Top-down Performance Analysis on Arm

openEuler Arm SIG Meetup

Yibo Cai

21-Jun-2024

What is Top-down Analysis?

- To solve the pain of optimizing complex applications
- Provides methodology to identify bottleneck on modern CPUs

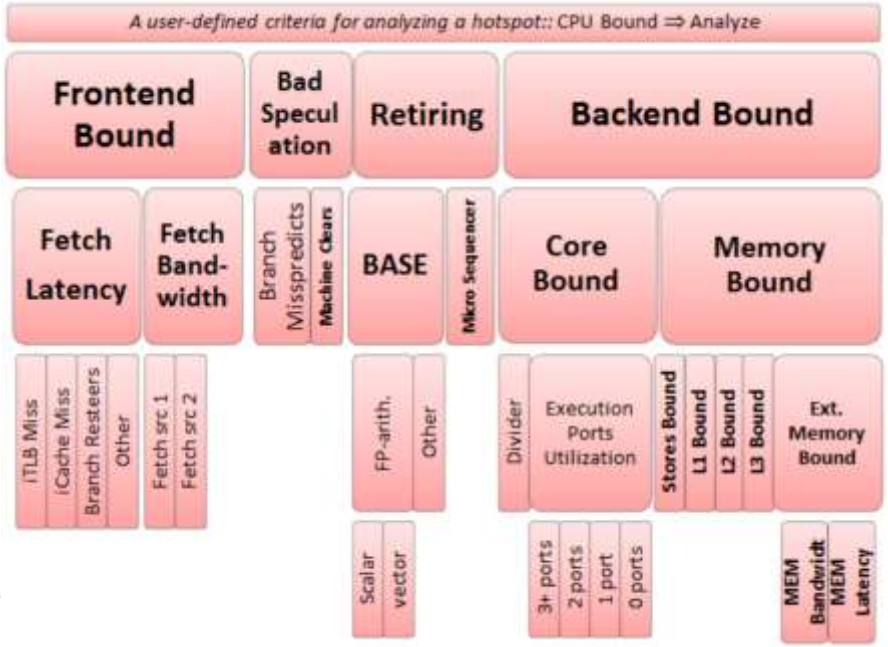
A Top-Down Method for Performance Analysis and Counters Architecture

Ahmad Yasin
ahmad.yasin@intel.com
Intel Corporation, Architecture Group

Abstract

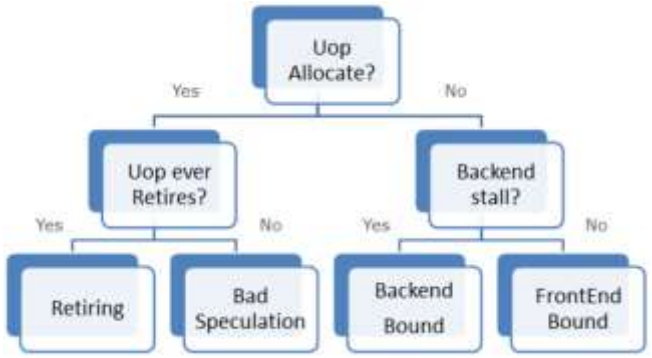
Optimizing an application's performance for a given microarchitecture has become painfully difficult. Increasing microarchitecture complexity, workload diversity, and the unmanageable volume of data produced by performance tools increase the optimization challenges. At the same time resource and time constraints get tougher with recently emerged segments. This further calls for accurate and prompt analysis methods.

In this paper a Top-Down Analysis is developed – a *practical* method to *quickly* identify *true bottlenecks* in out-of-order processors. The developed method uses designated performance counters in a structured hierarchical approach to quickly and, more importantly, correctly identify dominant



Top-down Analysis on x86

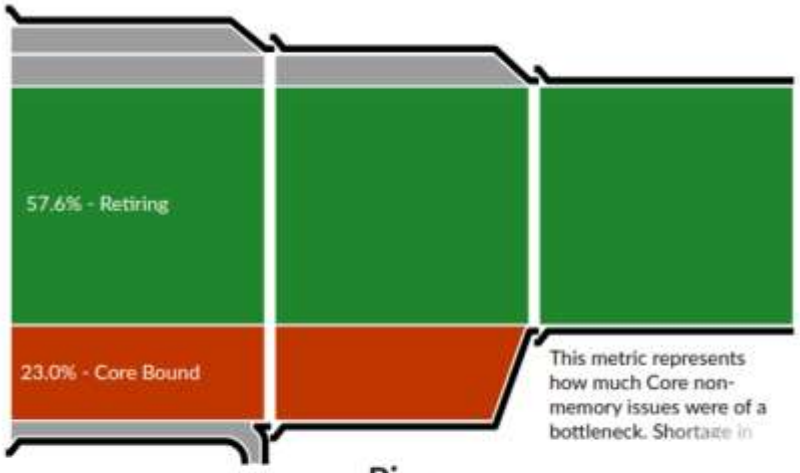
- Multiple levels
- Top level breakdown
  - *frontend_bound*
  - *backend_bound*
  - *bad_speculation*
  - *retiring*



