DAT105 Computer Architecture: Lab 2: Exploring the Impact of Branch Prediction and Instruction-Level Parallelism (ILP) on Processor Performance

Max Villing, Vinaykumar M K, Lab Group 4

September 2021

1 Project Goal

The goal of this project was to gain a deeper understanding on the impact of different branch prediction methodologies and instruction level parallelism had on processor performance.

2 Methodology

In order to accurately determine the impact branch prediction methods as well as hardware configurations have on performance the following methodology will be applied: The simulator simhome will be employed which can execute benchmark programs for different processor configurations. By comparing the results for different set ups it will be possible to observe the performance impacts of different forms of branch prediction. Additionally by configuring simhome to simulate processors with more Functional Units (FUs) it will be possible to see the impact instruction level parallelism has. Performance will be measured in terms of Execution Time (in cycles) as well as Cycles Per Instruction. Note that the lower these are the better since it means execution is faster. Performance was tested by having the simulator perform 5 different benchmark programs and recording the results for each.
One important addendum. Since the results are obtained by using a simulator it is possible for software errors to distort the accuracy of the reported results. Additionally due to the nature of simulators it is unlikely the reported results would be fully replicated in real world circumstances.

Note that since all results were obtained with the use of a simulator they may not match actual real-world conditions. Software bugs in the simulator may additionally distort results.

3 Observations

3.1 Branch Prediction

During the course of this laboratory the following branch prediction strategies were analyzed: Bimodial, 2 Level and Combined. Additionally simhome allows for the use of "perfect" branch prediction strategy that is intended to represent a processor that never miss predicts a branch. The "perfect" predictor was simulated, alongside a simulation that predicted all branches as being untaken, for the purpose of serving as comparisons to the strategies. Each strategy was tested with default parameters associated with it by simhome as well as on configurations were said parameters were altered. The following alternative configurations were tested:

- Bimodial prediction with a doubled table size.
- 2 Level prediction with a doubled size when either the L1 cache or the L2 cache was doubled in size as well as when both were doubled.

- Combined prediction with a doubled table size.

With the "perfect" prediction model, the untaken predictor as well as the base and altered configurations a total of 10 simulations were executed. Their results for the benchmark programs are recorded in the table below.:

	Application	dijkstra	qsort	stringsearch	gsm-untoast	jpeg-cjpeg
	Instruction Count*	54881769	41898644	300884	11704482	27259353
untaken	Execution Time	622626718	1065036797	53443157	39759487	192893984
	CPIbase	11.3449	25.4194	17.7582	3.3969	7.0762
Perfect	Execution Time	581254448	1024485121	5315342	32206374	181744356
	CPIbase	10.591	24.4515	17.6658	2.7516	6.667229263
bimod	Execution Time	550241160	968217974	4888420	32226125	170332108
	CPIbase	10.02593703	23.1085754	16.24685925	2.753314927	6.248574865
bimod - double	Execution Time	550244565	968218157	4888819	32225101	170823726
	CPIbase	10.02599907	23.10857977	16.24818535	2.753227439	6.266609703
2level	Execution Time	550204188	947978434	4909070	32305345	171133034
	CPIbase	10.02526336	22.62551585	16.31549036	2.760083274	6.27795656
2level- double	Execution Time	561335424	948346432	4942841	32325325	171471506
- l1 cache	CPIbase	10.22808547	22.6342989	16.42772962	2.761790312	6.290373289
2level- double	Execution Time	550159503	947747105	4888245	32249302	171000056
-l2 cache	CPIbase	10.02444916	22.61999469	16.24627764	2.755295108	6.273078308
2level- double	Execution Time	550259168	947839128	4925251	32253349	171188141
-l1 and l2 cache	CPIbase	10.02626515	22.62219102	16.36926856	2.755640873	6.279978142
combined	Execution Time	550074570	947933667	4893694	32245903	170885400
	CPIbase	10.02290159	22.62444739	16.2643876	2.755004707	6.268872192
combined	Execution Time	550074458	947933863	4893697	32246102	170883186
-double the table size	CPIbase	10.02289955	22.62445207	16.26439758	2.755021709	6.268790972

Based on these results we observe that 2 Level prediction with a doubled L2 cache generally has the best performance for all benchmarks, though other methods may have superior performance for certain programs.

We also note that the "perfect" strategy did not have the best performance in all categories. Based on consultation with the lab assistants we determine that this is likely due to a software error in simhome.

4 Impact of Functional Units and Instruction Level Parallelism on Performance

After determining an optimal branch prediction strategy we moved on to analyzing the impacts adding more Functional Units to a processor has on performance.

4.1 Addition of Functional Units

Here we tested the impact adding additional Algorithmic Logic Units (ALU) as well as Integer Dividers/-Multipliers (mults) had on performance without any other changes being made. Our results can be seen in the table below.

