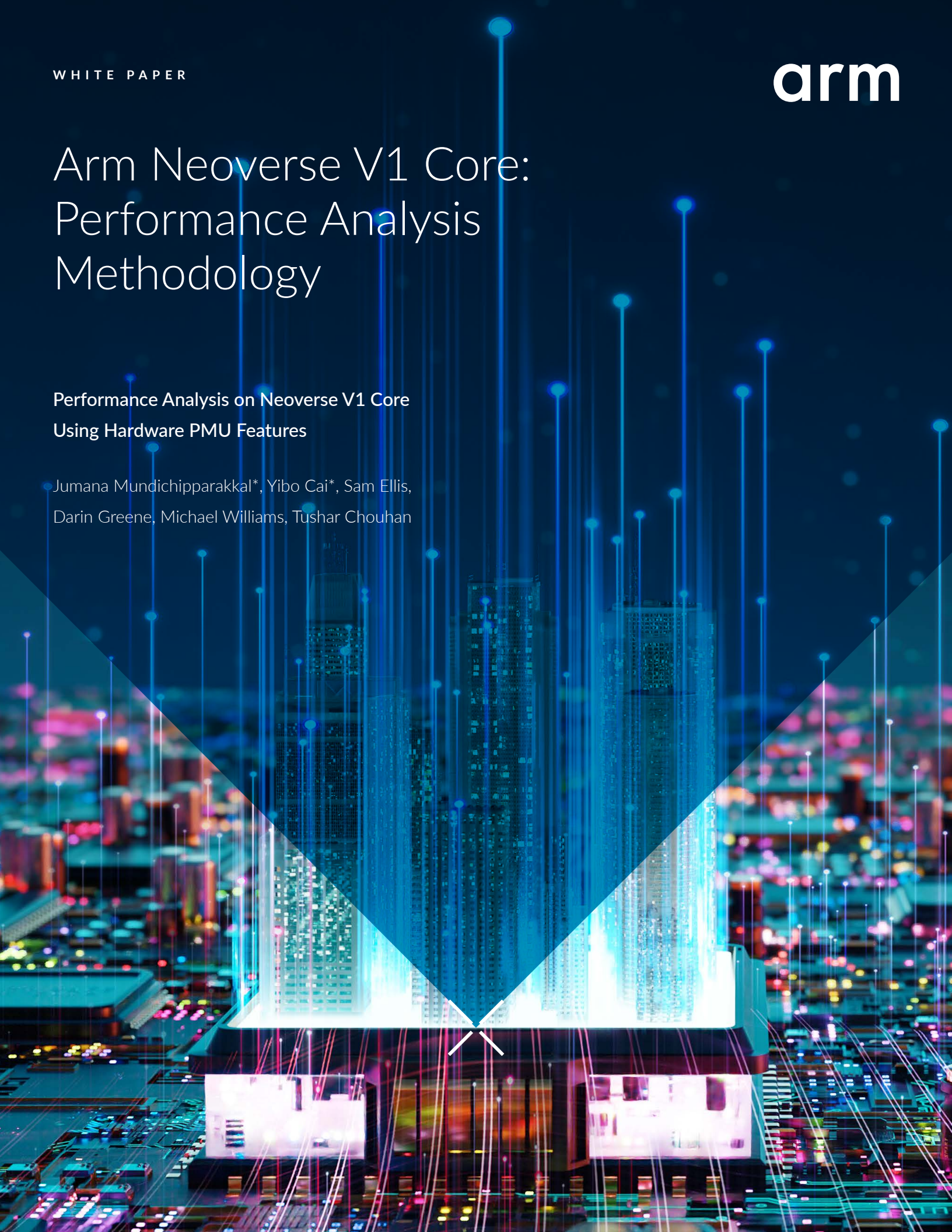
WHITE PAPER

arm

Arm Neoverse V1 Core: Performance Analysis Methodology

Performance Analysis on Neoverse V1 Core Using Hardware PMU Features

Jumana Mundichipparakkal*, Yibo Cai*, Sam Ellis, Darin Greene, Michael Williams, Tushar Chouhan



Contents

9 Appendix B. Neoverse V1 Events (Cont)

B1.11 SPE

B1.12 Spec_Operation

B1.13 Stall

B1.14 General

B1.15 TLB

B1.16 SVE

10 Appendix C. Neoverse V1 Metrics

C1.1 Metric Group: Topdown_L1

C1.2 Metric Group: Cycle_Accounting

C1.3 Metric Group: General

C1.4 Metric Group: MPKI

C1.5 Metric Group: Miss_Ratio

C1.6 Metric Group: Branch_Effectiveness

C1.7 Metric Group: ITLB_Effectiveness

C1.8 Metric Group: DTLB_Effectiveness

C1.9 Metric Group: L1I_Cache_Effectiveness

C1.10 Metric Group: L1D_Cache_Effectiveness

C1.11 Metric Group: L2_Cache_Effectiveness

C1.12 Metric Group: LL_Cache_Effectiveness

C1.13 Metric Group: Operation_Mix

1. Introduction

Neoverse V-series cores are designed to deliver the maximum single-threaded performance available from Arm for cloud and high-performance computing (HPC) workloads. Neoverse V1, the first in this new performance tier, is the first Arm-designed core to support key performance features including Scalable Vector Extensions, bFloat16 and Int8MatMul. Combined with platform capabilities such as DDR5 memory and PCIe Gen5 I/O, the Neoverse V1 platform provides leading performance for cloud, HPC and ML workloads.

This paper outlines a methodology for workload characterization and root cause analysis using the Performance Monitoring Unit (PMU) events on the Neoverse V1 CPU. The intended audience is software developers and performance analysts working on software analysis, optimizations, tuning, and development. The content can also support silicon engineers in selecting the right set of PMU events for conducting system analysis.

Background

This paper is an extension to the previous whitepaper titled "Arm Neoverse Core N1: Performance Analysis Methodology", which introduced the basic performance analysis methodology that can be followed on any Neoverse platform with a set of architecturally common PMU events and derived metrics. As this was the first document on PMU based methodology from Arm, a basic introduction to the Arm Performance Monitoring Unit [5, Chapter 2] and using the Linux perf tool for accessing PMU events and conducting performance analysis [5, Chapter 4] was already covered. In this paper, we focus on introducing new capabilities supported by the Neoverse V1 on top of the existing features in the Neoverse N1, which is the previous generation. It is recommended to read this paper for a basic understanding of the hardware PMU unit and common architectural events, as well as how to use Linux perf based tooling to conduct performance analysis.

Outline

The content of this paper is divided into 4 chapters.

Chapter 2 will introduce the hardware PMU on the Neoverse V1 with a list of the most relevant PMU events for workload characterization.

Chapter 3 will present the workload characterization methodology using the Neoverse V1 core PMU events, including some validation examples.

Chapter 4 will explore a case study on how the performance analysis methodology presented in Chapter 3 was used to conduct code optimization on the Apache Arrow workload.

2. Neoverse V1 Performance Monitoring Events & Metrics

The Neoverse V1 CPU implements the PMU extensions of the Arm v8.4[4] with support for 100+ hardware events. The Neoverse V1 PMU has six configurable counter registers and one dedicated function counter to count CPU cycles.

2.1 Neoverse V1 PMU References

There are three key references for the Neoverse V1 PMU events and other information needed for their perusal as below:

01 Arm Neoverse V1 PMU Guide[2]

02 Arm Neoverse V1 Technical Reference Manual[1]

03 Arm Neoverse V1 Software Optimization Manual[3]

04 Arm Architecture Reference Manual[4]

The PMU events implemented by the Neoverse V1 core are listed in the Arm Neoverse V1 Core Technical Reference Manual [1, Chapter 1]. These events are grouped per CPU function groups with enhanced SW consumer-friendly descriptions and presented in Appendix B.

We also provide the Neoverse V1 PMU Guide[2], a supplementary guide to the hardware PMU events implemented by the core. This PMU Guide provides detailed descriptions of PMU events categorized per CPU block. Micro-architectural and architectural definitions required for a better understanding of each PMU event are included, with relevant definitions marked as references to each PMU event description. The Neoverse V1 PMU Guide[2] also adds an exclusive CPU execution flow chapter that shows key CPU execution flows that the memory subsystem, with a

depiction of PMU events being counted in each stage. We recommend using the V1 PMU Guide[2] as the go-to reference manual for detailed descriptions of performance analysis activities using PMU events.

2.2 Neoverse V1 PMU Events Selection for Workload Characterization

For conducting performance analysis and workload characterization, Figure 1 presents a cheat sheet with Arm recommended subset of PMU events supported by Neoverse V1. Descriptions of these events can be referred to in Appendix B. These events can be used to derive metrics that can support the analysis methodology discussed in Section 3.

FIG. 1
Neoverse V1 PMU Events Cheat Sheet

Events Cheat Sheet

Topdown Level 1	Cycle Accounting	Misses Per Kilo Instructions
• BR_MIS_PRED (r10) • CPU_CYCLES (r11) • OP_RETIRED (r3a) • OP_SPEC (r3b) • STALL_SLOT (r3f) • STALL_SLOT_BACKEND (r3d) • STALL_SLOT_FRONTEND (r3e)	• CPU _CYCLES (r11) • INST_RETIRED (r08) • STALL_BACKEND (r24) • STALL_FRONTEND (r23)	• BR_MIS_PRED_RETIRED (r22) • DTLB_WALK (r34) • INST_RETIRED (r08) • ITLB_WALK (r35) • L1D_CACHE_REFILL (r03) • L1D_TLB_REFILL (r05) • L1I_CACHE_REFILL (r01) • L1I_TLB_REFILL (r02) • L2D_CACHE_REFILL (r17) • L2D_TLB_REFILL (r2d) • LL_CACHE_MISS_RD (r37)
Branch Effectiveness	**Instruction TLB Effectiveness**	**Miss Ratio**
• BR_MIS_PRED_RETIRED (r22) • BR_RETIRED (r21) • INST_RETIRED (r08)	• INST_RETIRED (r08) • ITLB_WALK (r35) • L1I_TLB (r26) • L1I_TLB_REFILL (r02) • L2D_TLB (r2f) • L2D_TLB_REFILL (r2d)	• BR_MIS_PRED_RETIRED (r22) • BR_RETIRED (r21) • DTLB_WALK (r34) • ITLB_WALK (r35) • L1D_CACHE (r04) • L1D_CACHE_REFILL (r03) • L1D_TLB (r25) • L1D_TLB_REFILL (r05) • L1I_CACHE (r14) • L1I_CACHE_REFILL (r01) • L1I_TLB (r26) • L1I_TLB_REFILL (r02) • L2D_CACHE (r16) • L2D_CACHE_REFILL (r17) • L2D_TLB (r2f) • L2D_TLB_REFILL (r2d) • LL_CACHE_MISS_RD (r37) • LL_CACHE_RD (r36)
Data TLB Effectiveness	**L1 Instruction Cache Effectiveness**	**L1 Data Cache Effectiveness**
• DTLB_WALK (r34) • INST_RETIRED (r08) • L1D_TLB (r25) • L1D_TLB_REFILL (r05) • L2D_TLB (r2f) • L2D_TLB_REFILL (r2d)	• INST_RETIRED (r08) • L1I_CACHE (r14) • L1I_CACHE_REFILL (r01)	• INST_RETIRED (r08) • L1D_CACHE (r04) • L1D_CACHE_REFILL (r03)
L2 Unified Cache Effectiveness	**Last Level Cache Effectiveness**	**Speculative Operation Mix**
• INST_RETIRED (r08) • L2D_CACHE (r16) • L2D_CACHE_REFILL (f17)	• INST_RETIRED (r08) • LL_CACHE_MISS_RD (r37) • LL_CACHE_RD (r36)	• ASE_SPEC (r74) • BR_IMMED_SPEC (r78) • BR_INDIRECT_SPEC (r7a) • DP_SPEC (r73) • INST_SPEC (r1b) • LD_SPEC (r70) • ST_SPEC (r71)

