E0 243 Computer Architecture Assignment 01

Jacob James (jacobk@iisc.ac.in)
Kawin M (kawinm@iisc.ac.in)

October 15, 2021

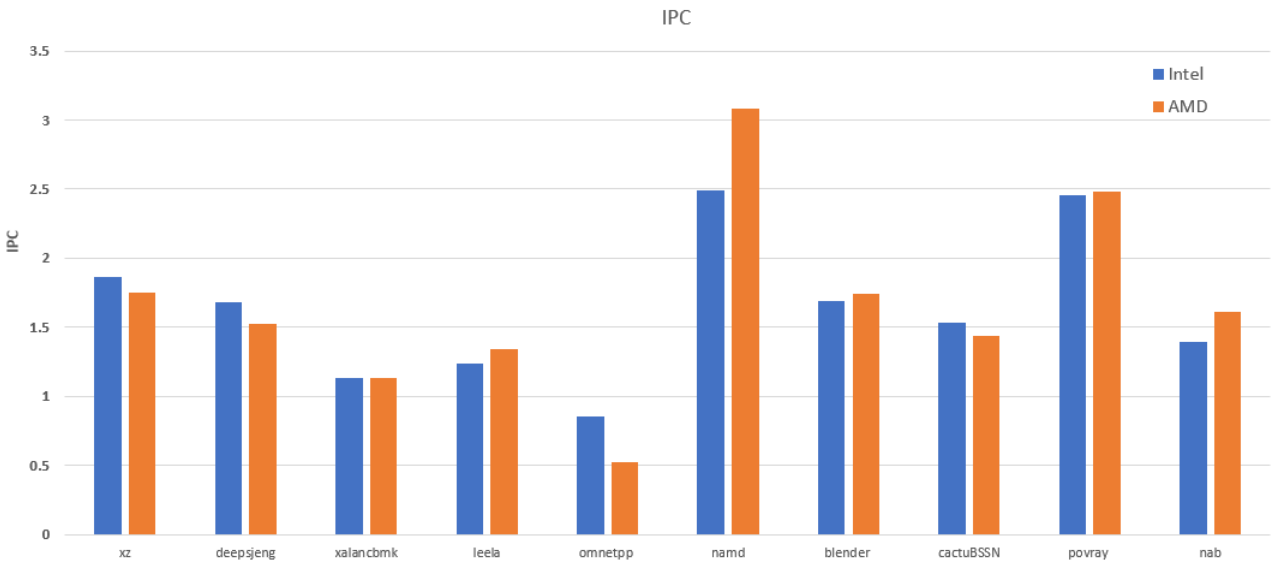
Processors used for evaluation

Architecture	KabyLake	Zen 2
Processor	Core i7	Ryzen 5500U
Frequency	2GHz	2.1GHz
Instruction Cache	32KB 8 Way	32KB 8 way
Data Cache	32KB 8 Way	32KB 8 Way
Total L2 Cache	256KB 4 Way (Shared)	3MB
Total L3 Cache	8MB (Shared)	8MB

- All the Benchmark programs are evaluated using Reference inputs.

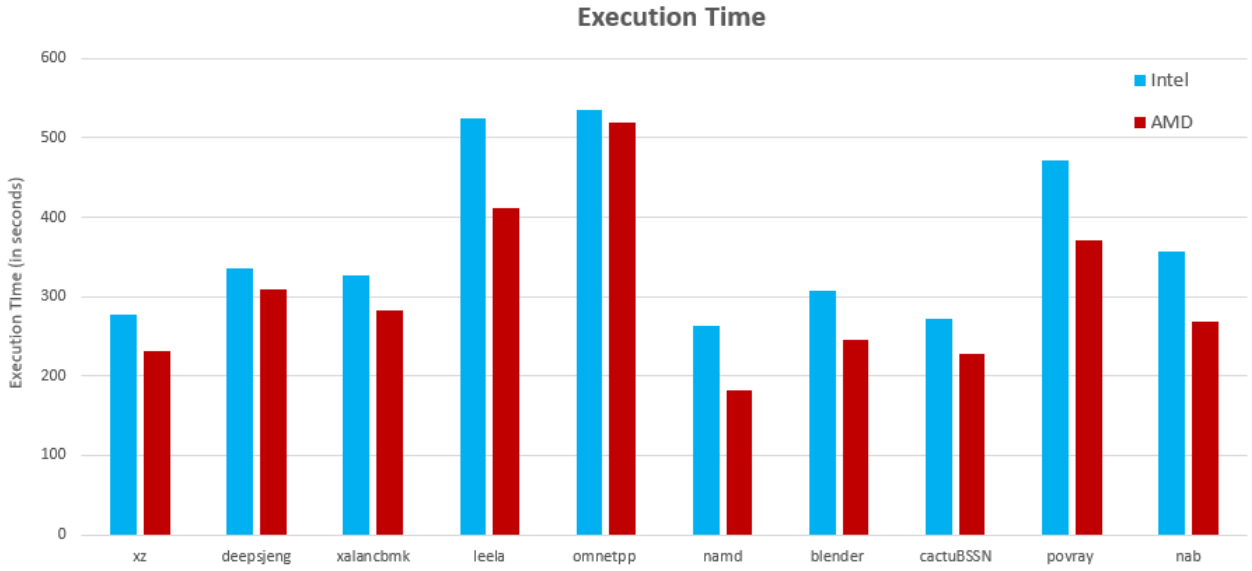
Question 1a)

1 Instructions Per Cycle (IPC) and Execution Time



From observations, Core i7 shows a higher IPC compared to Ryzen in majority of the SPECInt Benchmarks, while it is the opposite for SPEC-FP benchmarks. On average, Core i7 has IPC 18% more than that of Ryzen for SPEC-Int Benchmark and 6% lower IPC than that of Ryzen for SPEC-FP Benchmark.