- Tooling: VTune, Toplev

Elapsed Time: 10.429s

Clockticks:	28,209,500,000	
Instructions Retired:	62,514,000,000	
CPI Rate:	0.451	
Retiring:	57.6%	of Pipeline Slots
Light Operations:	54.8%	of Pipeline Slots
FP Arithmetic:	0.0%	of uOps
Memory Operations:	24.2%	of Pipeline Slots
Fused Instructions:	2.5%	of Pipeline Slots
Non Fused Branches:	1.6%	of Pipeline Slots
Nop Instructions:	0.4%	of Pipeline Slots
Other:	26.1%	of Pipeline Slots
Heavy Operations:	2.8%	of Pipeline Slots
Front-End Bound:	6.8%	of Pipeline Slots
Bad Speculation:	4.9%	of Pipeline Slots
Back-End Bound:	30.7%	of Pipeline Slots
Memory Bound:	7.7%	of Pipeline Slots
L1 Bound:	1.5%	of Clockticks
L2 Bound:	0.0%	of Clockticks
L3 Bound:	1.5%	of Clockticks
DRAM Bound:	4.4%	of Clockticks
Store Bound:	0.0%	of Clockticks
Core Bound:	23.0%	of Pipeline Slots
Divider:	4.4%	of Clockticks
Port Utilization:	15.7%	of Clockticks
Cycles of 0 Ports Utilized:	9.1%	of Clockticks
Cycles of 1 Port Utilized:	6.1%	of Clockticks
Cycles of 2 Ports Utilized:	9.1%	of Clockticks
Cycles of 3+ Ports Utilized:	30.3%	of Clockticks
Vector Capacity Usage (FPU):	12.5%	
Average CPU Frequency:	2.8 GHz	
Total Thread Count:	65	
Paused Time:	0s	



µPipe

This diagram represents inefficiencies in CPU usage. Treat it as a pipe with an output flow equal to the "pipe efficiency" ratio: (Actual Instructions Retired)/(Maximum Possible Instruction Retired). If there are pipeline stalls decreasing the pipe efficiency, the pipe shape gets more narrow.

Top-down Analysis on Arm – Level1

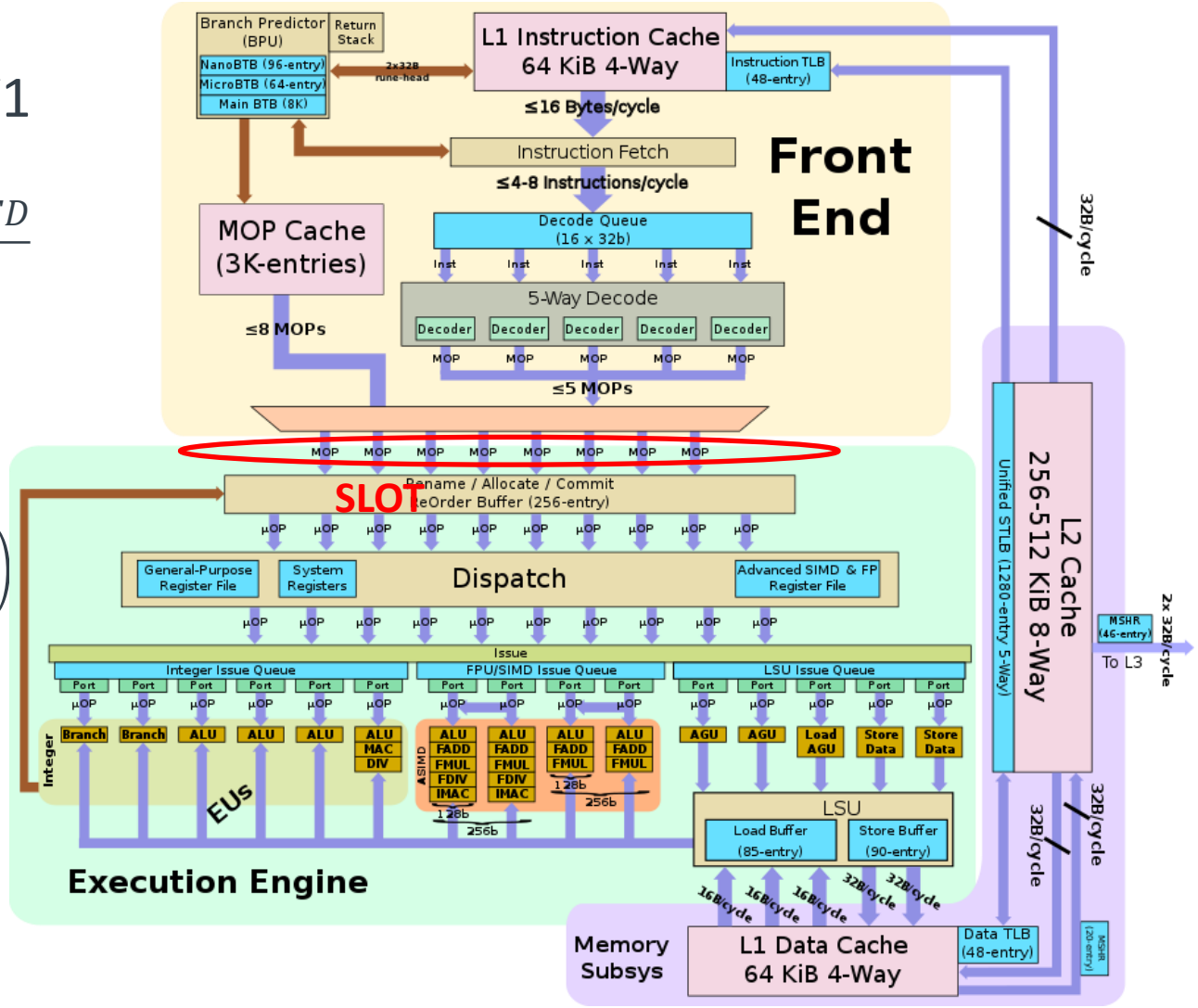
- E.g., Top-down Level1 on Arm Neoverse-V1