2.3 Neoverse V1 PMU Derived Metrics for Workload Characterization

For conducting performance analysis and workload characterization, Figure 2 presents a cheat sheet with Arm specified derived metrics using the PMU events shortlisted in Figure 1. Appendix C has the details of all the metric groups and metrics supported by Neoverse V1.

FIG. 2
Neoverse V1 PMU Metrics Cheat Sheet

Metric Cheat Sheet			
Topdown Level 1	**Cycle Accounting**	**Misses Per Kilo Instructions**	
• backend_bound • bad_speculation • frontend_bound • retiring	• backend_stalled_cycles • frontend_stalled_cycles	• brand_mpki • dtlb_mpki • itlb_mpki • l1d_cache_mpki • l1d_tlb_mpki	• l1i_cache_mpki • l1i_tlb_mpki • l2_cache_mpki • l2_tlb_mpki • ll_cache_read_mpki
Branch Effectiveness	**Instruction TLB Effectiveness**	**Miss Ratio**	
• branch_misprediction_ratio • branch_mpki	• itlb_mpki • itlb_walk_ratio • l1i_tlb_miss_ratio • l1i_tlb_mpki • l2_tlb_miss_ratio • l2_tlb_mpki	• branch_misprediction_ratio • dtlb_walk_ratio • itlb_walk_ratio • l1d_cache_miss_ratio • l1d_tlb_miss_ratio	• l1i_cache_miss_ratio • l1i_tlb_miss_ratio • l2_cache_miss_ratio • l2_tlb_miss_ratio • l1_cache_read_miss_ratio
Data TLB Effectiveness	**L1 Instruction Cache Effectiveness**	**L1 Data Cache Effectiveness**	
• dtlb_mpki • dtlb_walk_ratio • l1d_tlb_miss_ratio • l1d_tlb_mpki • l2_tlb_miss_ratio • l2_tlb_mpki	• l1i_cache_miss_ratio • l1i_cache_mpki	• l1d_cache_miss_ratio • l1d_cache_mpki	
L2 Unified Cache Effectiveness	**Last Level Cache Effectiveness**	**Speculation Operation Mix**	
• l2_cache_miss_ratio • l2_cache_mpki	• ll_cache_read_hit_ratio • ll_cache_read_miss_ratio • ll_cache_read_mpki	• branch_percentage • crypto_percentage • integer_dp_percentage • load_percentage	• scalar_fp_percentage • simd_percentage • store_percentage

We refer to Chapter 3 for the Arm performance analysis methodology to be used for conducting workload characterization using the list of events and metrics specified in Figure 1 and 2.

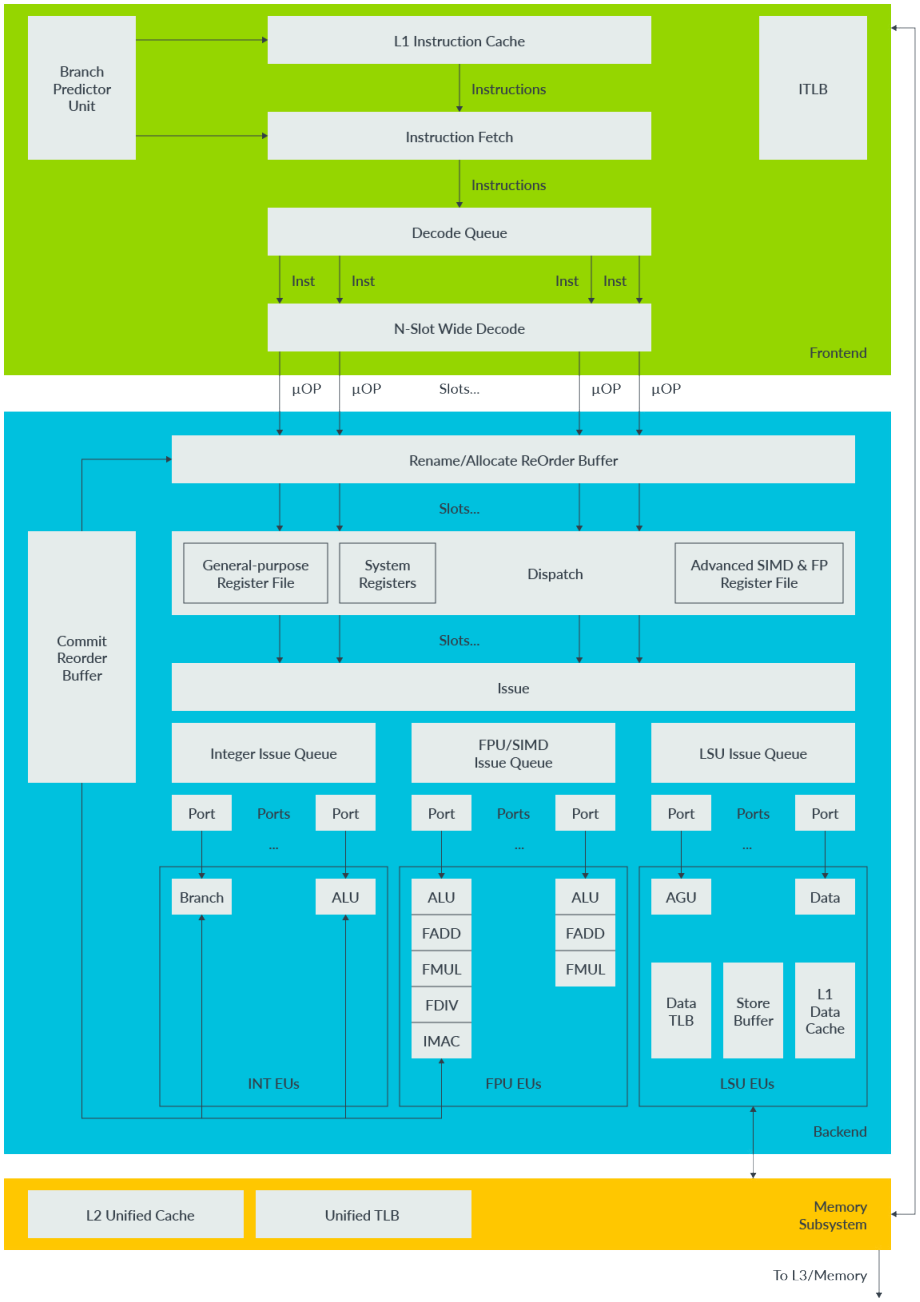
3. Neoverse V1 Performance Analysis Methodology

In this chapter, we give a brief introduction to the Neoverse V1 core micro-architecture. Followingly, we present the Arm top-down performance analysis methodology for Neoverse V1 with a set of validation tests to show how to use this methodology for optimization use cases. The Neoverse V1 is the first CPU in the Neoverse family that supports a full set of level 1 metrics for top-down analysis, which is introduced in Section 3.2.

3.1 Neoverse V1 Processor

Figure 1 shows the micro-architecture details of the Arm Neoverse core, which is a super pipelined super-scalar processor which has an in-order frontend and out-of-order backend.

FIG. 3
Arm Neoverse V1 Core
Micro-architecture Block Diagram



The frontend of the core comprises of the instruction fetch and decode units. The frontend also includes a branch predictor unit that fetches instructions ahead of the pipeline and helps to hide latencies caused by control flow bubbles in the pipeline. The fetch unit can fetch multiple instructions per cycle whose bandwidth is specific to a micro-architecture

design, which gets stored in a decode queue. The decode queue sends multiple instructions per cycle for decoding, whose bandwidth is determined by the number of decode slots available. The decode unit decomposes the Arm architecture instructions into micro-operations. The decode unit decode more than one micro-operations per cycle, which is fed to the re-order buffer for execution in the out-of-order backend. This bandwidth is determined by the number of renamed SLOTS available in the micro-architecture. From a micro-architecture standpoint, the rename unit is considered the boundary between the frontend and backend of the processor.

The backend of the core has a scheduler in the dispatch unit that orchestrates the operation executed when the issue queue associated with the operation can store the execution. The issue queue sends operations for execution when the execution unit is free and the operands are ready. Once the execution is complete, the results are sent to the commit reorder buffer(ROB) from where the instructions are retired when the speculated execution is confirmed. The backend of the CPU executes the operations out-of-order and stores results, with the help of the reorder buffer. The reorder buffer helps to track dependencies between operations (or is it the dispatch unit) and tracks the operand availability for the execution of operations. Register renaming is undertaken in this stage as well to mitigate data dependency hazards.

Issue queues are employed for:

01 Queuing the micro-operations(uops) to assigned ports.

02 Managing dependencies between operations.

03 Tracking operand availability for execution.

Each execution port supports different categories of operations. After the execution of operations, ROB is updated with execution results and operations that are completed are retired architecturally in the right program order.

Memory Subsystem

The Memory subsystem of the CPU handles the execution of load and store operations which rely heavily on the memory hierarchy levels. The Neoverse V1 has a dedicated L1/L2 cache per core, where the L2 cache is shared between the L1 data cache and the L1 instruction cache.

