

COEN-4730 Computer Architecture

HW #3

Dept. of Electrical and Computer Engineering, Marquette University
Cristinel Ababei

Objective:

To learn about SimpleScalar and Wattch tools. To learn how to find a suitable superscalar architecture for a specific benchmark through so called Design Space Exploration (DSE). This exercise should provide insights into superscalar architectures.

1. SimpleScalar and Wattch simulators

The **SimpleScalar tool set** is a software system infrastructure used to simulate a range of processors and systems. The tool set includes sample simulators ranging from a fast, functional simulator to a detailed, dynamically scheduled processor model that supports non-blocking caches, speculative execution, and state-of-the-art branch prediction. It used to be very popular. In early 2000's, more than one-third of all papers published in top computer architecture conferences used the SimpleScalar tools to evaluate their designs.

Wattch is an architectural simulator that estimates CPU power consumption. The power estimates are based on a suite of parameterizable power models for different hardware structures and on per-cycle resource usage counts generated through cycle level simulation. The power models have been integrated on top of the SimpleScalar architecture simulator.

2. Preparation

Install (on your Linux machine or virtual box that you used for the previous assignment) and familiarize yourself with the SimpleScalar 3.0 Toolset and the Wattch 1.02 Toolset.

A. Install SimpleScalar 3.0

The simplescalar 3.0 simulator installation files are provided with this assignment. It is simplesim-3v0e.tgz in the .zip file provided. Download it onto your machine. It could be also downloaded from: <http://www.simplescalar.com/>, but, the website may not be on.

Compile simplesim-3.0 (the SimpleScalar processor simulator), type the following commands at your terminal:

```
> cd $HOME/simplesim-3.0 ($HOME should reflect the location where you unpacked the .tar file; for example, in my case I use $HOME as /home/cristinel/coen4730/ because this is the directory where I unpacked and installed this tool)
> make config-pisa
> make
```

B. Test your SimpleScalar

Type the following commands:

```
> cd $HOME/simplesim-3.0
```

> `./sim-outorder ./tests/bin.little/test-math` (by default: out-of-order issue)

You should get an output that looks like this:

```
sim: ** starting performance simulation **
      pow(12.0, 2.0) == 144.000000
      pow(10.0, 3.0) == 1000.000000
      ...
      -1e-17 != -1e-17 Worked!
```

From the output, you will find that `sim_CPI` (cycles per instruction) is about 1.0497

Type the following commands:

> `cd $HOME/simplesim-3.0`

> `./sim-outorder -issue:inorder ./tests/bin.little/test-math` (in-order issue)

From the output, you will find that `sim_CPI` (cycles per instruction) is about 1.51
Why are the CPIs different?

C. *Getting Familiar with the SimpleScalar*

Take some time to read and get familiar with the documentation of SimpleScalar. Several .PDF files and other resources are included in the archive of files provided for this assignment. Try different simulators available: `sim-safe`, `sim-fast`, `sim-eio`, `sim-profile` (*you will use this for one of the items in the assignment described later in this document*), `sim-cache`, `sim-bpred`, `sim-outorder` (pre-compiled test programs can be found in the directory: `$HOME/simplesim-3.0/tests-pisa/bin.little/`), compare and understand the meaning and usage of these simulators.

Run the `sim-outorder` simulator, notice the impact of different branch prediction policies (e.g., always not taken, always taken, bimod, 2 level).

D. *Install Wattch 1.02*

Download Wattch from the .zip file provided for this assignment. It is `sim-wattch-1.02e.tar.gz`. But, you can also download it from here: <http://www.eecs.harvard.edu/~dbrooks/wattch-form.html>

Compile the Wattch simulator, type the following commands at your terminal:

> `cd $HOME/sim-wattch-1.02d` (again, this depends on where you unpacked the .tar file)

> `make config-pisa`

> `make`

Read the Wattch paper provided in the archive of this assignment. Try the Wattch `sim-outorder` and see what the new features are. Wattch incorporates four different power models: `avg_total_power_cycle`, `avg_total_power_cycle_cc1`, `avg_total_power_cycle_cc2`, and `avg_total_power_cycle_cc3`. What are the differences?