The reason that Ryzen is able to give a higher performance on Floating point benchmarks could be due to the partitioning of execution units into floating point and integer execution units in Ryzen compared to shared integer and floating point execution unit in Core i7. In Ryzen, the FP execution unit contains its own dedicated scheduler, register file, and renamer. Also, Ryzen has 4 execution pipelines for executing floating point or vector operation with a 4 issue dedicated scheduler whereas, Core i7 has

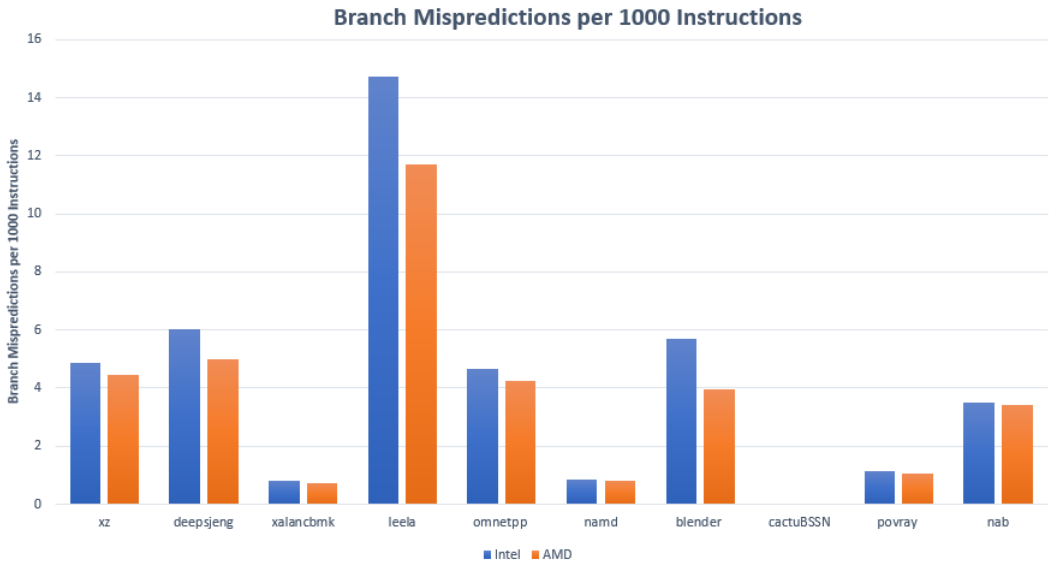


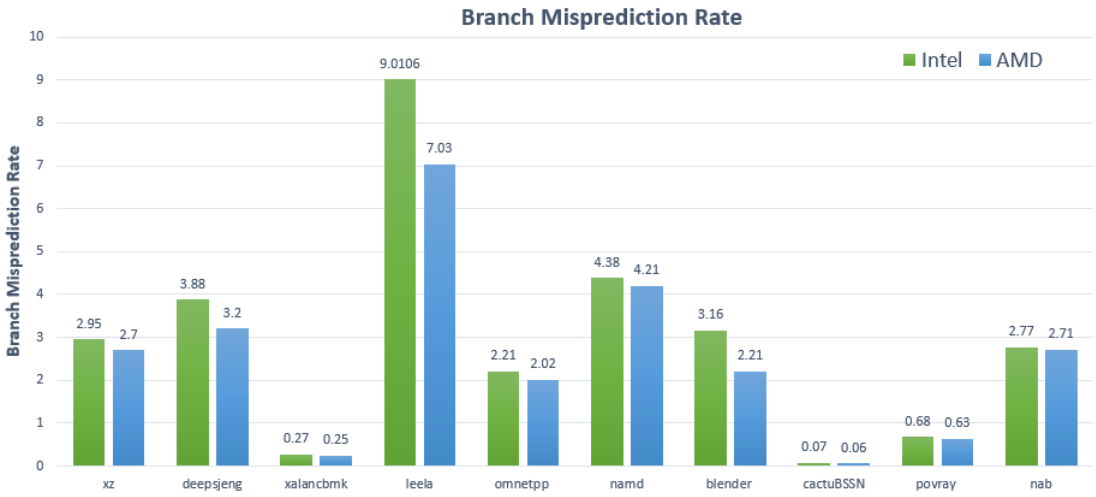
3 shared pipelines for both integer and floating point execution.

Even though Core i7 has a higher IPC for SPEC-Int Benchmarks, Ryzen has the lower execution time for all Benchmark program. On average, Ryzen has 17.63% lower execution time than that of Core i7.