The Load Store Unit controls the data flow between the caches and to memory. The Neoverse V1 has multiple load/store units, which can both handle read and write operations. The core supports two hierarchical set associative caches, L1 Data Cache and L2 Cache whose size is configurable per implementation. The private L2 cache of the core connects to the rest of the system via an AMBA 5 CHI interface.

Neoverse V1 System Configurations

All systems with the coherent mesh interconnect support a shared system-level cache. Understanding the cache hierarchy and configuration of the system being analyzed is crucial in deriving insights from the cache effectiveness Performance Monitoring Unit (PMU) events. It is always best to check with the Silicon Provider for details on the system configuration for the underlying system, including the cache sizes.

3.2 Neoverse V1: Topdown Performance Analysis Methodology

As explained in section 3.1, out-of-order machines are heavily pipelined to achieve higher instructions per cycle (IPC). These deep pipelines can be stalled in different parts of the pipeline simultaneously caused by different operations in flight. Pipeline stalls cause significant IPC drops, which result in inefficient execution of the program. We introduce the Arm topdown

performance analysis methodology as our solution to support performance analysis, workload characterization, and micro-architecture exploration on Arm architecture compliant CPUs that support the Performance Monitors Extension. This methodology uses PMU events in the hardware to help profile an application to identify processor bottlenecks, and aid root cause analysis.

To tune an application code for a micro-architecture, the first step is to detect the code bottleneck which is where most of the cycles are spent. For this, we need to measure the distribution of execution cycles spent, which provides insights into the cycles that were both efficient and wasted by pipeline stalls and redirections. Followingly, we need to measure micro-architectural metrics that help deep dive into the bottlenecking CPU component for further analysis. Arm top-down methodology for hotspot analysis and micro-architectural analysis is specified to be conducted in two stages, as depicted in Figure 2.

Stage 1: Topdown Analysis

Topdown analysis is the first stage to be followed in the methodology which helps with hotspot detection. A set of pipeline efficiency metrics are specified using the PMU events to measure, which helps to characterize the distribution of cycles spent by the processor. Topdown analysis metrics are formulated as a decision tree of metrics that need to be traversed within each metric group to help locate the bottleneck.

Neoverse V1 only supports the first level of this decision tree. Further levels in this stage will be supported by the future Neoverse cores.

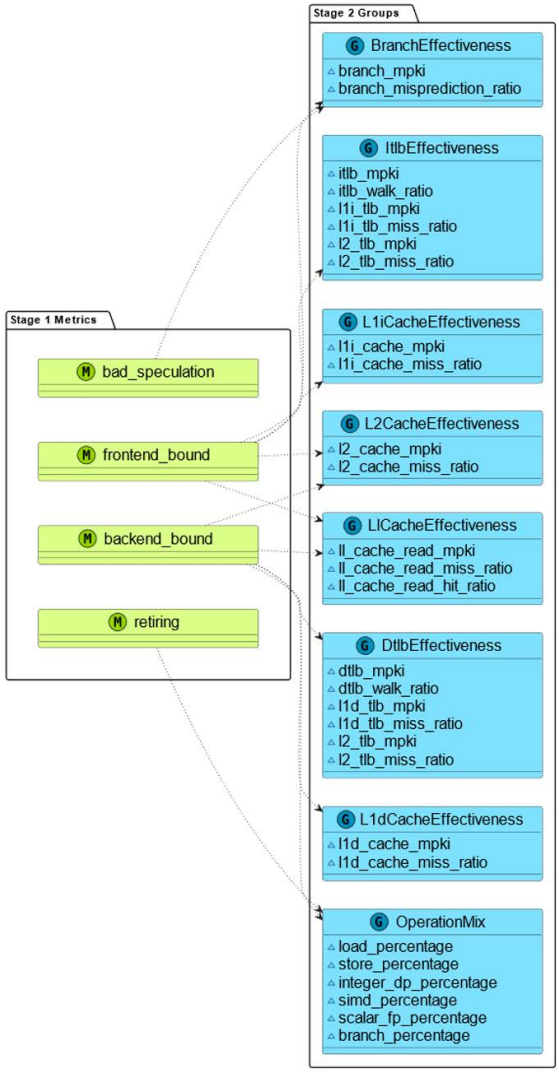
Stage 2: Micro-architecture Exploration

Once the potential hotspot in the processor pipeline is identified in stage 1, the next step is to conduct a micro-architectural analysis of the bottlenecking CPU resource. Stage 2 is defined as the micro-architecture

exploration stage for which we specify a set of CPU resource effectiveness metrics in metric groups per resource. Industry-standard metrics like Misses Per Kilo Instructions (MPKI) and Miss Ratios are also metric groups defined in this stage.

Arm recommends collecting all the metrics that are in Stage 1 and Stage 2 for workload characterization. For further analysis, we have also specified our recommended set of micro-architecture exploration metric groups against some of the hotspots detected in Stage 1. Note that all the Stage 2 metrics can be used to derive further insights into the overall micro-architecture behaviour during the execution of the application under investigation and can be used independently to Stage 1.

FIG. 4
Arm V1 Topdown Performance Analysis Methodology for Neoverse V1



3.2.1 Stage 1 : Topdown Analysis

Topdown analysis starts by making the following four measurements, each being a percentage of the total execution bandwidth of the processor:

- The percentage of execution bandwidth used by operations that are retired.
- The percentage of execution bandwidth lost to mis-speculation.
- The percentage of execution bandwidth lost to stalls in the frontend.
- The percentage of execution bandwidth lost to stalls in the backend.

The total execution bandwidth of the processor can be measured in execution slots for operations. Slots are defined as the execution slots in the rename unit which partitions the frontend and backend of the processor. Frontend of the processor decodes and decomposes AArch64 instructions to micro-operations that can be executed by the backend execution units as explained in Section 3.1. The number of slots supported by the core determines the execution bandwidth of the processor for top-down accounting. This is a micro-architectural parameter that is part of the formulae for deriving the execution bandwidth related metrics.

Neoverse V1 supports four key metrics for top-down analysis level 1 that are slot-based, which is a measurement of the efficiency of pipeline slots. The four metrics in the first level are defined part of the metric group TopDownL1 as below:

- *frontend_bound:* This metric is the percentage of total slots that were stalled due to resource constraints in the frontend unit of the processor.
- *backend_bound:* This metric is the percentage of total slots that were stalled due to resource constraints in the backend unit of the processor.

- *bad_speculation:* This metric is the percentage of total slots that executed operations that didn't retire due to a pipeline flush. This indicates the cycles that were used but were inefficient as well as cycles spent recovering from the mis-speculation, refilling the pipeline from the correct location.
- *retiring:* This metric is the percentage of total slots that retired operations. This indicates the cycles that were used and efficient.

We refer to Appendix C for details on this metric groups and its corresponding metrics. We demonstrate the usage of these metrics using a validation workload suite introduced below.

3.2.2 UStress: Micro-architecture Metrics Validation Workload Suite

In order to validate the specified performance analysis metrics & events, we developed an in-house validation suite comprising a set of micro-architecture workloads that stress some of the major CPU resources like branch prediction units, execution units (arithmetic and memory), caches, and TLBs. These workloads can cause various performance bottleneck scenarios in the CPU. The categories of tests and their respective list of micro-benchmarks are as below.

- Branch: *branch_direct_workload*, *branch_indirect_workload*, *call_return_workload*.
- Data Cache: *l1d_cache_workload*, *l2d_cache_workload*.
- Instruction Cache: *l1i_cache_workload*.
- Data TLB: *l1d_tlb_workload*.
- Arithmetic Execution Units: *div32_workload*, …, *fpdiv_workload*, …, *mul64_workload*.
- Memory Subsystem: *memcpy_workload*, *store_buffer_full_workload*, *load_after_store_workload*.

We will walk through the recommended performance analysis methodology using both stage 1 and stage 2 metrics with data collected by running these tests on a Neoverse V1 platform built with gcc-10.3. We refer to Chapter 4 in the previous whitepaper on Arm Neoverse Core N1: Performance Analysis Methodology for PMU data collection and sampling techniques using Linux perf tool[5].

As a first step, let us look at Stage 1 top-down analysis metrics measurements for each test in Figure 3 below.

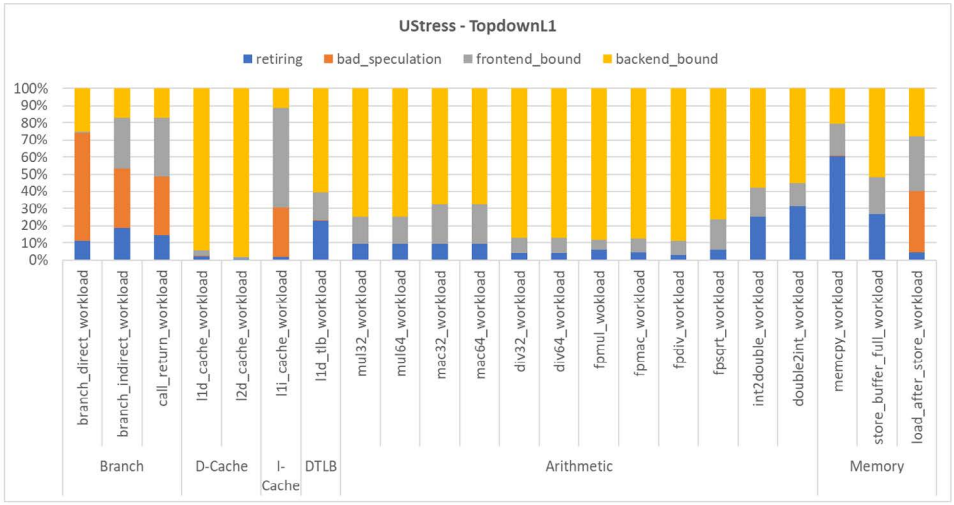


FIG. 5
UStress workloads: Top-down Analysis Level 1 Characterization

The following observations can be made on Figure 5 for each workload category.

01 Branch tests that cause mispredictions of different branch types result in pipeline flushes when branch targets or directions are resolved. These flushes translate to the bad speculation-related stalls. As expected, the relative percentage of pipeline bandwidth shows the trend of a high percentage on *bad_speculation* metric (34% ~ 63%) for this workload category.

02 Data Cache tests cause data accesses that result in heavy L1 data cache and L2 unified cache misses, which stall the backend of the processor. As expected, tests in this category show a high percentage of *backend_bound* metric (>90%).