$$frontend\_bound = \frac{STALL\_SLOT\_FRONTEND}{8 * CPU\_CYCLES} - \frac{4 * BR\_MIS\_PRED}{CPU\_CYCLES}$$

$$backend\_bound = \frac{STALL\_SLOT\_BACKEND}{8 * CPU\_CYCLES}$$

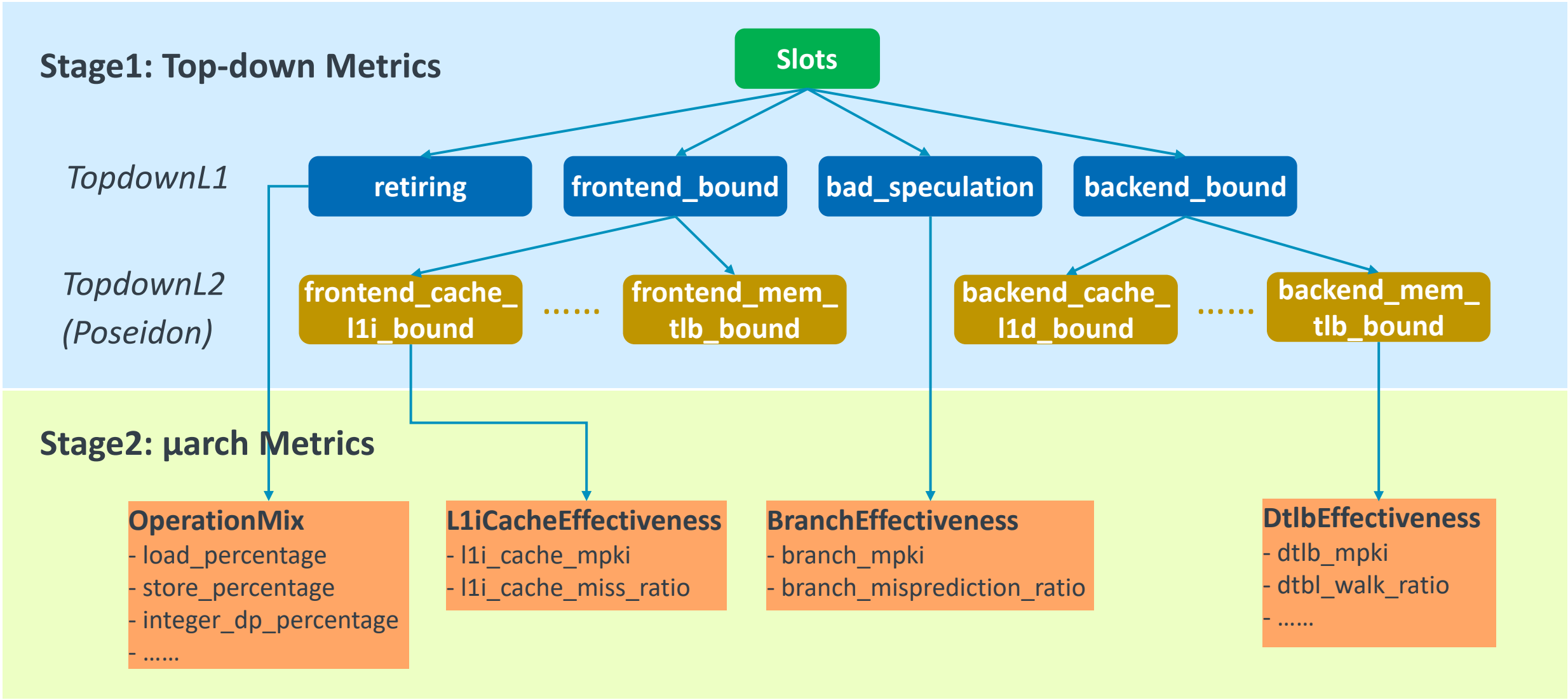
$$bad\_speculation = \left(1 - \frac{OP\_RETIRED}{OP\_SPEC}\right) * \left(1 - \frac{STALL\_SLOT}{8 * CPU\_CYCLES}\right) + \frac{4 * BR\_MIS\_PRED}{CPU\_CYCLES}$$