The reason being, KabyLake architecture having a pipeline depth of 13 stages and Zen 2 having a deeper pipeline depth of 20 stages. Also, the KabyLake has process node size of 14 nm, whereas Ryzen 5500U has a process node of size 7nm. Therefore, Zen 2 could lower the clock cycle time without affecting the Power consumption. Thus, even though Ryzen took more number of clock cycles to execute the programs than Core i7, but as Ryzen has a significantly lower clock cycle time (Measured to be around 4 GHz by perf), the execution time in Ryzen is lower than that of Core i7.

2 Branch Misprediction

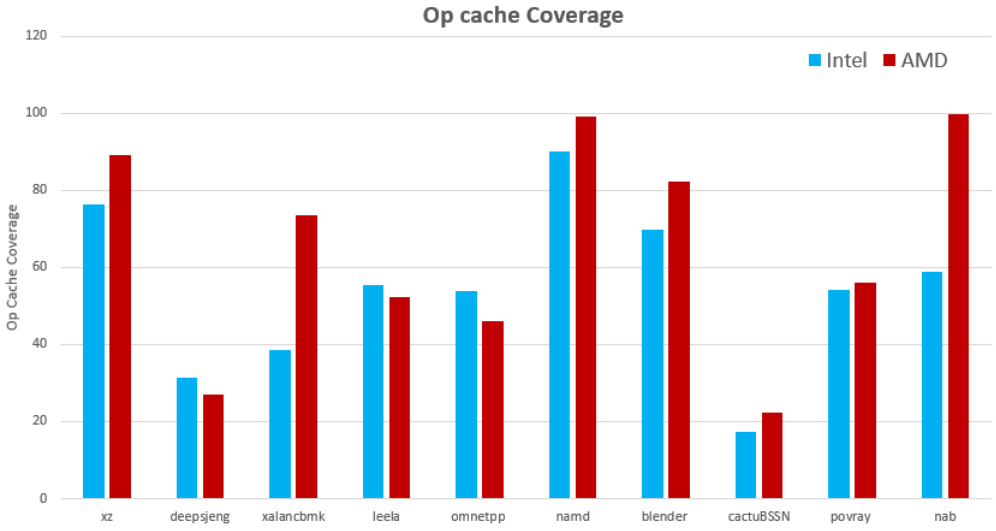




Core i7 has a slightly higher misprediction rate compared to Ryzen. The misprediction rate in worst case is 1.42 times than Ryzen and an average of 14.84% more Branch mispredictions per 1000 instructions than Ryzen from SPEC Benchmarks.

The reason for lower misprediction rate in Zen 2 is because of the introduction of a Tagged Hybrid Predictor (TAGE) in second layer along with a hashed perceptron predictor in BPU. From our evaluation, L2 TAGE predictor in Ryzen lowers the misprediction rate of L1 perceptron predictor by around 14 % to 32 % (Average of 19.84 %) in the Benchmark programs (AMD reports says 30%). Thus, the L2 TAGE predictor in Ryzen has paid off by effectively reducing Branch mispredictions.

3 Micro Op Cache Coverage



MicroOp Cache Coverage represents the fraction of microops delivered by the Microop Cache.

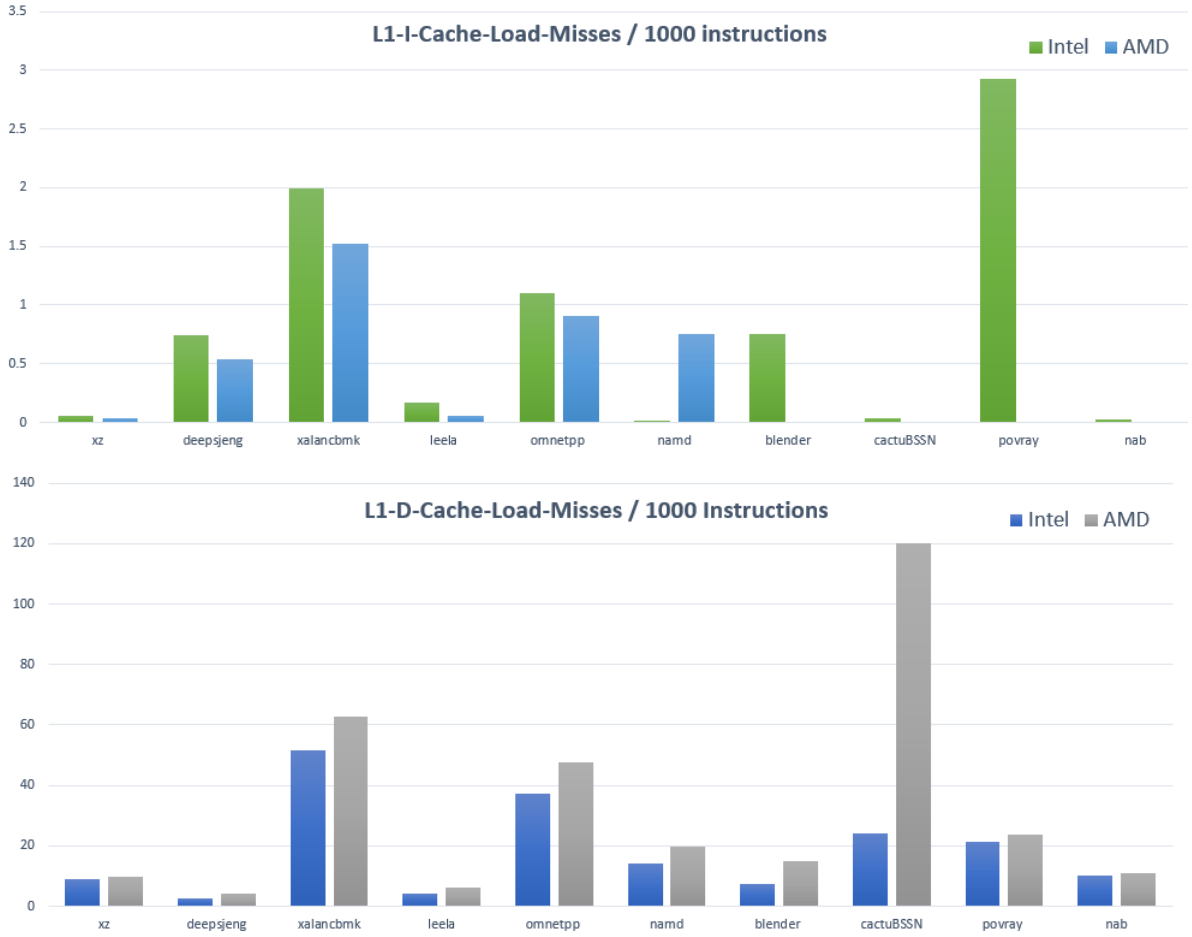
	KabyLake	Zen 2
Op Cache Size	32 sets 8 way 6μops per line = 1536 μops	4096 Macro-ops (Including Fused Branch Instructions)