Note:

For the purpose of just running simulations and collecting numbers to be able to do the assignment (described later in this document), it would be enough to download and use only the Wattch tool. That is because it is built on top of a version of the SimpleScalar simulator. When you run Wattch, the output will be similar to the one when you run SimpleScalar, but power numbers are reported too, which you will

need to compute the EDP. However, it is a good educational exercise to see both tools, the original and the improved one...

3. Design Space Exploration (DSE)

By this time, you should have played enough with SimpleScalar and Wattch tools. Here is the real-deal assignment. The task is to find out the most suitable superscalar architecture(s) for the *compress* benchmark. Wattch (sim-outorder) is mainly used in this assignment, as it can provide both performance and power results. However, other SimpleScalar tools, like sim-profile, sim-cache, sim-bpred,... may be helpful due to their faster simulation speed and more detailed information.

The goal of this assignment is to gain a better understanding of the superscalar architecture.

General Description

- The benchmark application we use in this assignment is *compress*.
- By tuning the parameters of SimpleScalar architecture simulator, we look for a processor architecture which is “most suitable” for running the *compress* application.
- We use two metrics to define “most suitable” in this assignment:
 - 1) **Performance**, in terms of IPC (instructions per cycle), where IPC corresponds to *sim_IPC* in the Wattch simulator.
 - 2) **Energy**, in terms of EDP (energy-delay product). Here, we use the approximate formula ($CPI * Energy/Cycle$)**CPI*. CPI (cycles per instruction) corresponds to *sim_CPI* and Energy/Cycle corresponds to *avg_total_power_cycle_cc3* in the Wattch simulator.We will mainly look at some key components of a superscalar architecture, such as the branch predictor and the memory system.

System Default Configurations

- Use “-dumpconfig” flag of *./sim-outorder* to dump the default architecture configuration to some file, (say *my1.cfg*);
- Try to understand the meaning of each parameter by referring to the SimpleScalar related documents and the help (“-h”);
- The parameters we are interested in, together with their default values, are listed in the parameters file **1_parameters_file.pdf** that is part of the archive of this assignment. (In this assignment, only these parameters are allowed to be tuned). The bottom line is that you must set a PRACTICAL configuration. For example, you are not allowed to choose the “Perfect” branch predictor as this is the ideal case. Likewise, you cannot modify the cache access/miss latency and set it to “0”, as it makes no sense either;
- The “-config” flag is a very useful flag for simulating your own configuration. You can first dump the default configuration file (say into *my1.cfg*), edit this file to tune the parameters, and save it as a new file (say *my2.cfg*), then run the simulation by simply calling “*./sim-outorder -config my2.cfg ...*” instead of explicitly configuring parameters on the command line one by one.

The *compress* Benchmark

- The benchmark we use in this assignment is *compress*, which is an integer benchmark. The pre-compiled binary (*compress.ss*) and its input file (*test.in*) are provided in the archive of this assignment. The sim-outorder simulator of Wattch is used in this assignment to get the CPI (*sim_CPI*), IPC (*sim_IPC*), and Power (*avg_total_power_cycle_cc3*). To simulate, using default settings, an application which has/uses its own input file (like the case of *compress.ss* using the input *test.in*) can be done like this:
> \$HOME/sim-wattch-1.02e/sim-outorder compress.ss < test.in

Use Scripts to automate DSE

- Scripts can ease the pain of performing design space exploration (DSE) on a large design/solution space. For example, let's say you want to explore the design space formed by only three parameters (e.g., -bpred, -cache:dl1, and -cache:il1). The search process would normally go like this: for each value of the 1st parameter, vary values of the 2nd parameter, and in turn for each value of the 2nd parameter, vary all values of the 3rd parameter, etc. The problem is a combinatorial problem and the computational time to sweep the entire search space increases exponentially with the increase of the number of parameters and number of values for each parameter.
- A simple sample-script is provided as part of the archive of this assignment. *You should modify and use it while working on your assignment – to make your task easy, but, it is not required to do that.* However, you are free to use/write your own scripts, using any scripting language (such as Perl) in order to automate your work and make your life easy. You can also use a script to pick-up data from the multiple output files, which can also make the analysis easier.
- Do not use a script to vary all the parameters and carry out a brute-force search of all the parameters from the parameters file **1_parameters_file.pdf**. It could run for very long and without any guarantee to provide a meaningful result.

