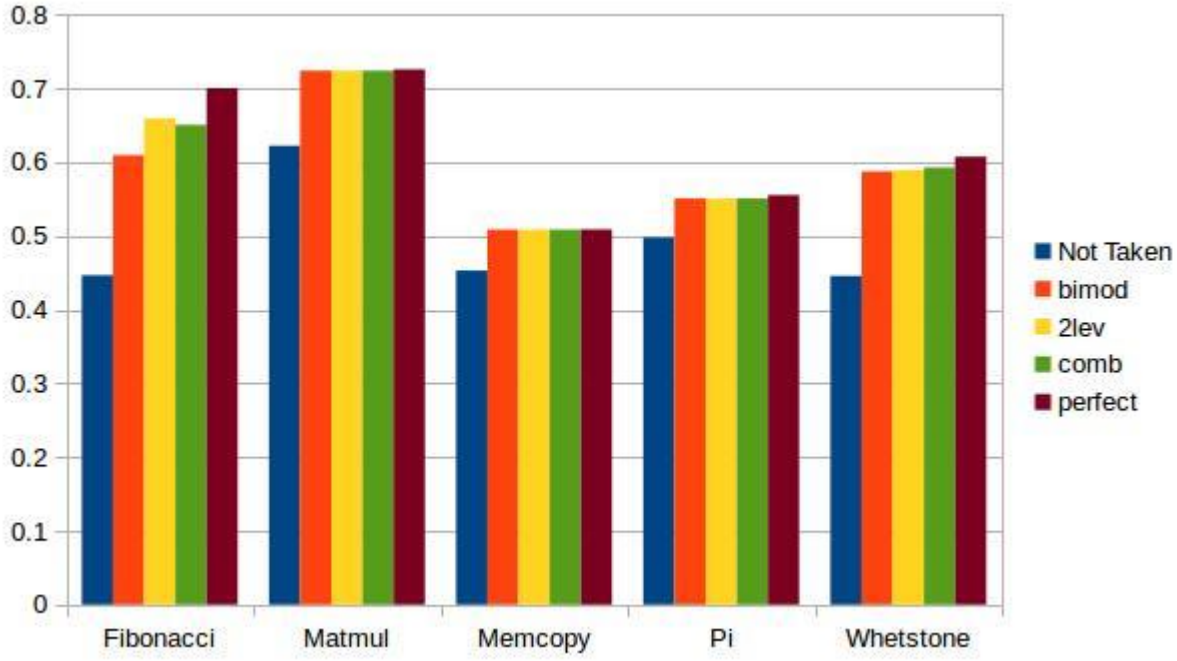
Assignment 3: SIMPLE SCALAR

Sagar Sharma, Teodora Anitoaei, Varun Gowtham

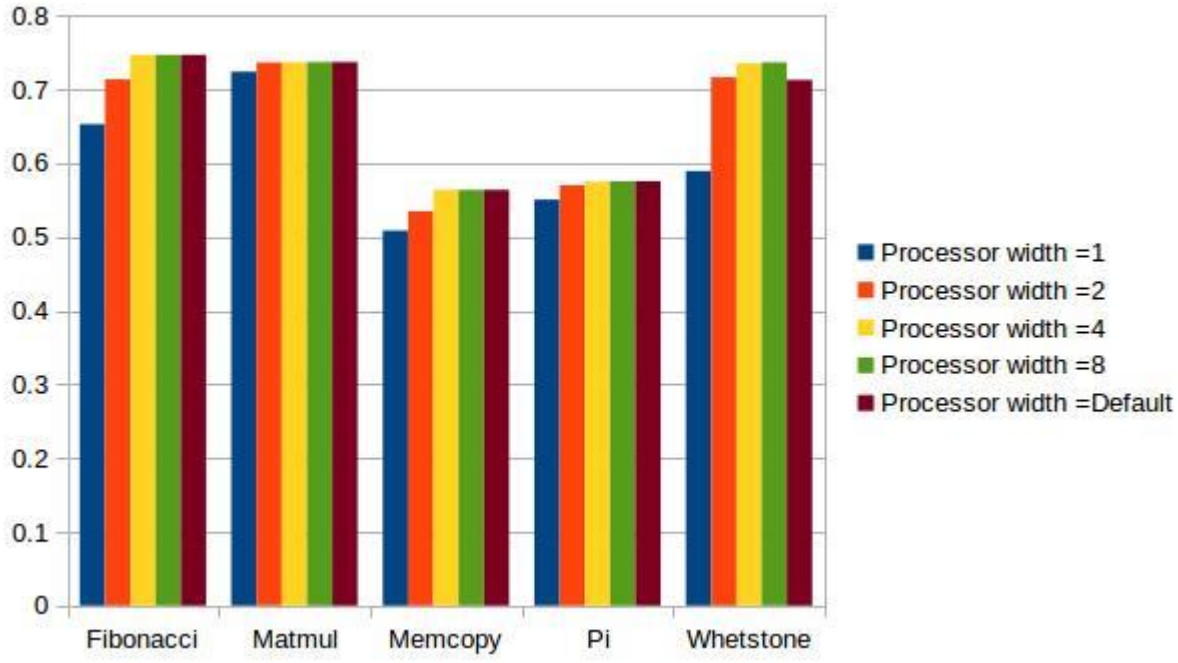
Section 1: Performance evaluation of branch prediction



Bar Graph 1

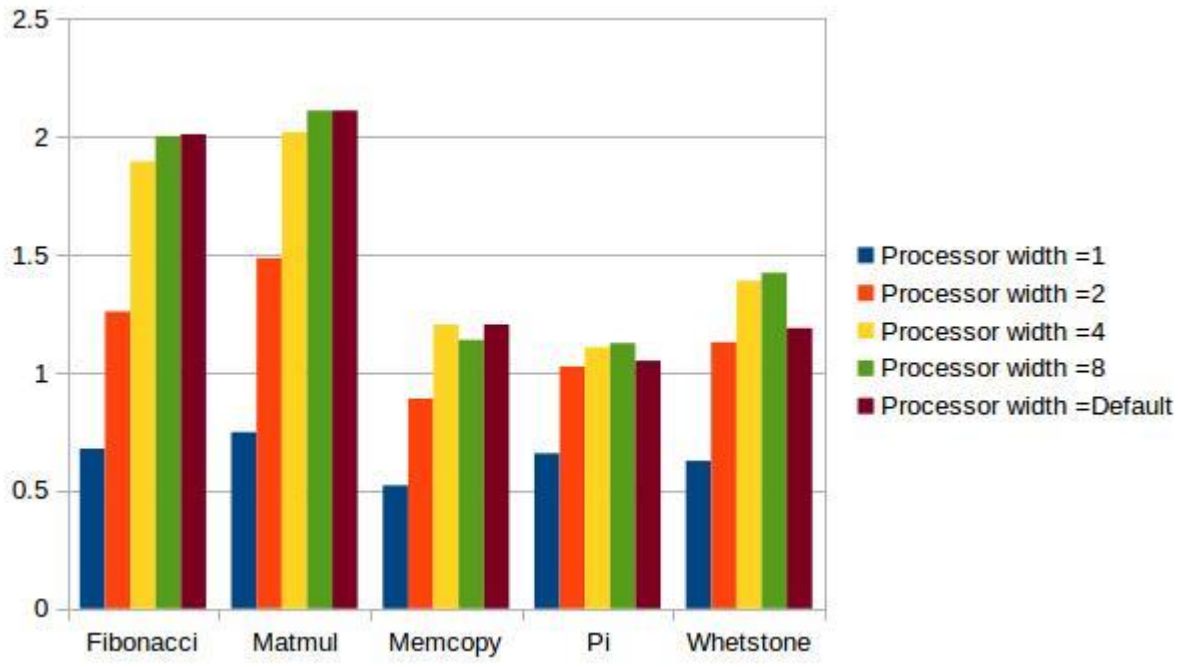
The bar graph represents the experimental results of each benchmark respective to the branch predictor. The X-axis represents the different benchmarks and Y-axis represents the Instructions per Cycle (IPC) respective to benchmark and branch prediction.