03 Instruction Cache test causes heavy L1I cache miss during instruction fetch, which stalls the frontend of the processor in the fetch stage. As expected, test in this category measures a relatively high percentage of *frontend_bound* metric (58%).

04 Data TLB tests that cause heavy data TLB miss that would cause stalls in the processor's backend caused by delays in the memory address translation stage. As expected, the tests show high percentage of *backend_bound* metric (61%).

05 Arithmetic execution unit tests stress the different execution units of the processor that process various arithmetic operations of different latency requirements. The pressure in the execution units will be reflected as a stall in the processor's backend. As expected, tests in this category measure a high percentage of *backend_bound* metric (55% ~ 89%).

06 Memory subsystem stresses the load store units associated with the memory operation executing in the processor's backend. As expected, these tests result in a high percentage of *backend_bound* metric (63% for *memcpy_workload*, 52% for *store_buffer_full_workload*). The test *load_after_store_workload* results in high *bad_speculation* (36%) because the code triggers many speculative loads which are abandoned due to mispredicted data address. The test *memcpy_workload* copies memory block smaller than L1D Cache efficiently in batch, which results in high retiring.

Based on Stage 1 top-down analysis measurements for the validation tests, we have shown that the proposed top-down level 1 metrics provide clear indication of the bottlenecking part of the processor pipeline, which is a first step to locating the bottleneck or hotspot of the program.
The next phase of the analysis process is to investigate the potential bottle-necking micro-architecture components. Such micro-architecture exploration metrics are grouped per CPU resource in the next stage termed “Stage 2”, which is discussed below.

3.2.3 Stage 2: Micro-architecture Exploration

Once the execution pipeline bottleneck region is identified from Stage 1, the next step is to deep dive for further analysis.

A relatively high *frontend_bound* metric shows that execution cycles are being wasted due to pipeline stalls in the in-order frontend division of the processor. This can be because of many reasons like inefficiency in the branch prediction unit, fetch latency due to instruction cache misses and translation delays caused by Instruction TLB walks.

A relatively high *backend_bound* metric shows that execution cycles are wasted due to pipeline stalls in the processor’s backend. This can be because of many reasons like inefficiency in backend units like execution units, data cache misses and translation delays caused by data TLB walks.

A relatively high *bad_speculation* metric shows the pipeline stalls caused by flushes or machine clears that break the pipeline needing a control flow change. Branch mis-predictions are one of the major causes for this, as well as exceptions.

A relatively high *retiring* metric means the pipelines were utilized. However, this metric could indicate inefficiency in terms of underutilization of the micro-architectural capabilities, for example scalar execution of a code that could have performed more efficiently with vector operations.

To analyze this further, we propose the below micro-architecture exploration metric groups that can be used for Stage 2 analysis. In this stage, we recommend a set of metric groups for narrowing down the further analysis of tests falling into the four categories of pipelined BW usage in Stage 1, as shown in Figure 2.

As a common step in Stage 2, we first introduce two metric groups MPKI and Miss rate, which can help with a quick behavioural analysis of the CPU components that could be the potential bottlenecks.

MPKI – Misses Per Kilo Instructions

Misses Per Kilo Instructions is a set of metrics that can be derived to normalize the misses in CPU components, mainly branches, caches and TLBs, against the total instructions executed. This is an industry-standard metric that also helps with comparison across different implementations of the Arm architecture, as instructions retired should count the same on all AArch64-based micro-architectures.

Section 2.3 lists all the MPKI metrics that can be derived for Neoverse V1 and Appendix B has all the metrics and their formulae.

FIG. 6
UStress Workloads: MPKI

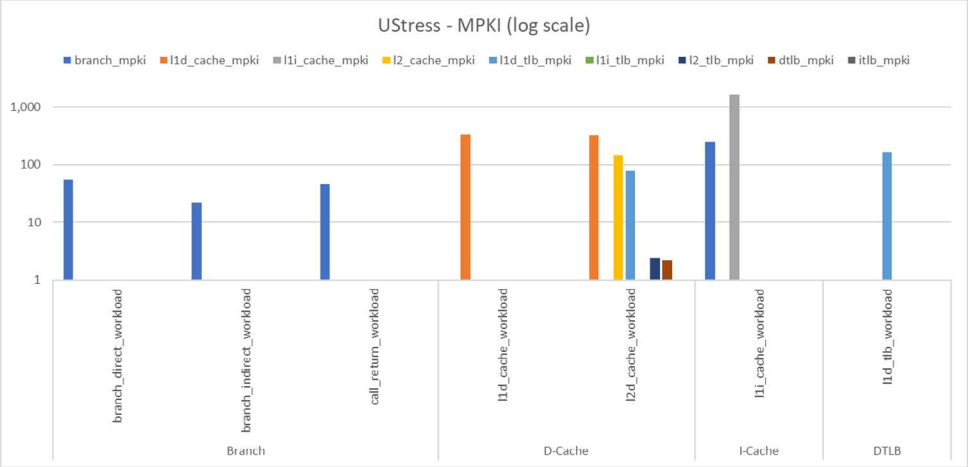


Figure 4 shows the results of the key MPKI metric measurements for Branch/Cache/TLB related UStress validation tests. Arithmetic and Memory tests are not plotted as they have very low MPKI. We refer to Table 1 in Appendix A for the full set of MPKI measurements for the UStress workloads.

The following observations can be made on Figure 6 for each workload category.

01 The branch tests show relatively high branch MPKI values compared to other metrics, as expected.

02 The data cache tests relatively high L1D MPKI and L2 MPKI for tests *l1d_cache_workload* and *l2d_cache_workload* respectively, along with some pressure in the frontend.

03 The instruction cache test shows relatively high L1I MPKI and branch MPKI, matching the expected behavior for a frontend_bound workload. In this scenario, we may need to explore branch and L1I cache effectiveness metrics further to determine the root cause. Sometimes pressure in one CPU resource can cause pressure in other components. In this test, L1I MPKI is above 1000 which is unusual as this means L1I_CACHE_REFILL is greater than INST_RETIRED. This is

because L1I_CACHE_REFILL can be triggered by speculatively executed code that did not retire. In this case it is advised to check the INST_SPEC against the INST_RETIRED count to see how big is the difference between speculatively executed instructions to the retired instructions.

04 Data TLB test shows relatively high L1D TLB MPKI. However, we see little DTLB Walk MPKI which shows no cost in terms of translation table walks.

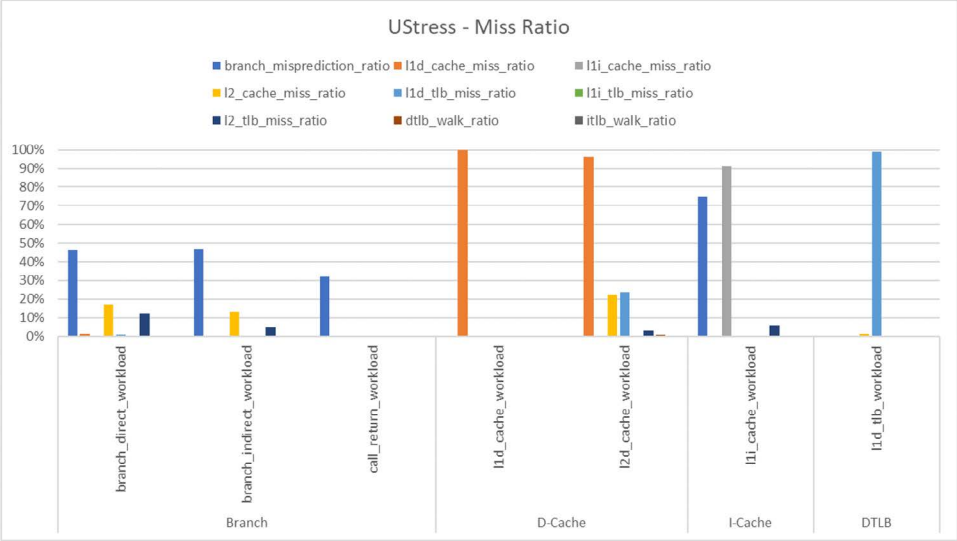
05 Arithmetic execution unit tests show very low MPKI. MPKI does not matter much in these tests as these are mainly core bound in the backend. These workloads will need more backend related metrics to further analyze the bottlenecks.

06 Memory subsystem unit tests show low MPKI as the test data always hit in L1D cache.

Miss Ratio

Miss ratio metric group provide a set of metrics that calculate ratio of the misses in the CPU components, mainly branches, caches and TLBs, against the total accesses in those components. These metrics provide insights on the efficiency of each CPU component in the pipeline and help to root cause issues.

FIG. 7
UStress Workloads: Miss Ratio



Section 2.3 lists all the Miss Ratio metrics that can be derived for Neoverse V1 and Appendix B has all the metrics and their formulae.

Figure 5 shows the results of the key Miss Ratio metric measurements for Branch/Cache/TLB related UStress validation tests. Arithmetic and Memory tests are not plotted as these workloads do not record misses. We refer to Table 2 in Appendix A for the full set of Miss Ratio measurements for the UStress workloads.

The following observations can be made on Figure 7 for each workload category.

01 Branch tests show relatively high branch mis-prediction ratio (30% ~ 50%) values against other metrics, which confirms the expected behaviour.

02 Data Cache tests show high L1D cache miss ratio (>95%) for *l1d_cache_workload* and high L2 miss rate for *l2d_cache_workload*.

03 Instruction Cache test *l1i_cache_workload* shows relatively high L1I cache miss ratio (>80%) and high branch misprediction rate (74.82%) as expected.

04 Data TLB test *l1d_tlb_workload* shows relatively high L1D TLB miss ratio (>95%), as expected.

05 Arithmetic execution unit tests mostly show high L2 cache miss ratio (~18%), but the corresponding MPKI is very low. The high miss ratio is due to very few L2 accesses, but not a true bottleneck.

06 Similar to Arithmetic unit tests, L2 miss ratio of the memory subsystem tests are due to few L2 accesses, not high misses.