	Application	dijkstra	qsort	stringsearch	gsm-untoast	jpeg-cjpeg
	Instruction Count*	54881769	41898644	300884	11704482	27259353
2level- double	Execution Time	550159503	947747105	4888245	32249302	171000056
l2 cache ALU -1	CPIbase	10.02444916	22.61999469	16.24627764	2.755295108	6.273078308
2level- double	Execution Time	550159503	947747105	4888245	32249302	171000056
l2 cache ALU -2	CPIbase	10.02444916	22.61999469	16.24627764	2.755295108	6.273078308
2level- double	Execution Time	550159503	947747105	4888245	32249302	171000056
l2 cache ALU -3	CPIbase	10.02444916	22.61999469	16.24627764	2.755295108	6.273078308
2level- double	Execution Time	550159503	947747105	4888245	32249302	171000056
l2 cache Mults -1	CPIbase	10.02444916	22.61999469	16.24627764	2.755295108	6.273078308
2level- double	Execution Time	550159503	947747105	4888245	32249302	171000056
l2 cache Mults -2	CPIbase	10.02444916	22.61999469	16.24627764	2.755295108	6.273078308
2level- double	Execution Time	550159503	947747105	4888245	32249302	171000056
l2 cache Mults -3	CPIbase	10.02444916	22.61999469	16.24627764	2.755295108	6.273078308

As can be observed the amount of ALUs and Mults had no impact on performance. This is because the processor was still set to execute instructions in program order with no attempts made at parallelism. Due to this the processor was unable to use the additional resources and thus no performance changes occurred.

4.2 Instruction Level Parallelism

Here we enabled out-of-order instruction execution for the processor and tested the performance impact of using different sizes for the Reorder Buffer (RUU) and the Load/Store Queue (LSQ). The constraints of the project limited configurations to having RUUs of sizes 16, 32, 64 and 128. Furthermore the LSQ was also constrained to be half the size of the RUU. Our results for the benchmark programs can be seen in the table below:

	Application	dijkstra	qsort	stringsearch	gsm-untoast	jpeg-cjpeg
	Instruction Count*	54881769	41898644	300884	11704482	27259353
2level- double	Execution Time	530768106	645145385	4831500	31578290	145883217
l2 cache RUU -16	CPIbase	9.671118764	15.39776287	16.05768336	2.697965617	5.351675698
2level- double	Execution Time	501130374	500461642	4821095	31420697	132865902
l2 cache RUU -32	CPIbase	9.131090035	11.94457849	16.02310193	2.684501288	4.874139969
2level- double	Execution Time	426309738	423395149	4813930	31896933	121831415
l2 cache RUU -64	CPIbase	7.767784198	10.10522319	15.99928876	2.725189632	4.469343605
2level- double	Execution Time	389920976	387704598	4813267	30449027	109326129
l2 cache RUU-128	CPIbase	7.104745038	9.253392496	15.99708526	2.601484372	4.010591484

We observe that the larger the RUU the higher the performance is. We find this to be logical since the more space there is in the RUU and the LSQ the more opportunities there is for parallel execution and thus the higher the performance will be.

4.3 Functional Units and Instruction Level Parallelism

Here we used the most optimal RUU and LSQ configuration we had found (128 and 64). The goal here was to observe the impact adding more Functional Units (FUs) could improve performance now that instruction level parallelism was possible. Additionally we sought to observe the impact differing instruction fetch and dispatch widths had on performance. The scope of the project restricted us to instruction widths in the range 1-8 and the FUs we could add were ALUs (restricted to 1-8), larger IF Queues (IFQs) (restricted to sizes 2,4 and 8) as well as Mults (restricted to 1-4). Additionally we could allow the program to continue execution of wrong paths even after they were discovered. With many more parameters than before we found it infeasible to merely test all possible processor setups and had to devise a more intricate methodology. We theorized that more resources and higher widths would lead to higher performance. To verify this we decided to test the benchmarks when all parameters were at their highest, as well as the default values (lowest for everything except IFQ was 4). From there we would adjust parameters to be closer to the side with higher parameters until we could draw a conclusion. Our results can be seen in the table below.

	Application	dijkstra	qsort	stringsearch	gsm-untoast	jpeg-cjpeg
	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

We observed that the higher values resulted in better performance. We then tried setting the parameters to be a midpoint between the two and observed that while performance was greater than when we used the lowest values it was still worse than the highest values. We then increased the size of the parameters even more, saw a performance increase but not to the extent of the highest values. From this we conclude that adding more FUs and a higher instruction width leads to better performance. We then tried the benchmarks while enabling wrong path execution, the results of that can be seen in the table below.

2level- double l2 cache	IF Queue - 8,	ALU- 8,	Mul-4,	(Wrong path)	
Application	dijkstra	qsort	stringsearch	gsm-untoast	jpeg-cjpeg
Instruction Count*	54881769	41898644	300884	11704482	27259353
Execution Time	371422826	357227375	4573609	20302139	88080753
CPIbase	6.767690488	8.525988932	15.20057231	1.734561085	3.231212164

We note that performance was better for some benchmarks and worse for others. We found this illogical since wrong path execution should always worsen performance. Consultation with lab assistants led us to conclude that this was likely due to a bug in the simulator and we were recommended to not test more.

5 Conclusion

Based on these tests we have gained a better understanding of processor performance. We have seen that different types of branch prediction algorithms have different effects and that some may benefit certain programs more than others. We have seen that the introduction of instruction level parallelism results in superior performance over sequential instruction execution. We find that in general the more resources the processor has for parallel execution the better its performance will be.