$$retiring = \frac{OP\_RETIRED}{OP\_SPEC} * \left(1 - \frac{STALL\_SLOT}{8 * CPU\_CYCLES}\right)$$



https://en.wikichip.org/wiki/arm_holdings/microarchitectures/neoverse_v1

Top-down Analysis on Arm – Two Stages



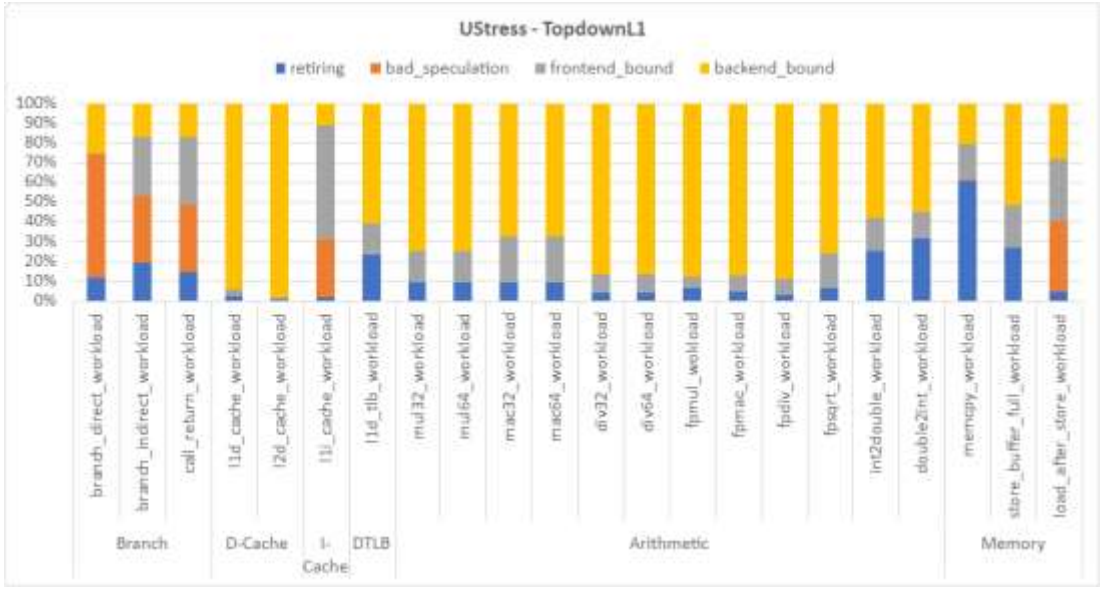
Top-down Analysis on Arm – Tooling

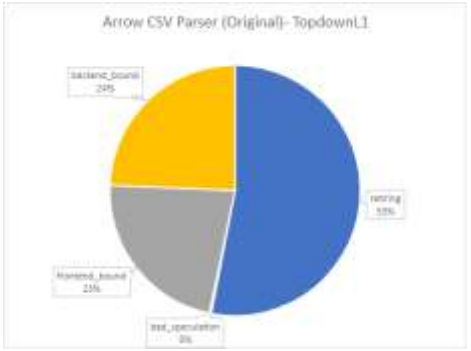
- Linux perf tool

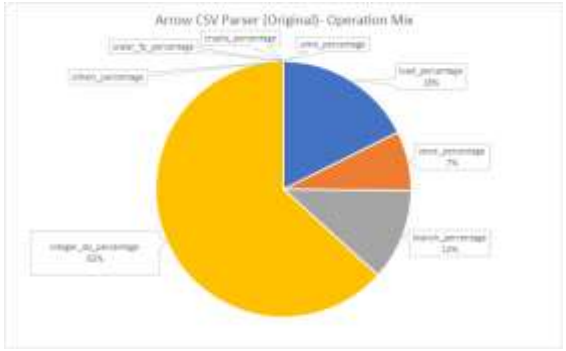
```
$ perf stat -M TopdownL1 -- workload

         31,678 BR_MIS_PRED           # 0.1 percent of slots  bad_speculation
 10,311,676,357 OP_SPEC               # 59.8 percent of slots retiring
 10,336,289,342 STALL_SLOT
  3,441,336,410 CPU_CYCLES
 10,297,795,725 OP_RETIRED
         30,386 BR_MIS_PRED           # 39.8 percent of slots frontend_bound
 10,295,827,924 STALL_SLOT_FRONTEND
  3,441,227,523 CPU_CYCLES
         29,211 BR_MIS_PRED           # 0.2 percent of slots  backend_bound
     40,563,280 STALL_SLOT_BACKEND
  3,438,935,083 CPU_CYCLES
```