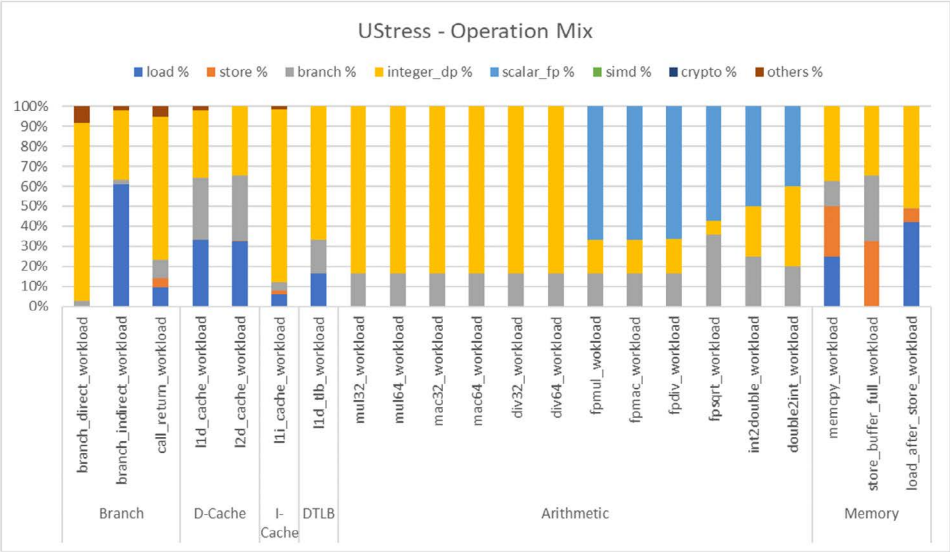
Operation Mix

The Neoverse V1 micro-architecture as shown in Figure 1 has a variety of execution units that can process five types of operations: branch, single-cycle integers, multicycle integers, load/store unit with address generation, and advanced floating-point/SIMD operations. Operations that are issued to these execution units can be counted by the PMU events listed in Section 2.2 under OperationMix.

Section 2.3 lists all the OperationMix metrics that can be derived for Neoverse V1 and Appendix C has all the metrics and their formulae.

Note that these metrics use events that count speculatively issued operations at the issue stage, which provide an estimate of the execution unit utilization, but not the retired instruction mix of the program. To derive the utilization of each operation type, the percentage of each type of operation is calculated as a fraction of the total operations issued, which is counted by the event INST_SPEC.

FIG. 8
UStress Workloads:
Operation Mix Metrics



Neoverse V1 does not support retired events for counting the architectural instruction mix. Neoverse V1 supports events to further break down the branch operations into immediate, indirect, and return branches, counted by events BR_IMMED_SPEC, BR_INDIRECT_SPEC, and BR_RETURN_SPEC respectively. Note that BR_RETURN_SPEC is a subset of BR_INDIRECT_SPEC. Sum of the BR_IMMED_SPEC and BR_INDIRECT_SPEC branch operation events can compute the total branches executed.

Figure 8 shows the Operation Mix metrics measurements for all the UStress workloads.

The following observations can be made on Figure 8 for each workload category.

01 Branch instruction proportion of branch mis-prediction tests looks counter-intuitive with high percentage of load and integer operations and a minimal percentage of branch operations (2.41% ~ 11.67%). For example, in the test *branch_direct_workload*, operation mix measurements show that only 2.94% of instructions are branches. Inspecting assembly code shows there should at least be 10% ~ 12.5% (1/10 ~ 1/8) branch instructions. On Neoverse V1, Operation Mix is calculated on the *speculated* operations issued to the processor, not the *retired* ones. In these tests, a high branch mis-prediction rate causes numerous speculated operations to be abandoned, causing a significant gap in the speculated and retired instruction counts. In the case of branches, Neoverse V1 supports both INST_RETIRED and BR_RETIRED events to compare the ratio of retired branch instructions against total retired instructions.

02 Data Cache tests show a large proportion of load operations (32.41%) as well as integer operations (34.77%) as expected.

03 Instruction Cache test shows very high scalar integer operations (>80%). This test calls a chain of functions located in gaps of instruction cache size by continuously incrementing and dereferencing a function pointer, which are integer operations. Moreover, due to high branch misprediction ratio, the speculated integer operations are much higher than the retired ones.

04 Data TLB test shows relatively high scalar integer operations and a mix of loads and branch operations (integer_dp=66.62%, load=16.61%, branch=16.73%).

05 Arithmetic execution unit tests (mul and div tests) show high scalar operation percentage (83.3%) for scalar integer tests and high floating point operations (66.5%) for the FP tests (fpmul and fpdiv). For tests that are double to integer conversion, we see a mix of scalar fp (40.0%) as well as scalar integer (40.0%) as expected.

06 Memory subsystem test *memcpy_workload* has a greater proportion of load and store operations, as expected. For the *store_buffer_full_workload* test, high store (32.5%) is observed while the *load_after_store_workload* test shows a high proportion of load operations (42.2%).

Branch Effectiveness & Branch Mix

Branch mis-predictions are costly in a deeply pipelined CPU, causing pipeline flushes and wasted cycles. As a general rule, workloads typically contain, on average, 1 branch in every 6 instructions. Though modern CPUs have optimized branch prediction units, there are many use cases like ray tracing, decision tree algorithms, etc. that are branch heavy and hard to predict. In some of these applications, there can be hundreds of unique branch paths to take and the target may be input data dependent. Branch prediction performance can be evaluated using two PMU events, BR_MIS_PRED_RETIRED and BR_RETIRED. BR_MIS_PRED_RETIRED provides an account of the total branches that were executed but mis-predicted. This means that the direction of the code path was wrong and the following operations in the path were wasted, causing a pipeline flush. BR_RETIRED counts the total branches architecturally executed by the CPU.

Two performance metrics that can be derived for a high-level evaluation of the branch execution performance regarding the overall program execution are the *branch_mpki* and *branch_misprediction_ratio* metrics. *branch_mpki* provides total branch mispredictions per kilo instructions. *branch_misprediction_ratio* gives an indication of the ratio of branches that were mis-predicted to overall branches.

Section 2.3 lists all the BranchEffectiveness metrics that can be derived for Neoverse V1 and Appendix C has all the metrics and their formulae.

In the UStress branch tests, the tests *branch_direct_workload*, *branch_indirect_workload* and *call_return_workload* measured high *bad_speculation* bound metric in Stage 1 top-down analysis stage. For these tests, we discussed the MPKI and Miss ratio values from Figures 6 and 7 respectively, which show branch-related metrics relatively high compared to the other resources. In a branch performance-bound workload, the PMU events specified here can be sampled using perf record, to determine which functions are causing the increased branch miss rates. We refer to Chapter 4 in N1 performance methodology whitepaper for guidance on how to perform sampling using Linux perf tool [5, Chapter 4].

Branch prediction units work differently depending on the branch type. There are three main sub-units that work for different branch types as below.

- Branch History Table (BHT) that stores the history of conditional branches, taken or not.
- Branch Target Buffer (BTB) that stores the target address for indirect branches.
- Return Address Stack (RAS) that stores the function return branches.

Neoverse V1 supports three events, BR_IMMED_SPEC, BR_RETURN_SPEC and BR_INDIRECT_SPEC, to categorize the immediate, indirect and return

branches executed, respectively. Getting a breakdown of the branch type helps to deep dive into each of these sub-units within the branch prediction unit. Branch tests *branch_direct_workload*, *branch_indirect_workload* and *call_return_workload* stress each of these branch sub-units. Let us look at the breakdown of the branch category events for each UStress branch tests in Table 1 as a set of Branch Mix metrics defined below:

```
%immediate = BR_IMMED_SPEC/   (BR_IMMED_SPEC  + BR_INDIRECT_SPEC)
%indirect  = ( BR_INDIRECT_SPEC – BR_RETURN_SPEC )/ ( BR_IMMED_SPEC +  BR_INDIRECT_SPEC )
%return    = BR_RETURN_SPEC / (BR_IMMED_SPEC + BR_INDIRECT_SPEC)
```

TABLE 1
Branch Operation Mix For UStress Branch Workloads

Tests	% immediate	% indirect	% return
branch_direct_workload	99.98%	0.01%	0.01%
branch_indirect_workload	11.99%	87.98%	0.02%
call_return_workload	42.26%	28.86%	28.88%

As expected, *branch_direct_workload* has high percentage of immediate branches (99.98%) and *branch_indirect_workload* contains high percentage of indirect branches (87.98%). The *call_return_workload* has a mix of direct, indirect and return branches with relatively high return branches (28.88%) compared to other tests.

TLB/MMU Effectiveness

Another important performance evaluation step is to check the virtual memory system performance, that affects the instruction fetch performance in frontend and memory access performance on the data side. The processor needs to translate a virtual address to physical address for any instruction/data memory access before it accesses the respective cache. Note that a program's view of memory is virtual address, but the processor works with the physical address when accessing cache or memory.

Virtual to physical mappings are defined in the page translation tables which reside in system memory. Accessing these tables requires one

or more memory accesses which take many cycles to complete—this is referred to as a translation table walk. However, to make these translations faster, Translation Lookaside Buffers (TLBs) cache translation table walks, greatly reducing the number of accesses to system memory.

Neoverse V1 implements a two level TLB hierarchy. The first level contains separate, dedicated TLBs for the instruction and data (load/store) address translations. Total accesses to these TLBs are counted by L1I_TLB and L1D_TLB respectively. The second level contains a unified L2 TLB that is shared by both instruction side and data side accesses. There are corresponding REFILL counters, that count the refills in these TLB levels. Some performance metrics that can be derived for a high-level evaluation of the TLB execution performance are the l<n>*_tlb_mpki* and l<n>*_tlb_miss_rate* metrics, where <n> stands for each levels of TLB instruction and data side.

Those accesses that cause a translation table walk due to misses in the instruction side and data side TLBs are counted by events, ITLB_WALK and DTLB_WALK respectively. For evaluating the TLB effectiveness and cost of latency caused by translation table walks specifically, *dtlb_mpki*, *dtlb_walk_ratio*, *itlb_mpki* and *itlb_walk_ratio* are the key metrics that can be derived. *itlb_mpki* and *dtlb_mpki* provide the rate of TLB Walks per kilo instructions for instruction and data accesses respectively. These derived metrics help to evaluate and correlate the TLB efficiency with respect to the total instructions. *dtlb_walk_ratio* provides ratio of DTLB Walks to the overall TLB lookups made by the program. Note that this is the same as DTLB_WALK/MEM_ACCESS as every MEM_ACCESS causes a L1D_TLB access. *itlb_walk_ratio* provides a percentage of ITLB walks to the overall TLB lookups initiated from the instruction side.

Section 2.3 lists all the TLBEffectiveness metrics that can be derived for Neoverse V1 and Appendix C has all the metrics and their formulae.

