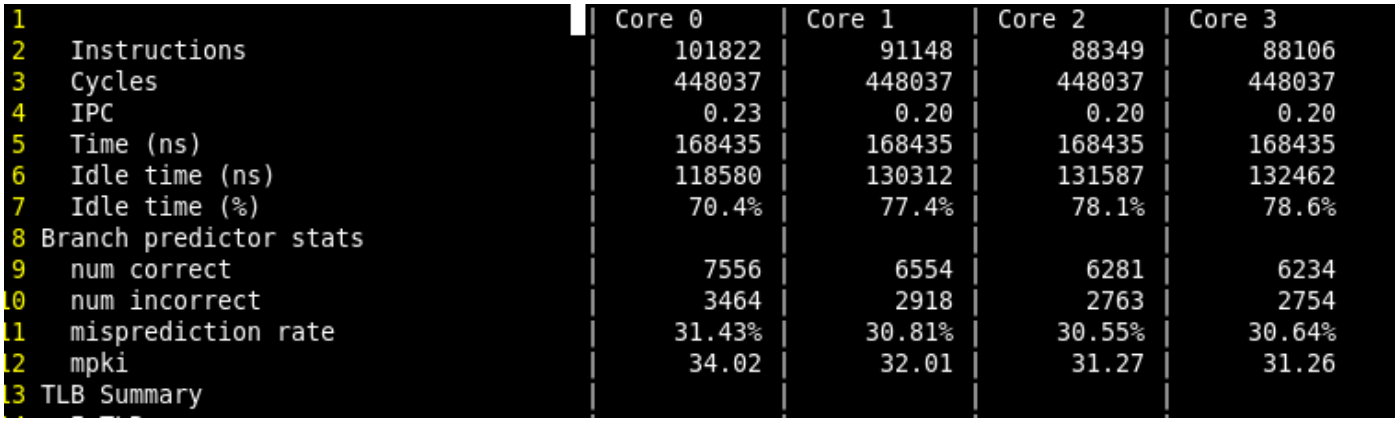
*See this link on how to run a program in python.

TA's Note: I usually use bash to run programs, create, and clean up directories and python to read .csv files and do some post processing on the raw information.

3.9 Pattern Matching

Pattern matching is based on string matching and comes in handy when you are dealing with a large number of files to process or if the file itself is large. Each scripting language offers pattern matching and has its own syntax. Pattern matching can be used to create csv files based on raw data or to extract specific data from csv files.

Practical approach The best approach is to stare at the raw data for a while, figure out what you want to do and then script it using a language you are comfortable with. If you haven't used scripting before, remember that it has a learning curve. So, give yourself some time and plan accordingly. Sniper will give you a text output that you should analyze. Figure 7 shows a partial snapshot of the file. Your script should be able to read the lines from the output file and populate a csv. Another approach is to turn | (pipe character) to , (comma) and save it to a csv.



	Core 0	Core 1	Core 2	Core 3
Instructions	101822	91148	88349	88106
Cycles	448037	448037	448037	448037
IPC	0.23	0.20	0.20	0.20
Time (ns)	168435	168435	168435	168435
Idle time (ns)	118580	130312	131587	132462
Idle time (%)	70.4%	77.4%	78.1%	78.6%
Branch predictor stats				
num correct	7556	6554	6281	6234
num incorrect	3464	2918	2763	2754
misprediction rate	31.43%	30.81%	30.55%	30.64%
mpki	34.02	32.01	31.27	31.26
TLB Summary				

Figure 7: A snippet of Sim.out for a test benchmark

Python: Regular Expression is essentially a highly specialized language that's imported into python as a lightweight package. There are many ways to open a file and find a string in it, see example of opening files and finding a string in it. Or, you can use a similar approach to replace special characters. See example to replace a string or intro to python's regular expression

bash: There are many tutorials to find patterns using bash for example:intro to pattern matching in bash. However, as any scripting you should plan a head what you want to extract and then use different instructions to implement your idea. For example, listing 3 shows one way you can find a pattern using bash. This example detects whether the substring "Linux" appears in another string.

Listing 3: Using bash to find a pattern in a string

```
#!/bin/bash

STR='GNU/Linux is an operating system'
SUB='Linux'
if [[ "$STR" == *"$SUB"* ]]; then
  echo "It's there."
fi
```

You can also use commands like **awk** or **grep** to either print the lines that match a pattern listing 4 or split lines into different fields as shown in listing 5.

Listing 4: using awk to print lines of file.txt that match a pattern

```
$ awk '/keyword/ {print}' file.txt
```

For each record i.e line, the awk command splits the record delimited by whitespace character by default and stores it in the $n variables. If the line has 4 words, it will be stored in $1, $2, $3 and $4 respectively. Also, $0 represents the whole line. See the complete example here.

