DAT105 Computer Architecture
Study Period II, 2017
Department of Computer Science and Engineering
Chalmers University of Technology, Göteborg, Sweden

Laboratory Assignment 2: Exploring the Impact of Branch Prediction and Instruction-Level Parallelism (ILP) on Processor Performance

Contributors: Mehrzad Nejat, Mohammad Waqar Azhar, Per Stenstrom

Learning outcome: The main objective of this lab assignment is two-fold: (1) to understand the impact of branch prediction on processor performance and (2) to explore several micro-architectural techniques to exploit Instruction-Level Parallelism (ILP). At the end of this assignment, you must be able to:

- Explore the design space of popular branch predictors and assessment of the impact on performance of the prediction accuracy
- Evaluate the impact of dynamic scheduling, multiple issue and hardware-based speculation on performance

Background: You must have finished the first assignment before starting this one. Therefore, you should already be familiar with the simulation environment and the basic simulation methodology. You should also have some basic knowledge about branch prediction, dynamic scheduling and out of order (OoO) execution, and multiple issue in superscalar architectures as covered by lecture 2 and 3 in the course.

In the first assignment, you have studied the impact of basic cache parameters (cache size, associativity, and block size) on overall performance. You tried to get close to maximum possible speedup and proposed a new optimum configuration for instruction and data caches. In this assignment, we continue to explore other performance enhancing techniques including branch prediction and different instruction scheduling techniques. Check the related sections of the simulator user guide to see what parameters are available and what each parameter refers to. You may need to have another look at the course material and references to make sure that you understand the concept and functionality of these configurations and parameters. If you have any doubts, discuss it with the teaching assistants.

Evaluation: You must write a report that includes: A brief description of the problem and the method you used, important assumptions if you had, simulation results and observations, your design choices and the reason behind them. Detailed report guidelines will be available on Ping-Pong.

Task 1: Branch Prediction

Start from the last configuration you proposed in the previous lab assignment (OPT2) as a base for this task and call it “base2”. If you check the branch predictor type, you will notice that it is set to “not taken”. This is a static branch predictor. There are also several types of dynamic predictors: Bimodal, 2-level, and combined. Think about the following questions:

- ➢ Static branch predictors always assume the same prediction for all the branches. Therefore, they have minimum overhead. But, what about their prediction accuracy compared to dynamic predictors? How big is the difference?
- ➢ How much variation do you expect in branch prediction accuracy among different types of dynamic predictors?
- ➢ How big is the effect of the accuracy of branch predictor on the overall performance?
- ➢ How much does the answer to the previous questions depend on the application?

Similar to the methodology in the previous assignment, start by finding the maximum possible speedup when improving branch prediction. You can use the ‘Perfect’ predictor available in the simulator configuration. First, perform the simulations for base2. Then, create new models by changing the branch predictor type[1] to perfect, bimodal, two level, and combined. Run the simulations and fill out the corresponding sections in Table 1. Calculate the speedup (SP) relative to base2 with the same method you used in the previous assignment.

If you look at the configuration file, you will notice more configuration details for each branch predictor, plus return address stack (RAS) and branch target buffer (BTB). In the final step try to alter these values to see how much further improvement you can make. But, you are limited to the following changes:

- Doubling the table size for bimodal
- Doubling L1 and/or L2 size of 2 level predictor
- Doubling table size of combined
- Changing BTB from “8,2” to “128,4”
- Changing RAS from 0 to 4 or 8

Once you have found the best configuration, name it Best-BP and complete Table 1.

- ➢ Can you answer the above questions now?