In the Data TLB tests from our validation suite, the test *l1_dtlb_workload* measured high *frontend_bound* metric in Stage 1 top-down analysis stage. For this test, we discussed the MPKI and Miss ratio values from Figure 6 and Figure 7, which show data-tlb related metrics relatively high compared to the other resources. In a data TLB performance-bound workload, the PMU events specified here can be sampled using perf record, to determine which functions are causing the increased branch miss rates. We refer to Chapter 4 in N1 performance methodology whitepaper[5] for guidance on how to perform sampling using Linux perf tool.

Cache Effectiveness

The Neoverse V1 implements a multi-level cache hierarchy. The first level (L1) includes a dedicated cache for instructions and a separate dedicated cache for data accesses. The second level (L2) is a unified L2 cache that is shared between code and data. Further down the hierarchy, the system could have an optional shared system level cache (SLC) in the interconnect. It is recommended to check with the platform providers for cache configurations.

The Neoverse V1 core supports hierarchical PMU events for all the cache hierarchy levels. For each level of caches, there are total access counts and refill counts. Note that AArch64 do not support cache MISS counters, but only REFILLs. A cache miss could lead to multiple cache line refills if the access is on a cache line boundary or multiple cache misses could be satisfied by a single REFILL. We refer to the V1 PMU Guide[2] for details on the cache event counter descriptions. Cache policies and associativity details can also be referred in the Chapter Micro-architecture details in the V1 PMU Guide[2].

Some performance metrics that can be derived for a high-level evaluation of the cache execution behavior are the l<n>*__mpki* and l<n>*_tlb_miss_ratio* metrics, where <n> stands for each levels of instruction and data caches.

Section 2.3 lists all the L<n>CacheEffectiveness metrics that can be derived for Neoverse V1 and Appendix C has all the metrics and their formulae.

In the cache tests from our validation suite, the test *l1i_cache_workload* measured high *frontend_bound metric* and the tests *l1d_cache_workload* and *l2d_cache_workload* measured high *backend_bound_metric* in Stage 1 top-down analysis stage. For these tests, we discussed the MPKI and Miss ratio values from Figure 6 and Figure 7, which show the respective cache-related metrics relatively high compared to the other resources. In cache performance-bound workload, the PMU events specified here can be sampled using perf record, to determine which functions are causing the increased branch miss rates. We refer to Chapter 4 in N1 performance methodology whitepaper[5] for guidance on how to perform sampling using Linux perf tool.

Core Memory Traffic

The MEM_ACCESS event counts the total number of memory operations that were issued by the Load Store Unit (LSU) of the core. As these operations are looked up in the L1D_CACHE first, both the events L1D_CACHE and MEM_ACCESS count at the same rate. Neoverse V1 also supports two additional events, MEM_ACCESS_RD and MEM_ACCESS_WR, that can provide the read and write traffic breakdown respectively. Note that these events are not the same as LD_SPEC and ST_SPEC since they count memory operations speculatively issued, but not necessarily executed.

Last Level Cache Counter Usage

On systems which support a shared system level cache in the interconnect, LL_CACHE_RD counts the total accesses to the SLC. In a system that has the SLC configured to count LL_CACHE_RD events, LL_CACHE_RD counter counts total SLC accesses made by the core and LL_CACHE_MISS_RD counts the access missed at SLC.

To study the last level read behavior, Last level cache read miss metrics that can be derived are *ll_cache_read_mpki* and *ll_cache_miss_ratio*. Another useful metric to measure the SLC hit percentage for the read traffic is the SLC Read Hit Ratio denoted as *ll_cache_read_hit_ratio*.

Last level cache events do not have a write variant in Neoverse V1 since SLC is only used as an eviction cache for the core and all the writes complete early at the interconnect when the transaction is acknowledged but not necessarily completed.

Section 2.3 lists all the LastLevelCacheEffectiveness metrics that can be derived for Neoverse V1 and Appendix C has all the metrics and their formulae.

Remote Cache Access

For Neoverse V1 systems with multiple sockets or SOCs, V1 supports the REMOTE_ACCESS event which counts the memory transactions that were completed by a subordinate source from another chip.

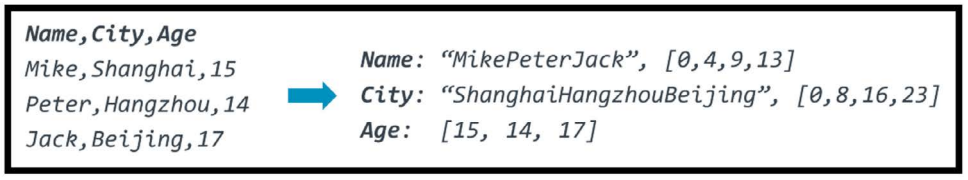
4. Case Study: Topdown Performance Analysis on Neoverse V1

This case study illustrates how the Arm topdown analysis methodology for Neoverse V1 was applied for code optimization of Apache Arrow CSV parser code, which achieved a performance uplift of 80%.

4.1. About Arrow CSV Parser

Apache Arrow[8] is an open-source project for efficient columnar data interchange. The library supports a variety of data structures that can be moved without ser-/deserialization. These data structures are highly efficient for in-memory computation. Apache Arrow supports multiple languages. In this case study, we work with the Arrow CSV Parser implemented in C++.

FIG. 9
9 CSV Data -> Arrow Data



Unlike the traditional dataset, which stores data row by row, the Arrow data is column-based where fields of the same column are contiguous in memory. The columnar format is especially convenient for Online Analytical Processing (OLAP) workloads[9].

Arrow CSV Parser converts CSV data (row based) to Arrow data (column based), as shown in Figure 9.

- On the left side is the CSV data. Each row represents one record. Fields of the same row are contiguous in the memory.
- On the right side is the Arrow data. Fields of the same column are contiguous in the memory. E.g., the three names (*Mike*, *Peter*, *Jack*) are packed in one large column buffer "*MikePeterJack*", with an index array [0,4,9,13] to slice the individual names from that buffer.

4.2. Hotspot Analysis with Topdown Methodology

We follow the below steps to analyze the performance of the CSV parser on a Neoverse V1 platform and evaluate if there are any optimization opportunities for this library.

Experiments are conducted on the Neoverse-V1 (Amazon Graviton3) built with Ubuntu 22.04 aarch64 OS and gcc-10.3 compiler. Apache Arrow code used is from the release build (-O3). Baseline performance is evaluated on commit f0110cf26 and optimized performance is evaluated on commit 464ccdef0.

Evaluate Baseline Performance

Firstly, we run the CSV parser benchmark to evaluate the baseline performance, which can be measured in IPC and the bandwidth obtained by the parser.

FIG. 10
Apache Arrow Baseline: Bandwidth and IPC

```
$ 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
```

From the benchmark result shown in Figure 10, Arrow CSV parser processes about 1G bytes CSV data per second. IPC (Instructions Per Cycle) achieved on the V1 platform > 4.5, which is quite high. Our first impression looking at this data would be that the Arrow library has probably adopted some optimization methods to achieve this high IPC. However, high IPC doesn't always mean efficient execution, so we will dig deeper into the code execution using stage 1 for top-down analysis first.

Conduct Stage 1 Topdown Analysis

We collected the PMU events to derive the *TopdownL1* metrics to evaluate the pipeline efficiency and plot it as in Figure 11.As the TopdownL1 chart shows in Figure 11, more than half of the cycles are retiring instructions, which matches the high IPC we observed. The code is ~25% *frontend_bound* and *backend_bound* with no speculation performance issues.

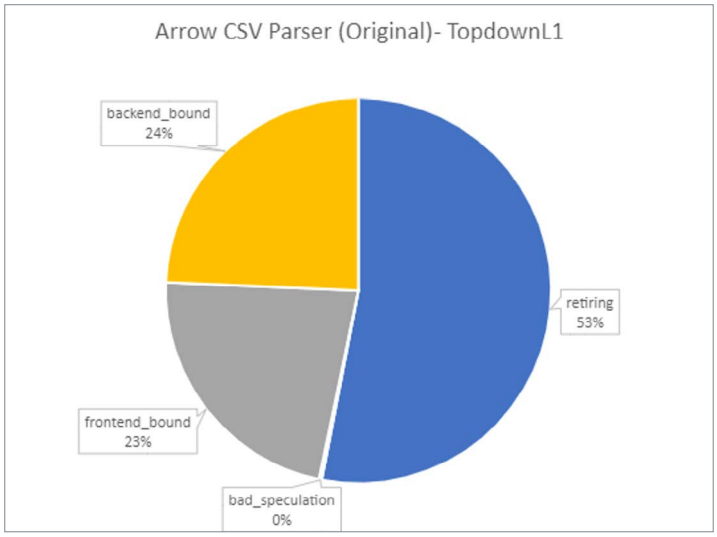


FIG. 11
Apache Arrow Baseline:
Stage 1 Top-down Analysis Metrics

Conduct Stage 2 Microarchitecture Exploration

Let us now look at the Stage 2 micro-architecture exploration metrics for CPU resource pressure evaluation. Firstly, we derive the *MPKI* and *MissRatio* metric groups which is presented in Table 2 below.

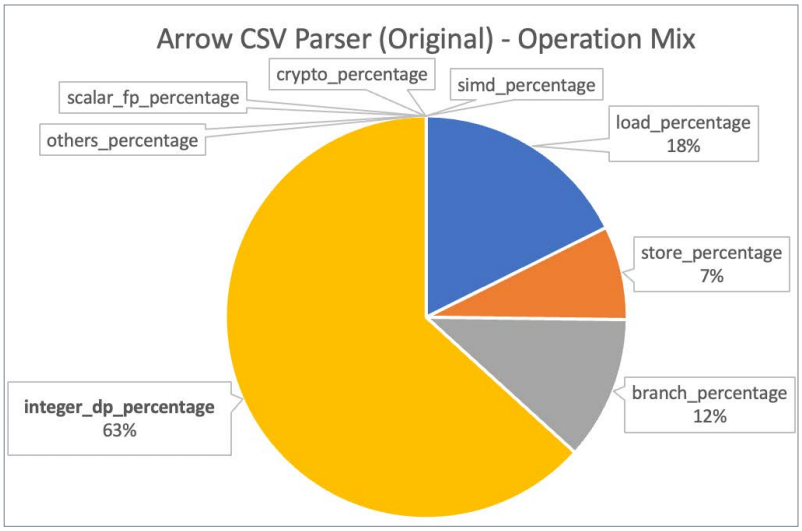
As shown in Table 2, both *MPKI* and *MissRatio* metrics are very low for most of the resources. The highest Miss Ratio comes from *l2_tlb_miss_ratio* = 43.85%, but the corresponding *l2_tlb_mpki* is only 0.05. There is no pressure on the Cache or Branch units. Now, let us look at the *Operation Mix* metrics of this workload.