Section 2: Performance evaluation of increasing Processor window with In and Out of order execution



Bar Graph 2

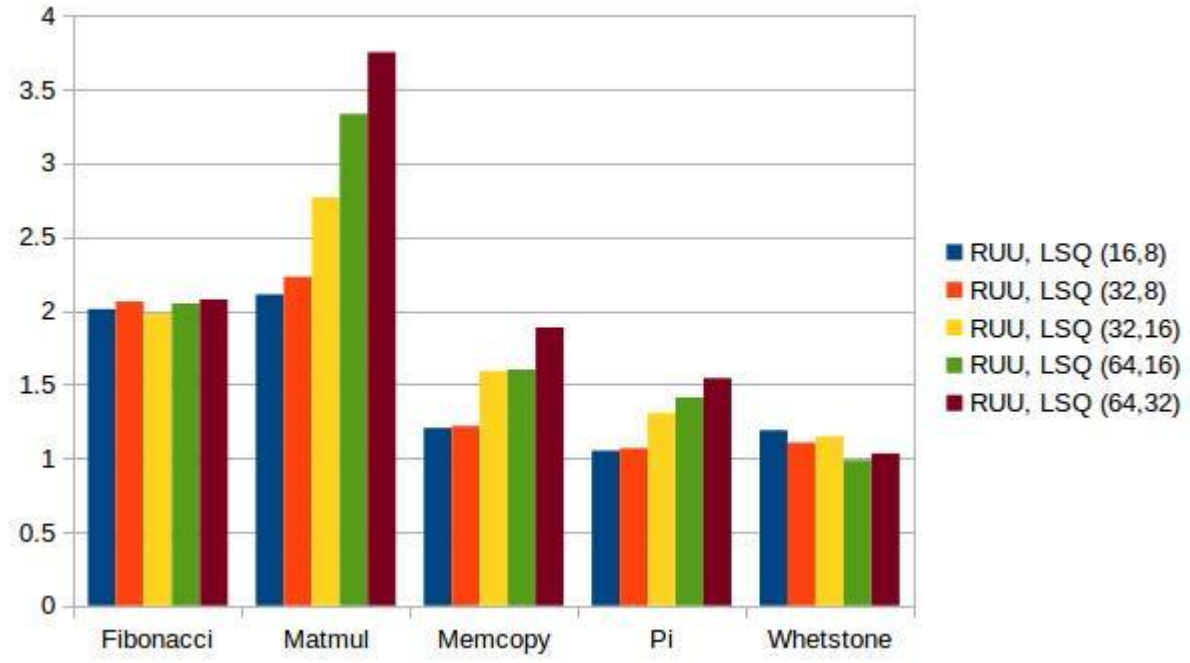
The bar graph represents the effect of increased processor width, in-order-execution with respect to different benchmarks. X-axis represents the benchmarks and Y-axis represent the IPC.