If the microops corresponding to a frequently accessed instruction does not fit into a microop cache line then the OpCache Coverage becomes less. As Zen 2 has an Op Cache more than double the size of the Op Cache in KabyLake, Zen 2 could store more number of frequently accessed macro-ops in it. Thus, the Op Cache in Zen 2 could deliver more macro-ops to the Op Queue.

For the Benchmark programs deepsjeng, leela and cactuBSSN, Core i7 has better Op Cache coverage than Ryzen. On observation, these three Benchmark programs has high percentage of branch instructions and also wasted work of these Benchmark programs in Ryzen is relatively very larger than that of in Core i7. So, the possible reason could be the enhance op cache algorithm in Intel, because Intel has been using an Op cache for several generations since Conroe. But AMD has introduced the op cache with the introduction of Zen microarchitecture. So even though Intel's Op cache is small, it is highly optimised and performs better at several occasions.

The advantage of an Op cache is to both increase the throughput of decoded instructions and also decreases the power consumption of the legacy decode path by a large factor. So, even though Zen 2 has a larger Op cache, AMD says, it decreases overall power consumption rather than to increase the power consumption, because we could skip the complex legacy decoder path.

4 L1 Instruction and Data Cache



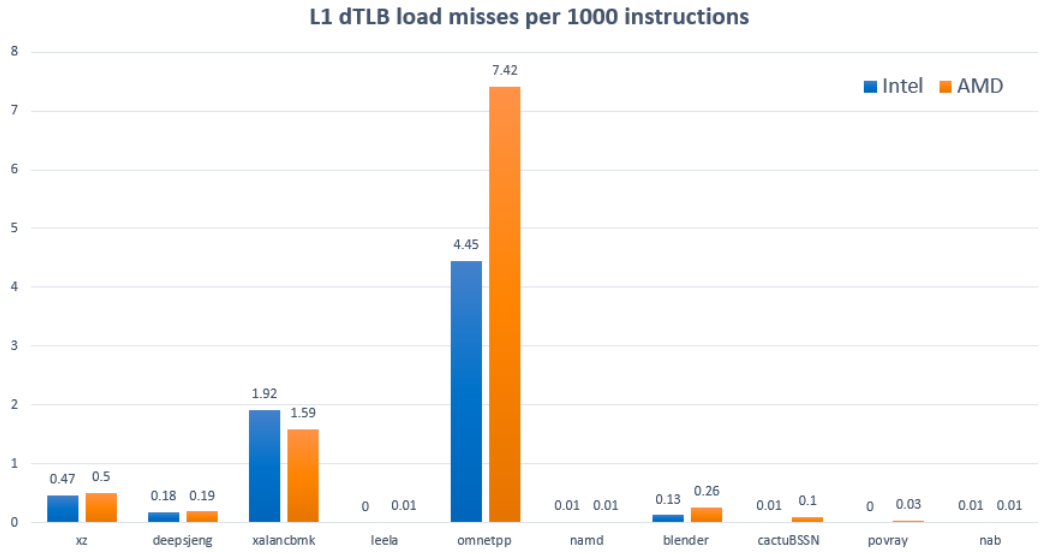
Even though, the cache size (32 KB), associativity (8 way) and cache line size (64 B) are same in both the processors. The cache fetch bandwidth is just 32 B per cycle in Ryzen, whereas Core i7 has a bandwidth double than that of Ryzen 64 B per cycle.

Even then, Ryzen has a lower L1 I-Cache Misses because of the better Branch predictor, that performs

14% better than the one in Core i7 and an efficient I-Cache prefetcher, that prefetches 13 additional consecutive blocks for each fetch request. Thus, the 32B per cycle fetch Bandwidth could be effectively hidden in L1 I-Cache.