Listing 5: using awk to split lines of file.txt

```
$ awk '{print $1,$4}' file.txt
```

3.9.1 Running Simulations

It does not make sense to run one simulation on the command line, wait, and then run another. Instead, a sensible person will write a script which will contain all simulation commands. Even better is to write a script which can execute your simulations *in parallel*. For this experiment, it is recommended that you use GNU parallels whose documentation can easily be found online. In it's simplest form, one can list all commands (such as the one above which you will be using for this lab) in a text file and then pass it to parallels in the following way:

parallel < commands.txt

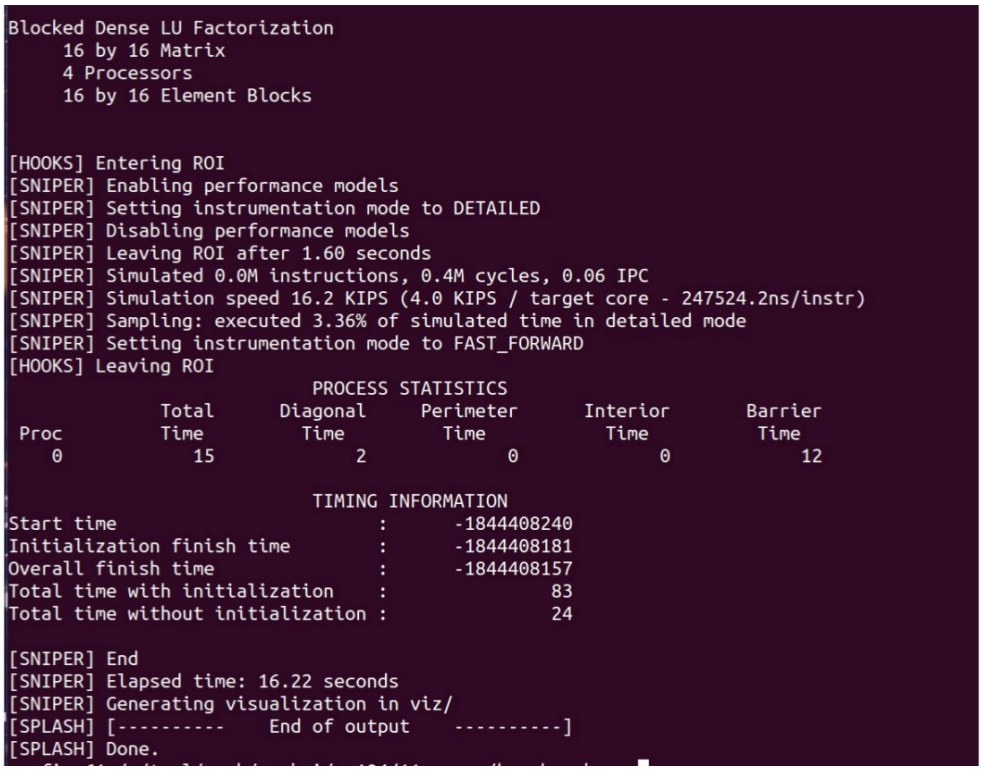


Figure 8: The end of lu simulation in terminal.

Note: it is highly recommended that you use a terminal multiplexing tool such as *tmux* (or screen) when running your simulations to prevent them from being disrupted by possible loss of connection to eecs/wormhole since simulations often take many hours to complete.

4 Assignment - Sweep of Configurations

Computer architects sweep benchmarks across different configurations to study design trade offs. These sweeps are typically for 100s of configurations and dozens of benchmarks; however, the purpose of this assignment is to introduce snipersim to you so we will do a small sweep. First, set the L1 size to 32KB, and then sweep the size of L2 cache as shown in Figure 9 for three different benchmarks. Finally, analyze the effect of cache size on IPC and energy consumption. Please see the following steps for more details.

Name of benchmark
benchmark1
benchmark2
benchmark3

X

L2 size (in KB)
512
1024
4096

Figure 9: L2 cache size sweeps for each benchmark.

1. Choose 3 of splash2 benchmark programs; use **-i small** for all of them.
   Here are the list of benchmarks you can choose from:
   cholesky, fft, fft_O0, fft_O1, fft_O2, fft_O3, fft_rep, fmm, lu.cont, lu.ncont, ocean.cont, ocean.ncont, radiosity, radix, raytrace, raytrace_opt.
   Note: It is highly recommended that you do not choose all benchmarks under the same application suites. For example, choosing fft, fft_O0, fft_O3 to be your benchmarks is likely to yield less interesting simulation results comparing to choosing fft, ocean.cont and radix as your benchmarks. For more details on each application suites, please reference the original splash2 paper.
   Note: It is possible that the names of these benchmarks here is a little bit imprecise. You can find correct benchmark names in section 3.5 or absolute/path/to/sniper/benchmarks/splash2/splash2/splashrun

2. Create a cfg file for L1 cache and set cache_size to 32KB in YOUR_PATH_TO_SNIPER/config/
   `keep this constant for all the experiments`.