Bar Graph 3

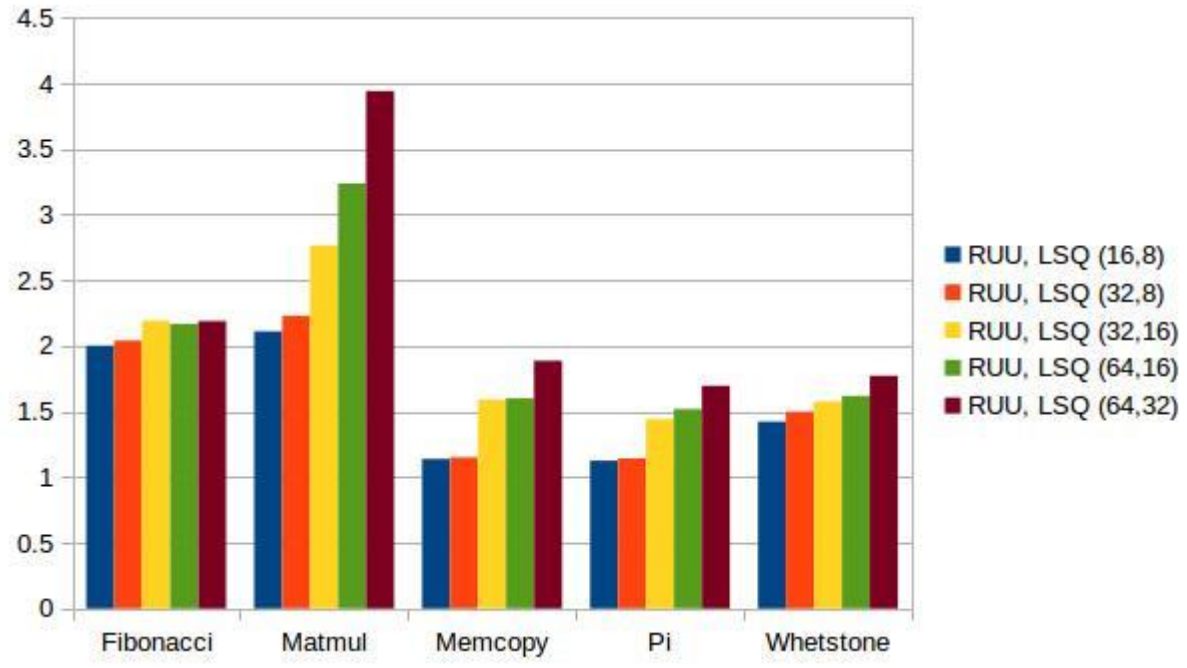
The bar graph represents the effect of increased processor width, in-order-execution with respect to different benchmarks. X-axis represents the benchmarks and Y-axis represent the IPC.

Section 3: Performance evaluation of Register Update Unit (RUU) and Load Store Queue (LSQ) with default and Double the default resources



Bar Graph 4

Bar graph 4 represents effects of increasing the size of register update unit (RUU) and load store queue (LSQ), default amount of memory resources (4/1/4/1) with respect to the benchmarks. X-axis represents benchmarks and Y-axis represents IPC.



Bar Graph 5

Bar graph 5 represents effects of increasing the size of register update unit (RUU) and load store queue (LSQ), double default amount of memory resources (8/2/8/2) with respect to the benchmarks. X-axis represents benchmarks and Y-axis represents IPC.

Section 4: Evaluation of performance of benchmark results from previous sections

In this section, we try to explain our findings on the performance based on the experimental results:

Evalution of Part1:

"Not taken" does not perform well in all cases since conditions in "for" loops and "if" statements would definitely need to be evaluated to make way for execution of the program.

Fibonacci: We observe that 2-level considerably performs better beacuse there is successive 2 stage if statements to be evaluated.

MatMul, Memcopy and Pi: Here we see that the performance of prediction methods hit a wall as, there is successve branches to be taken to evaluate for loops for correct execuction but stalls due to data dependency.

Whetstone: We observe that “Combined” performs slightly better because the program employs part which favours 2 level and part which favours bimodal.

Evaluation of Part2:

Effect of Processor Width andIn-Order execution:

Performace with processor width 1, is low beacuse of the instruction fetch cycle to be taken for each instruction.

Fibonacci: Since its a repetitive evaluation of "if" statements, using higher widths 4 and 8 increases performance compared to 2 but hits wall beacuse there is data dependency between instructions even though we increase the resources corresponding to processor window.

Matmul, Memcopy, Pi: Repetitive code, higher width is favoured but hits performance due to data dependency even though we increase resources corresponding to processor window. Due to the requirement of resources default case of Pi suffers slightly.

Whetstone: Higher width is favoured but due to higher computation, more resources are required. Due to the requirement of resources, default case suffers slightly.