Top-down Analysis on Arm – Tooling

Arm Telemetry Solution





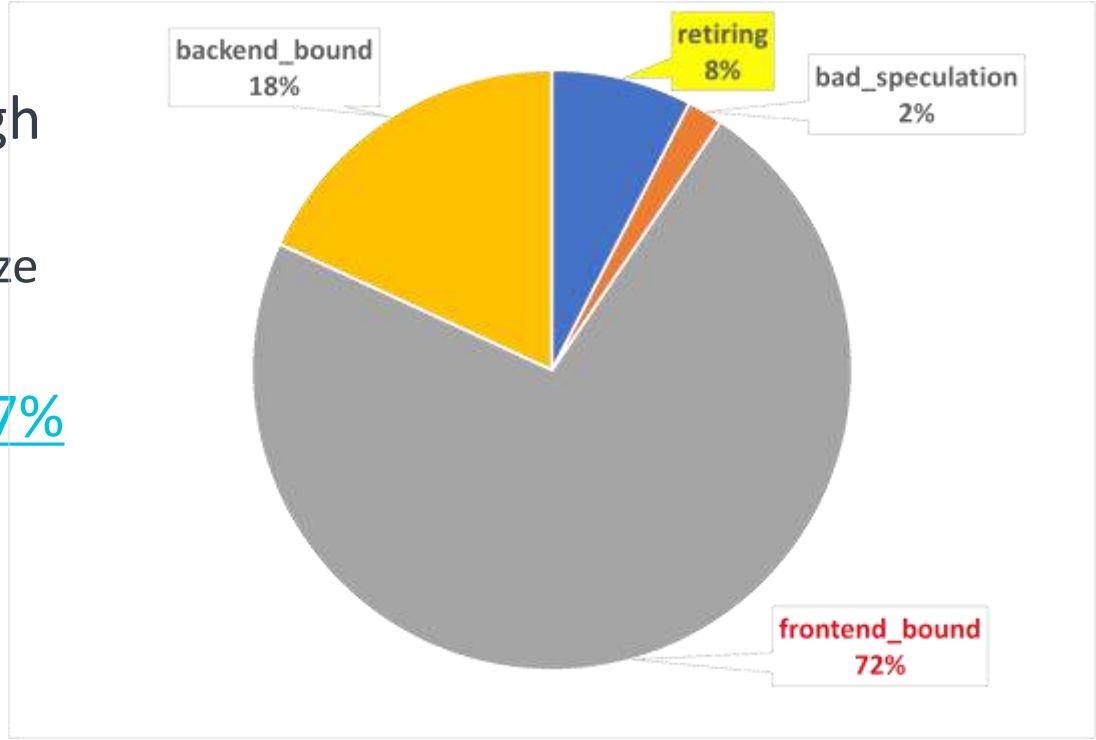



```
Stage 1 (Top-down metrics)
==========================
[Topdown Level 1]
Frontend Bound..................................... 33.91% slots
Backend Bound...................................... 32.67% slots
Retiring........................................... 31.54% slots
Bad Speculation.................................... 0.95% slots

Stage 2 (uarch metrics)
=======================
[Branch Effectiveness]
  (follows Frontend Bound)
  (follows Bad Speculation)
Branch Misprediction Ratio......................... 0.001 per branch
Branch MPKI........................................ 0.186 misses per 1,000 instructions

[Data TLB Effectiveness]
  (follows Backend Bound)
DTLB MPKI.......................................... 0.191 misses per 1,000 instructions
DTLB Walk Ratio.................................... 0.001 per TLB access

[Instruction TLB Effectiveness]
  (follows Frontend Bound)
ITLB MPKI.......................................... 0.008 misses per 1,000 instructions
ITLB Walk Ratio.................................... 0.000 per TLB access

[L1 Data Cache Effectiveness]
  (follows Backend Bound)
L1D Cache Miss Ratio............................... 0.020 per cache access
L1D Cache MPKI..................................... 6.930 misses per 1,000 instructions

…… omitted ……

[Operation Mix]
  (follows Backend Bound)
  (follows Retiring)
Speculative Branch Operations Percentage........ 14.52% instructions
Speculative Integer Operations Percentage....... 37.60% instructions
Speculative Load Operations Percentage.......... 16.61% instructions
Speculative Floating Point Operations Percentage 0.00% instructions
Speculative SIMD Operations Percentage.......... 23.07% instructions
Speculative Store Operations Percentage......... 9.33% instructions
```

Top-down Analysis in Practice – MySQL

- Topdown-tool shows MySQL write-only case suffers from significant frontend-bound and high instruction cache misses
  - PGO/AutoFDO/BOLT are common methods to optimize workloads with large code footprint
- BOLT improves MySQL write performance by 17%

```
$ ./topdown-tool --perf-args "--tid <MYSQL_CLIENT_TID>"
Stage 1 (Topdown metrics)
=========================
Frontend Bound..................... 72.50% slots
Backend Bound...................... 18.13% slots
Retiring........................... 7.58% slots
Bad Speculation.................... 1.93% slots

Stage 2 (uarch metrics)
=======================
[L1 Instruction Cache Effectiveness]
L1I Cache Miss Ratio............... 0.194 per cache access
L1I Cache MPKI..................... 64.789 misses per 1,000 instructions
```