It is not the same case with that of L1 D-Cache. Core i7 has a lower number of load misses per 1000 instructions. The latency for cache hit in Core i7 is less compared to that of in Ryzen. The Load-to-use Latency for L1 D-cache is just 4 cycles, whereas, in Ryzen the Load-to-use Latency is 5 cycles for Integers and 8 cycles for Floating points. On the first thought, it seemed like the lower fetch bandwidth and higher latency resulted in higher number of load misses in Ryzen. But on further analysis, the Core i7 has 29% fewer L1 D-Cache access than Ryzen. This could be because of the larger load and store queues in Intel, which could have resulted in a better load to store forwarding. Thus completely avoiding D-Cache accesses many time, so this could have minimised the conflict misses and which resulted in better D-cache miss rate in Core i7.

5 TLB Comparison



dTLBs in both Core i7 and Ryzen almost performs the same. dTLB in KabyLake is split for each page type with 64 entries for 4KB Pages, 32 entries for 2MB pages and 8 entries for 1GB pages. Whereas, dTLB in Ryzen is common for all page types with 64 entries.

This is the reason for high number of L1 dTLB misses in benchmark omnetpp because, it has the highest count of L1 dTLB accesses among all other benchmarks. So, due to smaller size of dTLB in Ryzen, this could have resulted in many conflict misses.

From further observation, shows that, the for benchmark xalancbmk, the dTLB misses is high in Core i7 even though it has a larger dTLB. This is because, in this benchmark, the fraction of dTLB misses are more for 4KB pages, rather than the super pages. Here comes the optimization of dTLB in Zen 2 with page coalescing, which coalesces 4 entries for 4KB pages together. Thus, almost 10% of the dTLB misses got hit in the coalesced pages. Thus, resulting in 17.18% lower L1 dTLB misses per 1000 instructions in Ryzen.

Question 1b)

Analysis was done on Intel KabyLake Processor

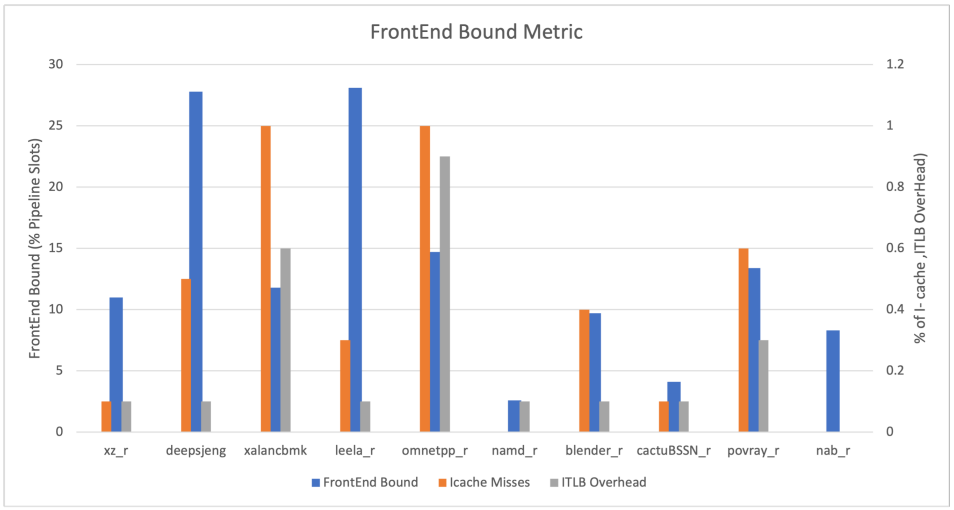
Front-end Bound Metric

Front-end Bound	itlb_misses.miss_causes_a_walk, itlb_misses.stlb_hit, itlb.itlb_flush, branch-misses, icache_16b.ifdata_stall, frontend_retired.latency_ge_4, frontend_retired.l1i_miss

The Front-End Bound metric represents the fraction of pipeline slots where the processors frontend(fetch and decode stages) cannot supply enough instructions to the backend (execution stages) even though the backend is available. On analysing the SPEC benchmarks through Vtune, It was seen that overheads due to ITLB misses and ICache Misses were very low (less than 1% of total clock cycles) which doesnt seem to be posing much of an overhead to the frontend. Possible reasons could be

- The high effectiveness of the prefetching logic in bringing the instructions that may be required in future and its address mappings to the L1 I-cache and L1 Itlb.
- The presence of instructions in the Instruction Decode Queue which acts as a buffer to feed the backend may be hiding the latency for instruction cache misses.

Therefore performance counters associated with Instruction cache and ITLB may not be useful for analysis in the context of SPEC benchmarks.