TABLE 2
MPKI and Miss Ratio For UStress Tests

	branch	l1d_cache	l1i_cache	l2_cache	l1d_tlb	l1i_tlb	l2_tlb	dtlb	itlb
MPKI	0.01	2.12	0.04	0.26	0.12	0.00	0.05	0.05	0.00
Miss Ratio	0.01%	0.84%	0.03%	2.55%	0.05%	0.13%	43.85%	0.02%	0.06%

As observed in Figure 12, *integer_dp_percentage* is very high for this workload at 63% of operations. This counts the percentage of scalar integer processing instructions which is about 2/3 of operations executed. This iscunexpected for the workload that is parsing heavy volumes of data for which we would expect high volumes of load and store instructions. The Arrow CSV parser copies data from the CSV buffer to the Arrow buffer, and it must treat normal chars and field separators (comma, EOL, etc.) separately. High volumes of integer calculations are achieving a very high IPC value, but this could be executed more efficiently if there is an opportunity to leverage vector processing using SIMD. To investigate this, we first need to check where in the code are these integer operations executed.

FIG. 12
Apache Arrow Baseline: Operation Mix



Why High integer_dp_percentage?

Analyzing code location using the PMU sampling approach [5, Chapter 4], it turns out that the Arrow code is already optimized to process input CSV characters in batch, instead of sequential char-by-char processing, which could have explained the high scalar integer execution. Looking closely at the logic, it was identified that there is a code path in the conversion of CSV row-based data to Arrow column-based format that needs to handle some special tokens (comma, EOL, etc) differently than normal field chars. This path breaks the parallel execution requiring input character processing one by one. This is causing high scalar execution, which has the potential to be optimized. If we can pre-check for the special characters and make sure there is no special token in the upcoming data block (e.g., 8 continuous characters), the input can be saved in batches. This scenario is very common in real-world CSV data. The figure below illustrates the diverging code path with an example.

The code tuning exercise can be summarized as "Given a string of 8 chars, and a predefined character set, how to rewrite the code to pre-check if the string contains any char in the predefined set". Arrow implements a bloom filter to perform a quick scanning of the upcoming string block. This essentially maps a char to one bit in a uint64 mask to check if it matches a predefined token. The bloom filter executes a lot of integer shift, logical and arithmetic operations, which is the reason there is a high *integer_dp_percentage*.

FIG. 13
Diverging Code Path in Apache Arrow CSV Parser

Note: *Readers may argue that this pre-check and process in batch approach may hurt performance if there are many very short CSV fields. This is a real concern, and the Arrow CSV parser will check field sizes and adjust the best approach dynamically.*

4.3. Code Optimization with Arm Neon

Instead of a bloom filter, we can try to see if we can use Neon to vectorize the routine in the code that checks if there are any special tokens in 8 or 16 continuous chars.

Vectorization Optimization

The scalar code below illustrates how we check if a char matches any of the five special CSV tokens: return (\r), newline (\n), delimiter (,), quote (“), and escape (\). The delimiter, quote, and escape chars are configurable.

```
bool Matches(uint8_t c) {
  return (c == ‘\r’) | (c == ‘\n’) |(c == delim) | (c ==
quote) | (c == escape);
}
```

If we vectorize the code using Neon, 8 input chars can be checked 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;
}
```

Benchmark the Optimized Code

To evaluate the performance, we run the optimized CSV parser benchmark again.

FIG. 14
Apache Arrow Optimized: Bandwidth & IPC