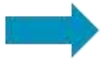


	Branch	L1d Cache	L1i Cache	L2 Cache
MPKI	5.34	19.24	64.79	35.59
Miss	2.60%	4.70%	19.40%	19.80%

Top-down Analysis in Practice – Apache Arrow

- Arrow CSV Parser converts CSV data to Arrow columnar format

```
Name,City,Age
Mike,Shanghai,15
Peter,Hangzhou,14
Jack,Beijing,17
```

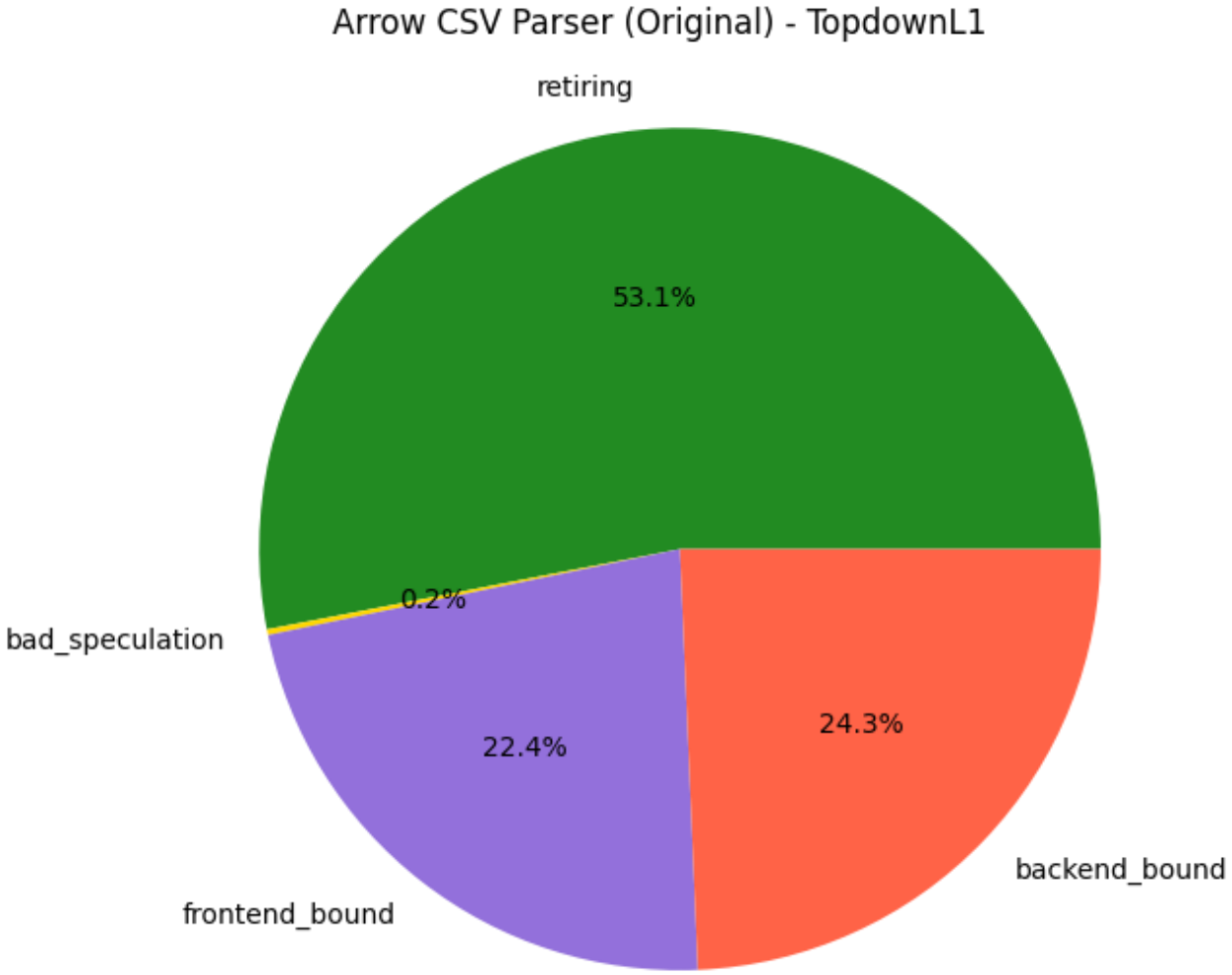


```
Name: "MikePeterJack", [0,4,9,13]
City: "ShanghaiHangzhouBeijing", [0,8,16,23]
Age:  [15, 14, 17]
```

