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

Edina Dedovic & Hadis Shirkosh

September 26, 2022

1 Goal

In this report the goal is to understand the branch prediction in order to overcome major obstacles and achieve higher performance for a high-performance CPU. An accurate branch prediction is required in order to achieve this performance but is limited in the different branch predictors. We will also see that it is the key to many techniques for enhancing and exploiting ILP. This lab is a comparison on different dynamic branch predictions and a static branch prediction schema.

2 Methdodology

For the purpose of determining the impact of branch prediction methods and hardware configurations. Simhome will be evaluated by executing benchmark programs against different processor configurations in order to analyse its effect on performance. A comparison of the results for different setups will allow us to observe the different effects of branch prediction on performance.

Finding a base configuration based on the Lab 1 was difficult however after some time we decided to do testing with both the optimal solution we found for qsort but also a different approach with the 512 cache size, 8 block size and 8 associativity. Here are the results in Table 1:

Additionally, by configuring simhome to simulate processors with a higher amount of FUs is will give us ability so see impact ILP has. Note, performance will be measured with CPI and the lower this is the better the performance is better. Something to understand from this is that; since it's a simulator we will see possible software errors that distort the accuracy of the reported results and hence the *perfect* branch prediction may not be be best prediction. Ergo, obtained results are generated with the use of simulator and not real-world conditions/testing.

Table 1: Base configuration base2.txt for <128><64><8>

Application	bimodal	perfect	2-level	comb	"not-taken"		Bimodal/double	Comb/double	2-level/double
Dijkstra	5.4351	6.0082	5.4345	5.4320	6.7618		5.4352	5.4320	5.4352
Qsort	10.3433	12.0878	10.0443	10.0402	13.1292		10.3433	10.0402	10.0397
Stringsreach	16.0798	17.5117	16.1518	16.0959	5.9811		16.0811	16.0959	16.2073
Gsm-untoast	2.6478	2.6474	2.6531	2.6495	3.2940		2.6477	2.6494	2.6500
Jpeg-cjpeg	3.7628	4.2048	3.7786	3.7681	4.5827		3.7627	3.7680	3.7809

Application	bimodal	perfect	2-level	comb	"not-taken"		Bimodal/double	Comb/double	2-level/double
Dijkstra	6.3433	6.9157	6.3427	6.3402	7.6688		6.3433	6.3433	6.3433
Qsort	19.1803	20.6540	18.8266	18.8245	21.6767		19.1802	19.1803	19.1803
Stringsreach	16.2369	17.6611	16.3026	16.2537	17.722		16.2382	16.2369	16.2369
Gsm-untoast	2.6622	2.6616	2.6675	2.6638	3.3080		2.6620	2.6622	2.6622
Jpeg-cjpeg	4.6628	5.0928	4.6748	4.6657	5.4810		4.6627	4.6628	4.6628

Table 2: Base configuration for base3.txt for <512><8><8>

2.1 Task 1: Observations and conclusions

During the laboratory the branch predictor were analyzed. Bimodal, 2-Level and combined. "Perfect" is something that simhome introduces; it's namely, a branch predictor that never miss predicts a branch: However that was not the case.

The results are somewhat surprising since both of them show a better performance in combined and bimodal branch prediction models. The perfect one does not however show any signs on improvement. With that being said, once we double the size of the best performing model (bimodal) we realised that it yet again gained in performance in CPI. This should have to do with the fact that for some of the programs, the branch prediction is so accurate that it does not stall the process and has to wait due to high missing prediction rate/direction prediction rate. Of course, this is dependent on the programs again we see that the string search and quicksort is the most affected since they have memory operations affecting the runtime and misses. Missing the prediction means flushing the memory and changing the prediction. Referring to the reference paper.

Every one of them acts different in regard to IC (highest for jpeg encode/decode), floating-point and memory (load and store), control etc.

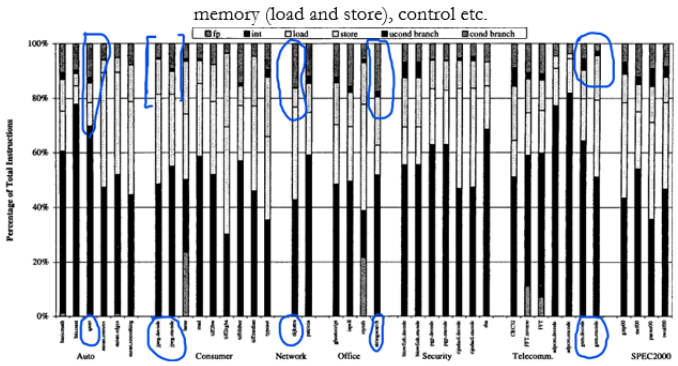

Figure 1: Dynamic Instruction Distribution for large data set.

Observations

Here we see that branch prediction can have a large impact in the application is forced to constantly workloads and stores. The difference between the static and dynamic vary in performance and programs gain a boost but it's not that evident in all cases. The program that benefits from it mostly are also the two most affected. Bimodal (doubled L1 and L2) as well as combined for the programs that do indeed have more unconditional and conditional branches. Usually, on modern CPUs the branch prediction cycle is between 1 and 20 cycles. There are four control flow instructions as mentioned unconditional branch, call/return, conditional branch taken, and conditional branch not taken.