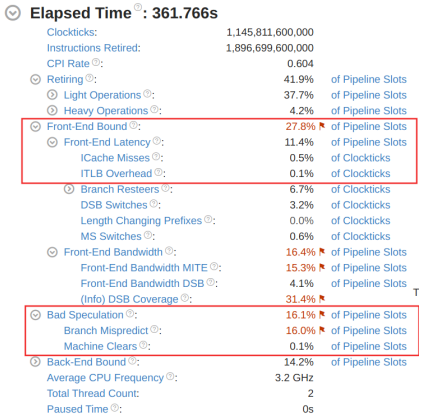


The benchmarks leela_r and deepsjeng_r showed a very high value for the FrontEnd Metric. The percentage of branch misses was high for these benchmarks (Question 1a). There was also considerable wastage of work due to bad speculation. Branch Misses causes the need to flush speculative instructions, leading to delay in the front end to fetch operations from the correct path, thus undersupplying the back end. Therefore, branch misses correlate with FrontEnd-Bound Metric even though other factors, such as inefficiencies due to instruction decoders, can also influence this metric.

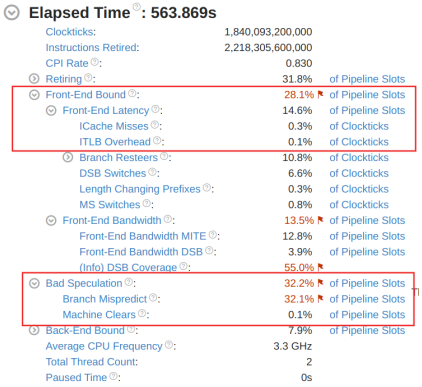
L1 Bound

The metric measures the amount of clock cycles in which the machine was stalled without missing the L1 Data Cache.

1. uops_dispatched_port.port2 : The number of microops dispatched to Load Queue



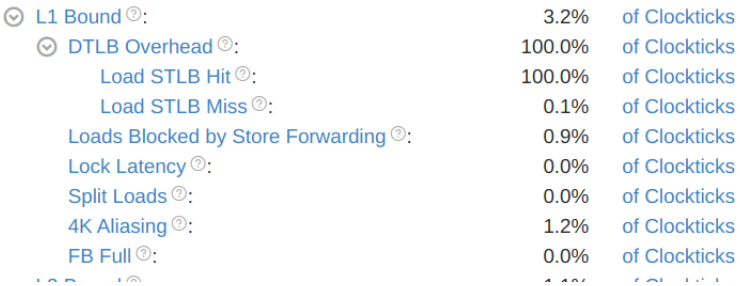
(a) deepsjeng Vtune Analysis



(b) leela Vtune Analysis

2. DTLB_LOAD_MISSES.MISS_CAUSES_A_WALK: The number of translations which faced a miss in all TLB levels.

3. mem_trans_retired.load_latency_gt_4 : counts randomly selected loads with load latecy greater than 4 cycles.



None of the SPEC benchmarks were showing loss of performance due to stall on the L1 Data Cache and, at the same time, expanding the metric in Vtune. among the clock cycles stalled on the L1 Cache, almost 100% of the stalls are due to DTLB overheads (DTLB hits and DTLB misses). Therefore the performance counter 2 is correlated with L1 Bound Metric. Counter 1 will have an impact if less store forwarding happens as load request will be pending in the L1 Cache even if it hits while waiting for an older store but the percentage was low from analysis through Vtune. Counter 3 was 0 most of the time when the programs' analysis was done through perf, which implies loads are faster and store forwarding mechanism is being used effectively.

Port Utilization

The metric measures the fraction of cycles during which execution was stalled on the core due to non divider related operations. This metric becomes high when there is a heavy dependency between instructions in the instruction window or when there is a sequence of operations that overload specific ports.

Port Utilization	bus-cycles, branch-misses, DTLB_LOAD_MISSES.MISS_CAUSES_A_WALK, mem_load_retired.l2_miss, mem_load_retired.l3_miss, uops_dispatched_port.port2

- bus-cycles :

- mem_load_retired.l2_miss : Number of references that missed L2 Cache
- mem_load_retired.l3_miss: Number of references that missed L3 Cache

branch_misses doesnt have an impact on Port Utilization as this counter is more relevant to the microrchitecture frontend. DTLB Misses were found to be less in SPEC benchmarks as most of the translations could be found in the Second Level TLB in the worst case (Previous analysis). On Analysis of the benchmark programs for Port Utilization, L2 Miss Rate and L3 Miss Rate

$$\text{L2 Miss Rate} = (1 - \frac{mem\_load\_retired.l2\_hit}{mem\_load\_retired.l1\_miss}) * 100$$

$$\text{L3 Miss Rate} = (1 - \frac{mem\_load\_retired.l3\_hit}{mem\_load\_retired.l2\_miss}) * 100$$

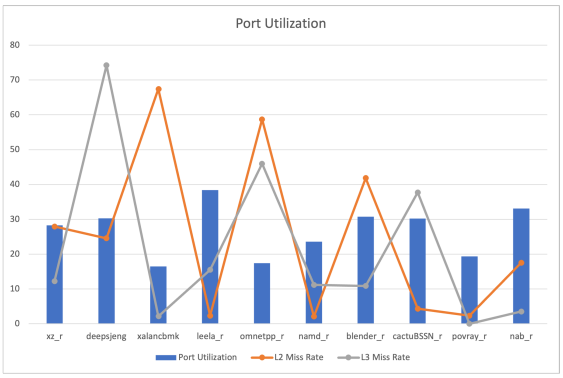


Figure 2: Port Utilization

The port utilization can become higher when loads on the Load Queue miss in the Cache and there might be other instructions waiting for the load operation queued on execution units, increasing port utilization. Similar behavior also happens when there are multiple load instructions queued on Load Queue due to cache misses.