- Evaluate current performance

```
$ perf stat -e cycles,instructions \
      original/arrow-csv-parser-benchmark --benchmark_filter=ParseCSVVehiclesExample

-------------------------------------------------------------------------------------------
Benchmark                                    Time             CPU   Iterations UserCounters...
-------------------------------------------------------------------------------------------
ParseCSVVehiclesExample/iterations:200   10427125 ns     10424094 ns          200 bytes_per_second=1100.14M/s

 Performance counter stats for 'original/arrow-csv-parser-benchmark --benchmark_filter=ParseCSVVehiclesExample'

       5454315340      cycles
      25288198650      instructions              #    4.64  insn per cycle

      2.106576442 seconds time elapsed

      2.086102000 seconds user
      0.020020000 seconds sys
```

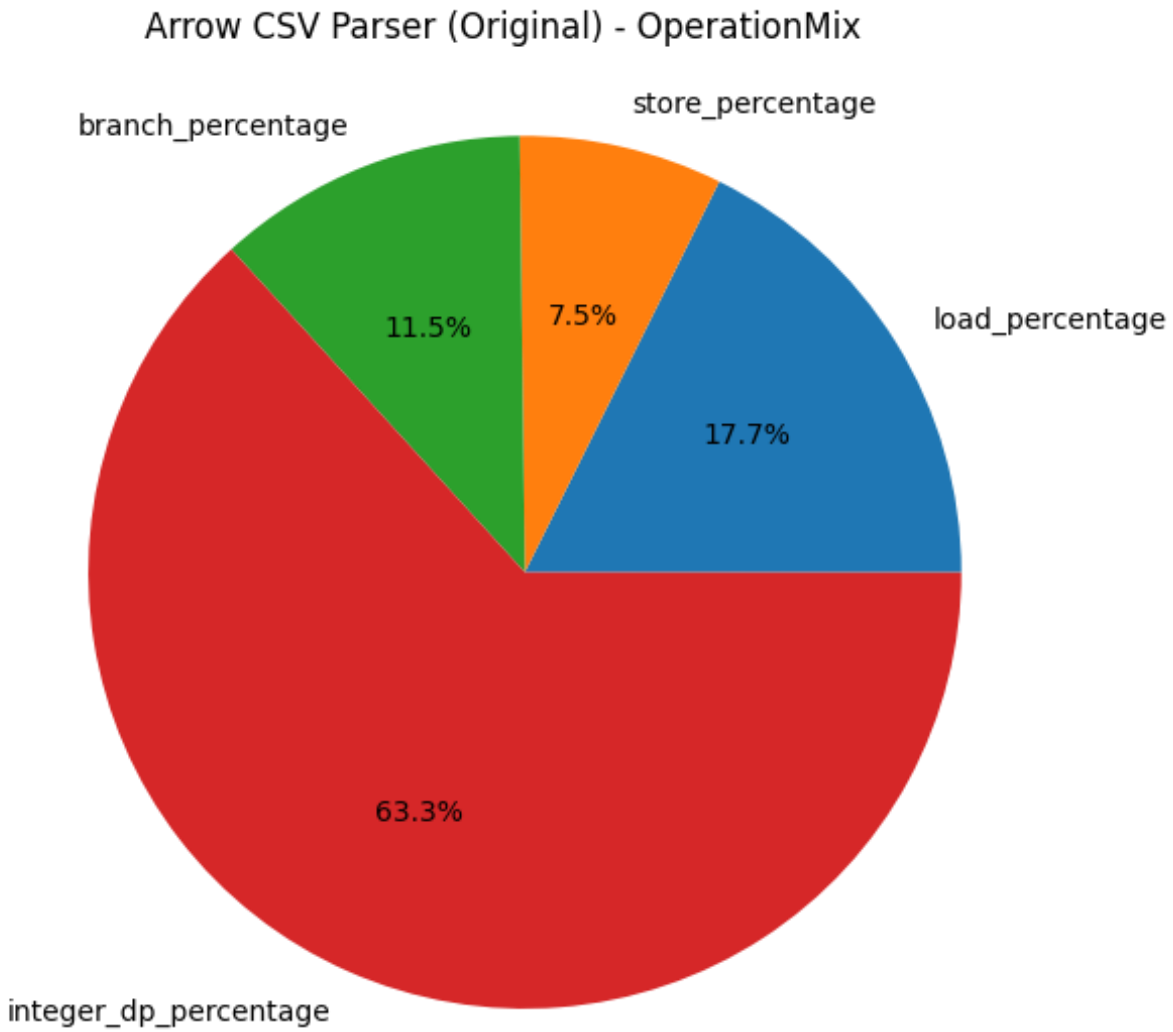
Stage1 – Check the TopdownL1 Metrics



- *Retiring* is above 50%, which matches the high IPC we observed.
- No obvious bottleneck from *frontend_bound* and *backend_bound*.
- For high *retiring* case, we may want to investigate the **Operation Mix** metrics.

arm

Stage2 – Check the Operation Mix



- Observed high volumes of integer calculation (*integer_dp_percentage* ≈ 2/3)
- Heavy integer calculation loading often means chances to leverage SIMD for better data parallelism

arm

Why High *integer_dp_percentage*?

