Lab Report 2

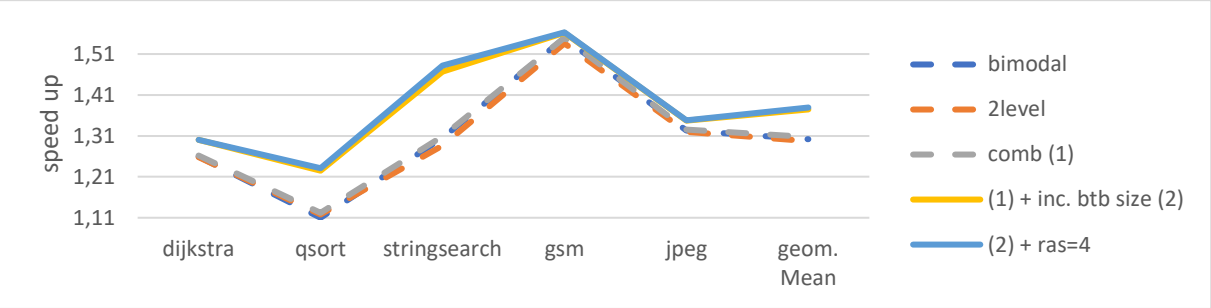
Brief description / used methods

Based on last assignments optimal configuration for the caches, we enable dynamic branch prediction and exploit Instruction-Level-Parallelism. We simulate different branch prediction strategies (bimodal, 2-level and combined), pick the best one and then try to optimize related parameters (table and buffer sizes). Using this new configuration, we increase the number of functional units, enable out-of-order execution and increase the processor width. We start with high resources and lower these step by step to stay close to the maximal speed up while saving costs for additional hardware that does not cause significant speed up. All simulations and speed ups are based on the cache optimal configuration (D-cache: 32kB, 128 Bit block size, 2way; I-cache: 32kB, 128 Bit block size, 4way).

Results and observations

Branch prediction

The base configuration uses "not taken" as static branch prediction (accuracy 49.1%). Enabling dynamic prediction rises the accuracy to over 90% regardless of the used strategy. Comparing the strategies and best/worst case, there is a difference of 5% for "stringsearch" whereas "Dijkstra" BP-accuracy has a variation of only 1.7%. The geometric mean of speed ups is 1.379, although this depends on the actual application. Speed up for "qsort" is only 1.23 whereas "gsm-untoast" has a speed up of 1.56. Doubling the table size for each strategy offers no noticeable performance improvement.



The highest speed up is reached using the "comb" strategy together with a 128-byte 4way associative branch target buffer. Enabling the RAS adds a little speed up, but there is no noticeable difference between sizes of four and eight.

Instruction level parallelism

Enabling out of order execution and increasing the reorder buffer size reduces the number of instructions that need to be stalled and thus, increases speed up. Setting all parameters (see table 1) to their maximum values while keeping processor width equal to 1 results in a speed up 1.17. As increasing the number of functional units only has an effect if we increase the width for issue, decode and commit – we can only process one instruction per cycle and cannot exploit more functional units. The same argument holds for IF queue size. As we can only process one instruction per time, increasing this parameter has no noticeable impact on performance.

Parameter	Max	OPT3	2wide	8wide
IF Queue Size	8	2	2	8
RUU Size	128	128	128	128
LSQ	64	64	64	64
#ALUs	8	1	2	3
#MULTs	4	1	1	1
Issue wrong path	False	false	false	false

Table 1: best configurations for ILP parameters

As we have a branch prediction accuracy of at least 96% per application, allowing wrong path issue and thus, reducing the miss-penalty rather, has a negative impact on performance.

Doubling the size of ROB and LSQ increases performance for Dijkstra (3.5% comparing 128-byte to 16 byte size), qsort (18.9%) and the jpeg application (4.5%), whereas stringsearch and gsm have no noticeable speedup

Enabling issue wrong path decreases the performance as we have branch prediction enabled and a high accuracy rate.

The biggest speed up was achieved by increasing issue, commit and decode width. This allows to process multiple instructions per cycle in these stages and thus, results in a CPI lower than one (in some cases, see table 2). As we issue multiple instructions, the processor can benefit from more functional units. But adding another ALU (2wide) or two more ALUs (8wide) is sufficient, the applications do not profit from more ALUs or MULT units since speed up stays on the same level.

Furthermore, performance of the 8wide processor can be optimized by increasing the fetch queue parameter to eight. This has no visible impact on the 2wide processor. Although we tested each parameter separately, this is an example of correlation between them. Raising the processor width has a bigger impact if we also add more functional units and extent the instruction fetch queue.

	App	Dijkstra	Qsort	Stringsearch	Gsm-untoast	Jpeg-cjpeg	GM
Single OPT3	CPI	1.43	1.86	1.50	1.14	1.39	1.45
	SP	1.26	1.23	1.10	1.07	1.18	1.17
2 wide	CPI	0.80	1.31	0.87	0.60	0.81	0.85
	SP	2.24	1.99	2.12	2.23	2.34	2.25
8 wide	CPI	0.54	1.08	0.59	0.34	0.55	0.57
	SP	3.36	2.10	2.78	3.64	3.00	2.92

Table 2: Highest reached speed ups (referring to base 3, only reflecting ILP improvements). See table 1 for configuration

Design choices and the reason behind them

As we are supposed to provide an optimal configuration for this lab, we use “comb” dynamic branch prediction strategy with a BTB containing 128 sets and associativity of four. As a RAS of size 8 does not cause noticeable improvements we set it to 4.

To exploit ILP, we see that processor width has the biggest impact on performance – if we adapt the other core parameters to it. So, we use an 8wide processor, with a ROB size of 128, LSQ size of 64. By running simulations with different number of functional units, it turned out, that this processor can exploit three ALUs and one MULT unit. Adding more units is expensive, while we do not see further performance improvements.

Conclusion / Learning outcome

Comparing the best configuration (see above) to the base configuration that we started with, there is a mean speed up of 4.03. The biggest increase of performance is caused by exploiting ILP and increasing processor width, whereas dynamic branch prediction significantly increases the branch prediction accuracy, but causes less speed up than expected (1.38).

Nevertheless, the reached speed up comes at a cost, as we increased buffer sizes roughly by a factor of 8, enabled parallelism for the fetch, decode and commit stages and added multiple functional units. This is a major change / improvement of the processor hardware and will be rather expensive.