3. Create 3 cfg files for L2 cache sizes 512KB, 1MB, 4MB in YOUR_PATH_TO_SNIPER/config/. An example of a costume cfg is given in Figure 2 .

4. Write a script (python or bash) to run your experiment. The script should

   - Create a directory for benchmark and sub-directories for each size of L2
   - Run sniper on each benchmark with 3 different L2 cache sizes as shown in Figure 9. Make sure to place each output in the correct directory

     Note: Remember, to run sniper on integrated benchmarks you have to cd to benchmarks folder first and use the run-sniper script in that folder.
     Note: The python virtual environment must be activated to run sniper.
   - Check if sim.out is created and if its size is not zero
   - Run the post processing python scripts for topology, cpistack, and mcpat in each of the output folders (total of 9 output folders).
     Note: you can find them in the following path:

     ```
     topology: YOUR_PATH_TO_SNIPER/tools/gen_topology.py

     cpi stack: YOUR_PATH_TO_SNIPER/tools/cpistack.py
     ```

```
mcpat: YOUR_PATH_TO_SNIPER/tools/mcpat.py
```

- Write some scripts to extract information from sim.out and power.txt/power.xml. Create a csv file with following columns: name of the benchmark, its input size, core0's IPC, the value of Peak Power of processor, and the value of Peak Power of L2 cache.

benchmark name	input size	L2 size	core0 IPC	Peak Dynamic Power of processor	Peak Dynamic of L2
fft	small	4	extract from sim.out	extract from power.xml/power.txt	extract from power.xml/power.txt
fft	small	8			
fft	small	16			

Table 1: A csv file is a comma separated vector. They are like your normal text files but they end with .csv. You can use scripting to manipulate these files easily. For example, you can combine your 9 different csvs into one csv called All.csv by using "cat". Something like: cat path1/test.csv ... path9/test.csv >All.csv

5. Compare the IPC and power consumption of cores and processor for all three benchmarks with the same size of L2 cache. For example, compare IPC of three benchmarks when L2 = 4MB.

Remember here is how you run sniper:

$ nice ./run-sniper -d /path/to/outputdir -p splash2-bnchmrk -i small -n 4 -c gainestown --roi --viz

5 Assignment - What to Provide

Note: Writing a report is an important skill, since it is used to communicate your results and analysis to your peers. You should write your report from the perspective that you are a computer architect working in industry who is trying to inform his or her colleagues. **It is highly recommended that you use LaTeX for your report.**

1. A report in `PDF format` describing your experiments. for this assignment, you need a short report which includes one paragraph on experimental setup, energy consumption, and performance analysis.

   - `Experimental setup`; describe which benchmarks, input size, and configuration file you used. Include a graphical representation of your architecture configuration. An example of topology is shown in Figure 4.
   - `Energy Results`; Report the energy stack for each benchmark as shown in Figure 6. Use your results to make a particular conclusion about the differences in energy consumption. For example, show the energy plot of 3 benchmarks when all of them have an L2 cache of 4MB, show the corresponding CPI stack as well. Write a paragraph that describes this graph and some observations.
   - `Performance Analysis`; Use your csv and plot IPC for each benchmark and all 3 cache sizes. Write a paragraph that describes these graph and some observations.

2. Submit your scripts (bash or python) for both post-processing and experimental setup. Please name them accordingly. The name of the scripts should express what the script is doing. Please make sure that there is no space between the words. For example, a script that sets up parallel tasks can be named as: settingUpParallelTasks.sh or setting_up_parallel_tasks.

3. README file explains your script and what each script does

4. The csv file : Combine all the csv files into one. You can use cat command as in caption of Table 1.

Bonus: (Extra 3 Credits) Consider all the configuration parameters for caches in sim.cfg. What other parameters would affect IPC and how? Please state your hypothesis, the logic behind it, test your hypothesis using sniper, and include your results.

To provide:

- ssh to homework or any other server : ssh homework
- zip all your files and provide it like this:

```
provide ee156 lab0 yourName_lab0.zip
```

References

[1] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulations," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, Nov. 2011, pp. 52:1–52:12.

[2] S. Li, J. H. Ahn, R. Strong, J. Brockman, D. Tullsen, and N. Jouppi, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *42nd Annual IEEE/ACM International Symposium on Microarchitecture, 2009. MICRO-42*, Dec. 2009, pp. 469–480.