Question 1c)

Wasted work in Benchmark Programs

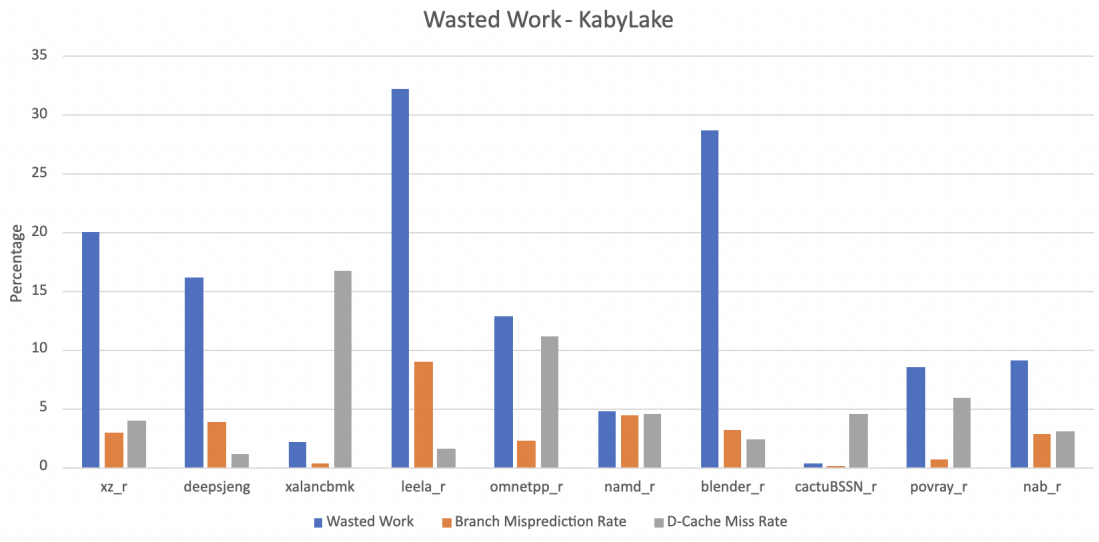


Figure 3: Wasted Work, D-Cache Miss Rate, Branch Miss Rate - Intel core i7 8th Gen

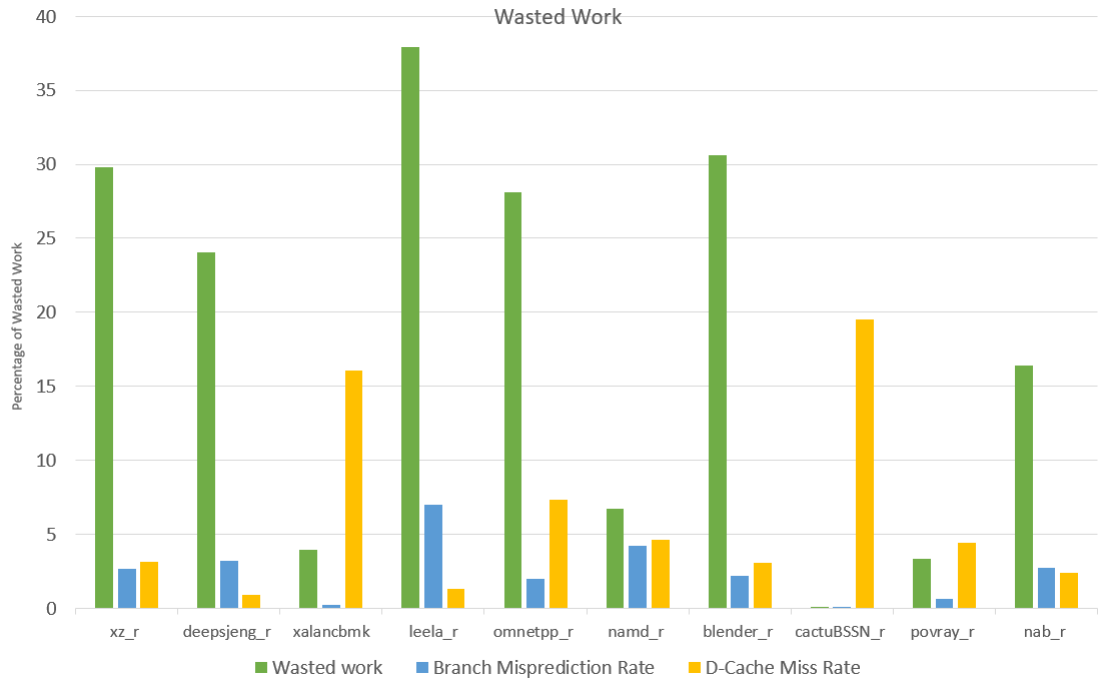
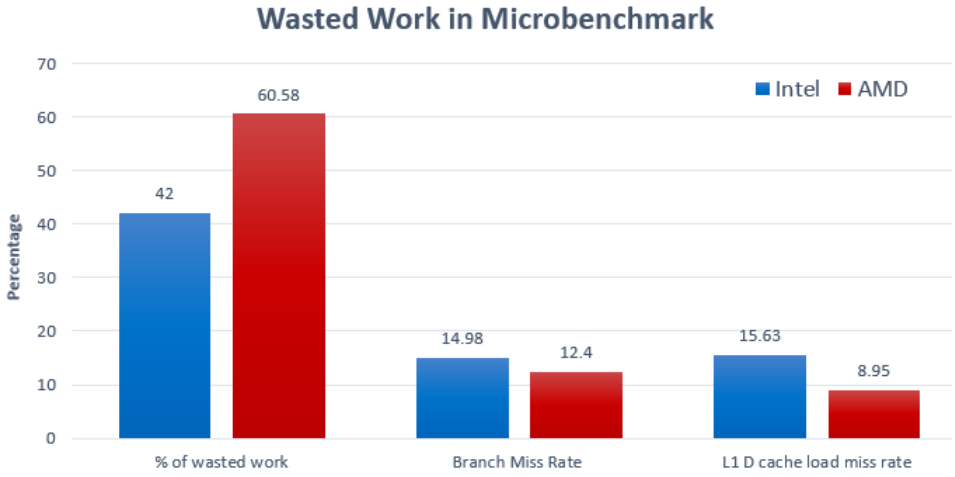