Assignment Guidelines

1. Set up a baseline: run the *compress* application with the default configuration/settings. The performance of this run, i.e., *IPC*, and energy-delay product, i.e., $(CPI * Energy/Cycle) * CPI$, are the performance baseline and performance-energy baseline respectively.
2. Profile the benchmark using *sim_profile*, and identify its characteristics (e.g., percentages of each instruction type). This task is part of the assignment.
3. The parameters that you can tune during this assignment are pre-categorized into four groups: A) Branch prediction; B) Memory system; C) Function units; and D) Other data path (see also the parameters file **1_parameters_file.pdf**).
 - A) Branch Prediction**
 - branch predictor type (-bpred)
 - bimodal predictor config (-bpred:bimod)
 - 2-level predictor config (-bpred:2lev). Refer to Yale Patt's papers for more information.
 - return address stack size (-bpred:ras)
 - B) Memory System**
 - level-1 data cache (-cache:dl1). You need to find out the impact of 1) different caches sizes; 2) same size, but different associativity; 3) impact of block size; 4) impact of replacement policy
 - level-1 instruction cache (-cache:il1). You need to find out the impact of 1) different caches sizes; 2) same size, but different associativity; 3) impact of block size; 4) impact of replacement policy
 - Same as above, but for level-2 cache
 - unified cache or separated cache
 - C) Function Units**
 - number of integer function units (-res:ialu, -res:imult)
 - number of floating point function units (-res:fpalu, -res:fpmult)
 - D) Data Path & Others**
 - impact of instruction decode width (-decode:width)
 - impact of in-order or out-of-order (-issue:inorder)
 - impact of reorder buffer size (-ruu:size)
4. **You must select one parameter from each group. You will perform a solution search in the space formed by your chosen parameters.** For any chosen parameter, when possible, do your search by sweeping at least three different values. Try not to select all four parameters such that they are all the same as of your colleague(s) with whom you are discussing this assignment.

5. Perform design space exploration (DSE)
 - Take each of your chosen parameters individually and sweep its values (keep all other parameters – including your other selected three – at the default values) by running simulations.
 - Plot both the performance graph and the EDP graph with respect to the different swept-values/settings. You will have to include all 8 plots in your report. Confirm whether your plots are what you expected. Explain the results you get in one or two sentences for each plot.
 - Now, perform a full solution space search by varying all your four parameters. You must find the combinations of four parameters that give you the best performance and the best EDP respectively. Note that here you will run 81 simulations because you have 4 parameters with 3 values for each: $3 \times 3 \times 3 \times 3 = 81$. You should use a script especially for this step to automate your work!

4. Deliverables

A report with the following sections:

1. Title, your name, course name, and date
2. **Description:** a paragraph to describe what this assignment is about. What are its objectives.
3. **Profiling:** A listing of the profiling of the benchmark using *sim_profile*. Plus, a short description with observations about it.
4. **Sweeping of individual parameters:** The eight plots discussed above and a paragraph long description for each. Discuss your results and draw conclusions. Each plot should have axis labels. Figures should all include figure numbers and captions that clearly specify what each plot represents. In your descriptions, please refer to individual figures.
5. **DSE of four parameters:** Include a table or two plots (one for performance and one for EDP) with the results of all 81 simulations. Report the combinations of four parameters that give you the best performance and the best EDP respectively.
6. **Conclusion:** Summarize your findings. Be clear about what you learned in this assignment.

7. References and Credits

This assignment was inspired and adapted from the teaching materials of Prof. Henk Corporaal and his TA, Zhenyu Ye.