The branch variation in the different branch predictors was not at all varied, for cases such as Dijkstra's algorithm the optimization was already achieved with just implementing the bimodal, one needn't go further.
The branch prediction in this lab is very important, it has been an important role in the single pass process of data. Branch prediction enables the processor to execute instructions long before they are established to be true. Not only have we looked at the branch prediction but also jump prediction.
A modern processor may predict that, say an actual function, is the same as in the previous call and starts executing before pointer is pointing to the actual function. If the jump prediction was wrong, then the CPI is increased even further.

This concludes, bimodal branch prediction can be optimised by utilizing more branch history. However, the main advantage we seen that static branch prediction has been the simplified complexity of the architecture. Lastly, we have seen that static branch prediction is worse in performance since the dynamic branch prediction has analysed the history of access and made realistic prediction in the history table. Also, we see that the optimization with implementing the branch prediction is a bit better but needs some time to "warm up".

2.2 Task 2: Impact of FU and ILP on performance

In this experiment, we tested the performance impact of adding additional Algorithmic Logic Units (ALU) along with integer division and multipliers. There will however not be a table along with the test for adding (or rather changing parameters of them) ALU and dividers. We observed that the amount of ALUs and Mults had non impact on performance whatsoever, due to single-pass. Thus processors was not set to use the additional resources and therefore no performance changes. As previously stated, some benchmarks were better than other and after reading upon the material; we realised it has to do with the software bugs.

2.3 Instruction Level Parallelism

Our goal was to test the performance impact of using different sizes for the Load/Store Queue (LSQ) and the Reorder Buffer (RUU) when the processor executes out-of-order instructions. We enabled the out-of -order execution for the processor; along with that we tested performance of RUU and LSQ. Additionally, we followed some constraints. Results are in Table 2, below.

Here we observed that the larger RUU the higher performance gets. This is logical since the more space is available in the RUU and the LSQ the more opportunities there is for more parallel execution.

	Application	dijkstra	qsort	ss	gsm	jpeg
	Instruction Count	54881769	41898644	300884	11704482	27259353
2level- double	Execution Time	530768106	645145385	54 831 500	31578290	145883217
l2 cache RUU -16	CPIbase	9.6711	15.3977	16.0576	2.6979	5.3516
2level- double	Execution Time	501130374	500461642	4821095	31420697	132865902
l2 cache RUU -32	CPIbase	9.1310	11.9445	16.0231	2.6845	4.87413
2level- double	Execution Time	426309738	423395149	4813930	31896933	121831415
l2 cache RUU -64	CPIbase	7.7677	10.1052	15.9992	2.7251	4.46934
2level- double	Execution Time	389920976	387704598	30449027	21342306	109326129
l2 cache RUU -128	CPIbase	7.1047	9.2533	15.9970	2.6014	4.0105

2.4 FUs and ILP

In this task the goal was to observe the impact of additional FUs, to see whether that changed the impact performance since the ability of parallelism is enabled. We also observed the impact instruction fetch and dispatch had on the performance. Moreover, we enabled program to continue execution even after wrong path was discovered. There was some speculation as if higher widths would also allow higher performance. Stating all this we tested, the results are below in Table 3:

As stated, we have indeed observed that higher values resulted in better performance. Trying

	Application	dijkstra	qsort	ss	gsm	jpeg
	Instruction Count	54881769	41898644	300884	11704482	27259353
Width -8, IFQ -8	Execution Time	369810735	359284608	4571169	20307782	91520777
ALU-8, Mul-4	CPIbase	6.738316598	8.57508916	15.19246288	1.735043208	3.357408263
Width -8, IFQ - 4	Execution Time	371494341	362043342	4593288	21284577	92899983
ALU-4, Mul-2	CPIbase	6.768993561	8.640932198	15.26597626	1.818497991	3.408003961
Width - 8, IDQ - 4	Execution Time	371369376	362005924	4593294	21301543	92897797
ALU-6, Mul-3	CPIbase	6.766716576	8.640039138	15.2659962	1.819947521	3.407923768
Width -4, IFQ -4	Execution Time	372301282	362883965	4594148	21342306	93188020
ALU-4, Mul-2	CPIbase	6.783696823	8.660995449	15.2688345	1.823430204	3.418570499
Width -1, IFQ -4	Execution Time	389920976	387704598	4813267	30449027	109326129
ALU-1, Mul-1	CPIbase	7.104745038	9.253392496	15.99708526	2.601484372	4.010591484

Figure 2: Table 3.

out parameters between highest and the lowest of the values. Concluding that adding ore FUs and higher instruction width generates performance.

Conclusion

Conditional and unconditional branch instructions remain as a critical issue of scalar processing. We have seen that the different branch prediction alorithms have different effect and certian beneftis over other. Lastly, conluding this, the general idea is that once a processor has more resources to use for parallel execution, the better the perfroamnce will be. As seen in *Task 2*.