- The issue: special tokens (, \n, …) hurts performance

```
Name,City,Age
Mike,Shanghai,15
Peter,Hangzhou,14
Jack,Beijing,17
```

```
Name: "MikePeterJack", [0,4,9,13]
City: "ShanghaiHangzhouBeijing", [0,8,16,23]
Age:  [15, 14, 17]
```

- Leverage Neon to process CSV data in batch

`Mike,Shanghai,15` *Special tokens found. Go **slow** path. Check char by char.*

`Mike,Shanghai,15` *No special token. Go **fast** path. Process 8 chars at once.*

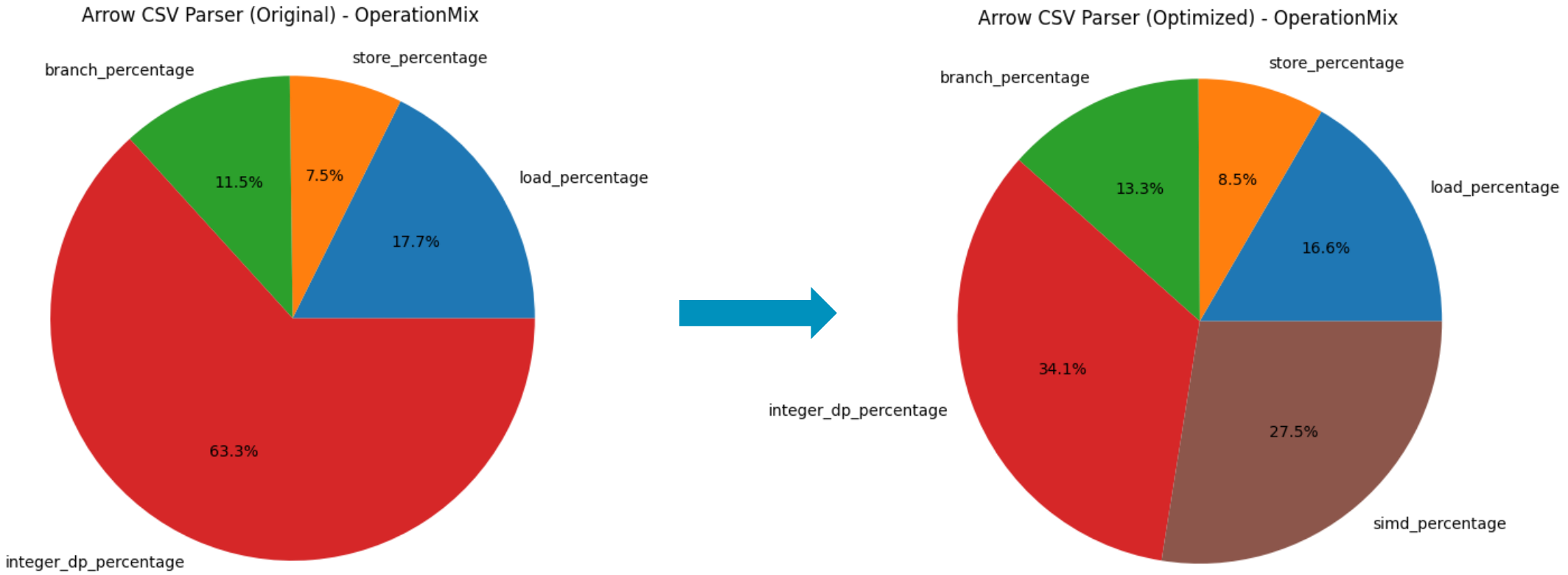
```
bool Matches(uint8x8_t w) {
  v = vceq_u8(w, vdup_n_u8('\r'));
  v = vorr_u8(v, vceq_u8(w, vdup_n_u8('\n')));
  v = vorr_u8(v, vceq_u8(w, delim_));
  v = vorr_u8(v, vceq_u8(w, quote_));
  v = vorr_u8(v, vceq_u8(w, escape_));
  return (uint64_t)v != 0;
}
```

Evaluate the Optimized Code

```
$ perf stat -e cycles,instructions \
      optimized/arrow-csv-parser-benchmark --benchmark_filter=ParseCSVVehiclesExample

-------------------------------------------------------------------------------------------
Benchmark                                         Time             CPU   Iterations UserCounters...
-------------------------------------------------------------------------------------------
ParseCSVVehiclesExample/iterations:200    5642248 ns      5639743 ns          200 bytes_per_second=1.98576G/s

 Performance counter stats for 'optimized/arrow-csv-parser-benchmark --benchmark_filter=ParseCSVVehiclesExample'

        2971634240      cycles
       12771507212      instructions              #    4.30  insn per cycle

       1.154066440 seconds time elapsed

       1.123869000 seconds user
       0.027996000 seconds sys
```

- Performance improves **~80%** from 1.10G/s to 1.99G/s
- Total instructions drops ~50% from 2.53E+10 to 1.28E+10

Compare the Operation Mix



- *integer_dp_percentage* drops from ~2/3 to ~1/3
- *simd_percentage* now occupies over 1/4 total instructions