Figure 4: Wasted Work, D-Cache Miss Rate, Branch Miss Rate - AMD Ryzen 5500U

Formula Used:

$$\text{Wasted Work } = \frac{(uops\_dispatched - uops\_retired)}{uops\_dispatched} \times 100$$

MicroBenchmark



Correlation with Microbenchmark:

The Microbenchmark is a single loop with 8 consecutive branch instructions, that if evaluated to be true, writes a data to a buffer. The Microbenchmark also periodically reads a value from a 75x75 matrix in a column major order.

As the two level branches are highly efficient in recognising the patterns in global history. We made sure that the values for the branch conditions are random, so that no pattern can be found by the two levels of Branch Predictors. This is done by generating pseudo random numbers and comparing the i^{th} byte of the number to a value.

The generation of a random number could be done with any standard library, but, it is observed that for increasing the wasted work, the percentage of branch instructions in the program should be high. So there was a need for generating random number with minimum number of x86 instructions[1], as it is called once for each iteration of the loop.

Further increase in number of branches inside the loop or increasing the number of loop iterations, increases the wasted work upto a maximum of 66%. But, also decreases the D-cache miss rate to less than 3-4%.

Also, the data is stored in a 75x75 array, which is accessed in a column major order and made sure that the same element is not accessed. So as to completely avoid temporal and spatial locality, thus, increasing the number of L1 D-cache misses.

On further increase of the size of the matrix, the D-cache miss rate increases upto a maximum of 34% with a 500x500 array. But that significantly increases the number of MOV instructions in the program, which resulted in decrease in the percentage of branch instructions in the program. Thus, lower both the branch mispredictions and the amount of wasted work to around 10-12%.

Correlation with Microarchitecture paramaters:

The amount of work that is wasted depends on both the misprediction rate and the issue rate sustained during the time that the mispredicted branch was followed[2]. So the parameters that has an impact on the misprediction rate and the issue rate, also has the same impact on the wasted work.

Microarchitectural Parameters	Impact on Wasted Work	Reason
Increase in **Branch Misprediction**	Increases Wasted Work	Each Branch misprediction is followed by flushing of pipeline after the branch condition is evaluated. This is the primary reason for existence of wasted work.
Increase in **ROB Size**	Increases Wasted Work	ROB size is the maximum number of operations that can be in-flight in the execution unit. If a branch misprediction is found out, all the following speculative instruction inside ROB has to be flushed. As the ROB size has increased, the number of instructions that is needed to be flushed also increases.
Increase in **Branch Misprediction Penalty**	Increases Wasted Work	Misprediction penalty is the number of cycles it takes to find out the branch is mispredicted. Every cycle, new operations are dispatched to the backend. Thus, increase in this number means increases in dispatch of speculative operations that are wrongly speculated and needs to be flushed.
Increase in **Dispatch width**	Increases Wasted Work	This increases the number of operations issued to the backend. Thus, increase in number of operations that needs to be flushed.
Increase in **L1 D-Cache Misses**	Decreases Wasted Work	With D-cache miss, all the dependent operations on the load operation have to be kept on scheduler. Thus, this increases the Backend bound to a considerable amount. So, the number of operations dispatched are reduced.
Increase in **Cache Miss Penalty**	Decreases Wasted Work	This increases the number of cycles all the dependent operations have to be kept in the scheduler. Thus, this also increases the Backend bound amount. So, the number of operations dispatched are reduced.
Increase in **L1 I-Cache Misses**	Decreases Wasted Work	This decreases the number of available instructions to be decoded. Thus, this increases the Frontend bound to a considerable amount. So, the number of operations dispatched are reduced.
Increase in **Op cache coverage**	Increases Wasted Work	Dispatch unit can then dispatch operations at maximum limit. Thus, increasing the number of operations dispatched.

References

[1] https://lemire.me/blog/2019/12/06/amd-zen-2-and-branch-mispredictions/

[2] Computer Architecture: A Quantitative Approach - Book by David A Patterson and John L. Hennessy