Effect of Processor Width andOut-of-Order execution:

Performace with processor width 1, is low beacuse of the instruction fetch cycle to be taken for each instruction.

Fibonacci: Out-of-Order boosts performance compared to in-order. Processor width 8 graph would stall anyway which does not require the resources and performance increased due out-of-order execution as compared to in-order due to data independence.

Matmul: Out-of-Order boosts performance compared to in-order. Processor width 8 graph would stall because increasing the processor window would help, although performance would reduce after out of order execution could no more be exploited.

Memcopy: Here most of the program is loading but calculating address requires resources, demand for resources reduces the performance. In window of 8, performance slightly suffers due to the high instruction fetch window.

Pi: Same as in Memcopy, default scores less due to less resources.

Whetstone: Due to Out-of-Order execution and requiremement of resources, higher processor widths are favorable. Default case suffers because of less resources is observed.

We observe that Out-of-Order (O-o-O) execution is better than in-Order execution because O-o-O facilitates data independence.

Evaluation of Part3:

RUU-LSQ: Summary of results in:

Case 1: Default resources (4/1/4/1) and

Case 2: Double the default resources (8/2/8/2).

Fibonacci: Higher RUU-LSQ would help in better speculation but increasing HW would not improve the performance.

Matmul: Exploits RUU-LSQ very well due to better speculation and in Case 2, added HW would further increase the performance.

Memcopy and Pi: Performance in both cases remains the same due lesser speculation capability even though resources are increased in Case 2.

Whetstone: Higher RUU-LSQ would be helpful however due to resource deficiency in Case 1, we see an increasing curve in graph of Case 1. With added resources in Case 2, effect of higher RUU-LSQ is boosted.

Section 5: Design of cache configuration to reduce Average Access Memory Time (AMAT)

IPC for initial memory latency of 18 = 1.5894

AMAT for initial memory latency of 18= Hit time of L1+ Miss rate of L1 (hit time L2 + (Miss Rate of L2*Miss Penalty of L2))

1+0.1190(6+(0.1710*18))=2.080282

IPC for initial memory latency of 200 = 0.5430

AMAT for for initial memory latency of 18 = Hit time of L1 + Miss rate of L1(hit time of L2+(Miss Rate of L2*Miss Penalty of L2))

1+0.1190(6+(0.1710*200))=5.7838

To reduce the Average memory access time we try to reduce the miss rate. Reducing miss rate calls for larger block size, larger cache size and higher associativity.

We cannot fully associate cache size because it is expensive in hardware and over all processor rate reduces. Changing cache size reduces the conflict misses and capacity misses as cache size increases as it spreads to provide reference to more blocks.

Larger block size reduces compulsory misses because of the principle locality has two components :

1) temporal locality and 2) Spatial locality.

We propose following cache configurations for UL2 Cache:

1) Proposed UL2 configurations to mask the performance penalty : Number of sets =4096, block size = 512, Associativity = 8 (4096:512:8:l)

- AMAT for proposed UL2 configurations (4096:512:8:l) = Hit time of L1+Miss rate of L1(hit time of L2+(Miss Rate of L2*Miss Penalty of L2) =1+0.1189(8+(0.0058(214)))=1.9798

- Miss penalty = Initial Latency + ((Cache block size/memory bus width)-1)*Subsequent memory chunk = 200+(((512/64)-1)*2)=214
- IPC for proposed UL2 configurations (4096:512:8:l)= 1.7578

2) Proposed cache configurations to mask the performance penalty – Number of sets =2048, block size =512, Associativity=8 (2048:512:8:l)

- AMAT for proposed cache configurations (2048:512:8:l) = Hit time of L1+Miss rate of L1(hit time of L2+(Miss Rate of L2*Miss Penalty of L2)) = 1+0.1189(7+(0.0058(214)))=1.9798
- IPC for (2048:512:8:l)=1.7578

The reason we have chosen the proposed configurations is because the average memory access time for the proposed configurations is closer to that of the average memory access time of the initial memory latency.