[1] Do not change anything else

Table 1: Simulation results for branch prediction

	Application	dijkstra	qsort	stringsearch	gsm-untoast	jpeg-cjpeg	**GM**
base2	CPI						NA
	Misses*						NA
	BP-Accuracy						NA
Perfect-BP	CPI						NA
	SP						
Bimodal	CPI						NA
	Misses						NA
	BP-Accuracy						NA
	SP						
2level	CPI						NA
	Misses						NA
	BP-Accuracy						NA
	SP						
Combined	CPI						NA
	Misses						NA
	BP-Accuracy						NA
	SP						
Best-BP	CPI						NA
	Misses						NA
	BP-Accuracy						NA
	SP						

* Branch miss predictions

Task 2: Exploiting Instruction-Level Parallelism

Usually in a simple single-issue pipeline, CPI is more than one. The reason is that during the execution of the program there are some additional cycles wasted on waiting for memory accesses, recovering from a branch miss prediction, or handling dependencies between instructions. You have already minimized the wasted cycles due to memory accesses and branch prediction inaccuracy. Now, you will try to reduce the wasted cycles due to instruction dependencies by improving the processor core and allowing out-of-order execution. Then, you will further utilize ILP by increasing the pipeline width such that multiple instructions can be issued and executed at the same time. Use the best configuration you proposed in the previous task as a base for this task and name it “base3.txt”. Similar to the previous assignment, perform the improvements step-by-step.

2.1. Increasing the Functional Units (FU)

In the first step, increase the number of FUs. These FUs are Arithmetic-Logical-Units (ALU) and Multiplier/Dividers for integer and floating point operations. However, if you have checked the results of the previous simulations more carefully, you have noticed that floating point units are never used. That is because the benchmark programs you are using in this lab are integer. So, you should focus on integer units. Increase the number of ALUs to 8 and Mults to 4. Run the simulation and fill out Table 2. In this table, calculate the Busy Rate by dividing the number of busy cycles by total execution cycles.

Table 2: Increasing the number of Functional Units

	Application	dijkstra	qsort	stringsearch	gsm-untoast	jpeg-cjpeg	**GM**
base3	CPI						NA
	ALU Busy Rate*						NA
	Mult Busy Rate						NA
More-FU	CPI						NA
	ALU Busy Rate						NA
	Mult Busy Rate						NA
	SP						

* Busy cycles divided by total execution cycles

- Did you notice any performance improvement? What do you think is the reason?

2.2. Out-of-order execution

In this step, you will simulate a single issue out-of-order core. Keep decode, issue, and commit widths equal to one and change the issue to out-of-order. You can change other parameters[1] of the core according to Table 3.
Start from the maximum values and fill out the first section of Table 4. Obviously, maximizing the resources improves the performance. But, it is not for free[2]. In the next step, try to reduce the values as much as you can, such that performance reduction is not considerable. Once you found the optimum configuration, name it OPT3 and fill the corresponding sections in Table 3 and Table 4.

Table 3: Values for core parameters

Parameter	**Possible values**	**OPT3**	**2 wide**	**8 wide**
IF Queue size	2,4,8			
RUU size	16,32,64,128			
LSQ	RUU size ÷2			
Number of ALUs	1-8			
Number of Mults	1-4			
Issue wrong path	True/False			

- How does each core parameter affect the performance?
- Is there any correlation between these parameters?
- Which parameters are more effective in performance improvement?

2.3. Wide issue

In the last step, increase the processor width (issue, decode, commit) to 2 and 8. Run the simulations and also try different values for core parameters. If you decide to make any change in these parameters compared to OPT3, report it in Table 3. Otherwise live the corresponding columns blank. Finally, complete Table 4.

[1] In this simulator register update unit (RUU) works as the reorder buffer (ROB)

[2] Evaluating the different costs of additional hardware resources is out of the scope of this assignment.

Table 4: Out of order and wide issue simulation results

	Application	dijkstra	qsort	stringsearch	gsm-untoast	jpeg-cjpeg	**GM**
Single Max	CPI						NA
	SP						
Single OPT3	CPI						NA
	SP						
2 wide	CPI						NA
	SP						
8 wide	CPI						NA
	SP						

- ➢ How much performance improvement did you notice by increasing the processor width? Compare it with the improvement you made by changing to out of order and altering the core parameters.
- ➢ Did the effect of core parameters on performance change after increasing the width?

Look back and summarize

At this point, you have improved the performance step-by-step from cache optimization to increasing branch prediction accuracy and exploiting ILP.

- ➢ Compare the best CPI you achieved with the CPI of the first base configuration.
- ➢ Compare the improvements you made in each task. Which improvements were more effective?
- ➢ Reflect qualitatively upon how much you increased the hardware resources to achieve this improvement?