```
$ 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
```

Compared with the original baseline code, optimized code performance increases by ~80% with a bandwidth uplift from 1.10GB/s to 1.99GB/s. Table 3 shows the change in cycles, instructions and IPC. The total instruction decreases by ~50% and cycles reduced significantly by ~45%. IPC is still very high at 4.30 though with a small drop from previous value. Note that IPC alone is not a measurement to evaluate performance drop or improvement as it is a ratio and in this case both instructions and cycles dropped heavily.

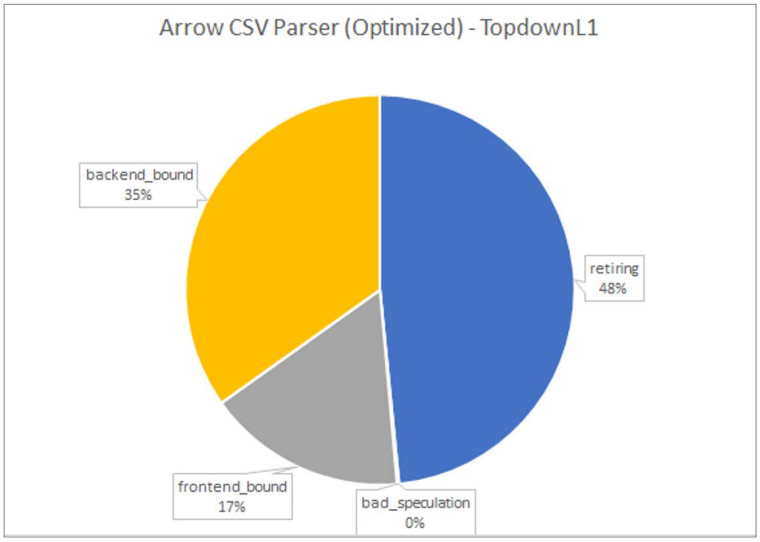
Re-plot the Stage 1 Top-down Metrics

TABLE 3
Apache Arrow Performance Comparison Between Baseline and Optimized Code

	Cycles	Instructions	IPC	Benchmark Score
Original Baseline	5.45E+9	2.53E+10	4.64	1.10G/s
Optimized	2.97E+9	1.28E+10	4.30	1.99G/s

Let us now collect the top-down level 1 metrics for further evaluation. Figure 15 shows high retiring rate for this workload with 35% *backend_bound* metric. Let us now look at how the top-down level 1 metrics compare before and after optimization in Table 4.

FIG. 15
Arrow CSV Parser Optimized: TopdownL1



From Table 4, it can be observed that after optimization, the top-down level 1 analysis shows a drop in *retiring* percentage from 53% to 48% and *backend_bound* metric increases from 24% to 35%. This shows increased backend pressure with reduction in retiring caused by efficient vector unit utilization.

TABLE 4
Apache Arrow Topdown Level 1 Metrics: Baseline and Optimized Code

	retiring	frontend_bound	backend_bound	bad_speculation
Baseline	53%	23%	24%	0%
Optimized	48%	17%	35%	0%

Re-plot the Operation Mix Metrics

Let us now collect the operation mix metrics for further evaluation. Figure 16 shows *simd_percentage* of 28% that reflects the SIMD optimization conducted with a reduction in *integer_dp_percentage*.

FIG. 16
Apache Arrow Optimized: Operation Mix

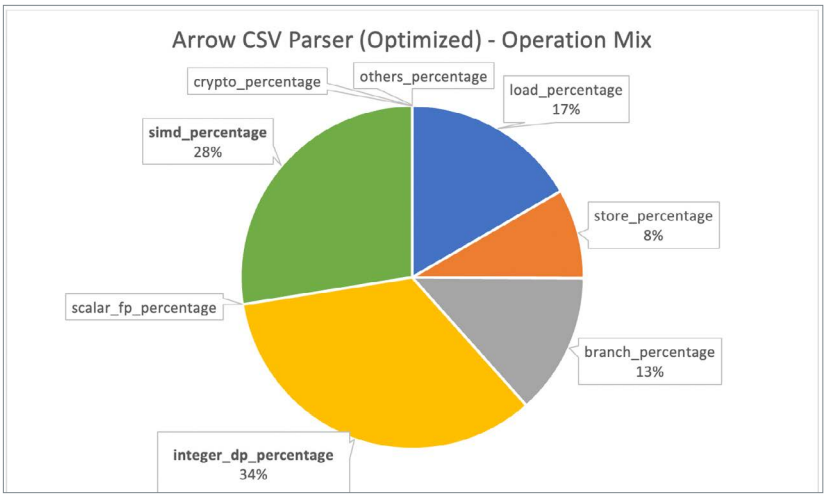


Table 5 below shows the comparisons of the Operation Mix metrics before and after optimization. Compared with Operation Mix before optimization, *integer_dp_percentage* drops half from ~2/3 of the total. After vectorization, *simd_percentage* now occupies over 1/4th of the total operations. operations to ~1/3 of the operations.

TABLE 5
Apache Arrow Operation Mix Metrics: Baseline and Optimized Code

	integer_dp_ percentage	simd_percentage	load_percentage	store_percentage	branch_ percentage
Baseline	63%	0%	18%	7%	12%
Optimized	34%	28%	17%	8%	13%

4.4. Summary

The original baseline Arrow CSV parser performance was quite good from an IPC point of view. However, by conducting hotspot analysis with the top-down methodology proposed in this paper, we could identify hotspot in the code where high volumes of integer processing instructions are being executed. This provided us insights into opportunities for optimization leveraging Arm Neon technology. The optimized code improves throughout by about 80% and cuts down half of the total instructions by exploiting vector instructions.

For code optimization, it is always helpful to identify hotspots in the code that are areas of pipeline bottlenecks or that are stealing most of the CPU cycles. Arm top-down methodology presented in this document methodology can help us find the bottlenecks quickly and focus optimization exercises on the performance-critical code.

The upstream PR of this optimization is available at: github.com/apache/arrow/pull/11896.

If you try this methodology and have any further comments or code optimization stories to share, we would love to hear from you. Feel free to reach out to us at sw-ecosystem@arm.com.

5. Glossary

Term Meaning

CMO Cache Maintenance Operations

CPU Central Processing Unit

LSU Load Store Unit

MMU Memory Management Unit

PE Processing Element

PMU Performance Monitoring Unit

SiP Silicon Provider

SLC System Level Cache

TLB Translation Lookaside Buffer

SIMD Single Instruction Multiple Data

6. Acknowledgements

We would like to thank David Murray, Alessandro Di Bella, Nick Forrington, Manisha Malhotra, Ryan Harkin and Al Grant for their support and review of this paper.

7. References

[01] Arm®, "Arm® Neoverse V1 Technical Reference Manual", developer.arm.com/documentation/101427/latest/

[02] Arm®, "Arm® Neoverse V1 PMU Guide developer.arm.com/documentation/PJDOC-1063724031-605393/r1p1/

[03] Arm®, "Arm® Neoverse™ V1 Software Optimization Guide Documentation", developer.arm.com/documentation/pjdoc466751330-9685/latest/

[04] Arm®, "Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile Documentation" developer.arm.com/docs/ddi0487/latest

[05] Arm®, "Arm® Neoverse N1 Core: Performance Analysis Methodology armkeil.blob.core.windows.net/developer/Files/pdf/white-paper/neoverse-n1-core-performance-v2.pdf

[06] "Arm Telemetry Solution Repository" gitlab.arm.com/telemetry-solution/telemetry-solution

[07] "Arm PMU Event Repository", github.com/ARM-software/data

[08] "Apache Arrow", github.com/apache/arrow

[09] en.wikipedia.org/wiki/Column-oriented_DBMS#Access_time

8. Appendix A. UStress Data

A.1. UStress Tests: MPKI

Table 6 shows the results of the key MPKI metric measurements for all the UStress validation tests.

TABLE 6
UStress Tests: MPKI

		branch_mpki	l1d_cache_mpki	l1i_cache_mpki	l2_cache_mpki	l1d_tlb_mpki	l1i_tlb_mpki	l2_tlb_mpki	dtlb_mpki	itlb_mpki
Branch	branch_direct_workload	54.45	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00
	branch_indirect_workload	21.86	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00
	call_return_workload	45.76	0.18	0.01	0.00	0.11	0.00	0.00	0.00	0.00
D-Cache	l1d_cache_workload	0.66	331.59	0.03	0.02	0.03	0.00	0.00	0.00	0.00
	l2d_cache_workload	0.05	317.08	0.13	144.35	77.89	0.00	2.39	2.20	0.00
I-Cache	l1i_cache_workload	243.70	0.02	1637.87	0.03	0.03	0.00	0.00	0.00	0.00
DTLB	l1d_tlb_workload	0.00	0.04	0.01	0.02	165.28	0.00	0.03	0.00	0.00
Arithmetic	mul32_workload	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00
	mul64_workload	0.00	0.00	0.01	0.01	0.00	0.00	0.00	0.00	0.00
	mac32_workload	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00
	mac64_workload	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00
	div32_workload	0.00	0.01	0.02	0.01	0.00	0.00	0.00	0.00	0.00
	div64_workload	0.00	0.01	0.01	0.01	0.00	0.00	0.00	0.00	0.00
	fpmul_workload	0.00	0.00	0.01	0.01	0.00	0.00	0.00	0.00	0.00
	fpmac_workload	0.00	0.01	0.01	0.01	0.00	0.00	0.00	0.00	0.00
	fpdiv_workload	0.01	0.01	0.03	0.02	0.01	0.00	0.00	0.00	0.00
	fpsqrt_workload	0.00	0.01	0.01	0.01	0.00	0.00	0.00	0.00	0.00
	int2double_workload	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	double2int_workload	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Memory	memcpy_workload	0.24	0.01	0.02	0.00	0.00	0.00	0.00	0.00	0.00
	store_buffer_full_workload	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	load_after_store_workload	0.01	0.01	0.02	0.01	0.01	0.00	0.00	0.00	0.00

A.2. UStress Tests: Miss Ratio

Table 7 shows the results of the key Miss ratio metric measurements for all the UStress validation tests.

TABLE 7
UStress Tests: Miss Ratio

		branch_misprediction_ratio	l1d_cache_miss_ratio	l1i_cache_miss_ratio	l2_cache_miss_ratio	l1d_tlb_miss_ratio	l1i_tlb_miss_ratio	l2_tlb_miss_ratio	dtlb_walk_ratio	itlb_walk_ratio
Branch	branch_direct_workload	46.15	1.37	0.00	17.02	1.06	0.00	12.32	0.09	0.00
	branch_indirect_workload	46.81	0.00	0.00	13.08	0.00	0.00	4.87	0.00	0.00
	call_return_workload	32.03	0.11	0.00	0.01	0.06	0.00	0.24	0.00	0.00
D-Cache	l1d_cache_workload	0.20	99.74	0.01	0.00	0.01	0.00	0.14	0.00	0.00
	l2d_cache_workload	0.02	95.93	0.04	22.63	23.57	0.01	3.06	0.67	0.01
I-Cache	l1i_cache_workload	74.82	0.00	91.22	0.00	0.01	0.00	5.72	0.00	0.00
DTLB	l1d_tlb_workload	0.00	0.03	0.00	1.52	99.10	0.00	0.02	0.00	0.00
Arithmetic	mul32_workload	0.00	1.16	0.00	18.10	0.79	0.02	5.88	0.09	0.01
	mul64_workload	0.00	1.50	0.01	17.51	1.06	0.02	11.72	0.10	0.01
	mac32_workload	0.00	1.31	0.00	18.08	0.73	0.02	10.44	0.11	0.01
	mac64_workload	0.00	1.24	0.01	14.96	0.78	0.02	9.34	0.12	0.01
	div32_workload	0.00	1.18	0.01	18.41	0.67	0.02	12.38	0.08	0.01
	div64_workload	0.00	1.13	0.01	17.63	0.70	0.02	10.58	0.07	0.01
	fpmul_workload	0.00	1.26	0.01	17.13	0.72	0.02	8.52	0.09	0.01
	fpmac_workload	0.00	1.27	0.01	19.21	0.72	0.02	11.65	0.10	0.01
	fpdiv_workload	0.01	1.47	0.02	16.82	0.71	0.02	10.25	0.08	0.01
	fpsqrt_workload	0.00	1.32	0.01	16.56	0.81	0.02	12.09	0.08	0.01
	int2double_workload	0.00	1.27	0.00	17.51	0.75	0.00	13.22	0.09	0.00
	double2int_workload	0.00	1.24	0.00	18.39	0.80	0.00	11.49	0.09	0.00
Memory	memcpy_workload	0.19	0.00	0.01	0.00	0.00	0.00	1.42	0.00	0.00
	store_buffer_full_workload	0.00	0.00	0.00	18.18	0.00	0.00	8.85	0.00	0.00
	load_after_store_workload	0.24	0.00	0.00	17.71	0.00	0.00	4.72	0.00	0.00

9. Appendix B1. Neoverse V1 Events

B1.1 Bus

TABLE 8
Neoverse V1 Events: Bus

Event Num	Event Mnemonic	Description
0x0019	BUS_ACCESS	Counts memory transactions issued by the CPU to the external bus, including snoop requests and snoop responses. Each beat of data is counted individually.
0x0060	BUS_ACCESS_RD	Counts memory read transactions seen on the external bus. Each beat of data is counted individually.
0x0061	BUS_ACCESS_WR	Counts memory write transactions seen on the external bus. Each beat of data is counted individually.
0x001D	BUS_CYCLES	Counts bus cycles in the CPU. Bus cycles represent a clock cycle in which a transaction could be sent or received on the interface from the CPU to the external bus. Since that interface is driven at the same clock speed as the CPU, this event is a duplicate of CPU_CYCLES.

B1.2 Chain

TABLE 9
Neoverse V1 Events: Chain

Event Num	Event Mnemonic	Description
0x001E	CHAIN	Counts whenever the even numbered PMU counter registers overflow. This event is used when the even/odd pairs of registers are used as a single counter.

TABLE 10

Neoverse V1 Events: Exception

B1.3 Exception

Event Num	Event Mnemonic	Description
0x0084	EXC_DABORT	Counts exceptions that are taken locally and are caused by data aborts or SErrors. Conditions that could cause those exceptions are attempting to read or write memory where the MMU generates a fault, attempting to read or write memory with a misaligned address, interrupts from the nSEI inputs and internally generated SErrors.
0x0087	EXC_FIQ	Counts FIQ exceptions including the virtual FIQs that are taken locally.
0x008A	EXC_HVC	Counts HVC exceptions taken to EL2.
0x0086	EXC_IRQ	Counts IRQ exceptions including the virtual IRQs that are taken locally.
0x0083	EXC_PABORT	Counts synchronous exceptions that are taken locally and caused by Instruction Aborts.
0x000A	EXC_RETURN	Counts any architecturally executed exception return instructions. Eg: AArch64: ERET
0x0088	EXC_SMC	Counts SMC exceptions take to EL3.
0x0082	EXC_SVC	Counts SVC exceptions taken locally.
0x0009	EXC_TAKEN	Counts any taken architecturally visible exceptions such as IRQ, FIQ, SError, and other synchronous exceptions. Exceptions are counted whether or not they are taken locally.
0x008C	EXC_TRAP_DABORT	Counts exceptions which are traps not taken locally and are caused by Data Aborts or SError interrupts. Conditions that could cause those exceptions are: 1. Attempting to read or write memory where the MMU generates a fault, 2. Attempting to read or write memory with a misaligned address, 3. Interrupts from the SEI input, 4. internally generated SErrors.
0x008F	EXC_TRAP_FIQ	Counts FIQs which are not taken locally but taken from EL0, EL1, or EL2 to EL3 (which would be the normal behavior for FIQs when not executing in EL3).

Event Num	Event Mnemonic	Description
0x008E	EXC_TRAP_IRQ	Counts IRQ exceptions including the virtual IRQs that are not taken locally.
0x008D	EXC_TRAP_OTHER	Counts the number of synchronous trap exceptions which are not taken locally and are not SVC, SMC, HVC, data aborts, Instruction Aborts, or interrupts.
0x008B	EXC_TRAP_PABORT	Counts exceptions which are traps not taken locally and are caused by Instruction Aborts. For example, attempting to execute an instruction with a misaligned PC.
0x0081	EXC_UNDEF	Counts the number of synchronous exceptions which are taken locally that are due to attempting to execute an instruction that is UNDEFINED. Attempting to execute instruction bit patterns that have not been allocated. Attempting to execute instructions when they are disabled. Attempting to execute instructions at an inappropriate Exception level. Attempting to execute an instruction when the value of PSTATE.IL is 1.

TABLE 11

Neoverse V1 Events: L1D_Cache

B1.4 L1D_Cache

Event Num	Event Mnemonic	Description
0x0004	L1D_CACHE	Counts level 1 data cache accesses from any load/store operations. Atomic operations that resolve in the CPUs caches (near atomic operations) counts as both a write access and read access. Each access to a cache line is counted including the multiple accesses caused by single instructions such as LDM or STM. Each access to other level 1 data or unified memory structures, for example refill buffers, write buffers, and write-back buffers, are also counted.
0x0048	L1D_CACHE_INVAL	Counts each explicit invalidation of a cache line in the level 1 data cache caused by: - Cache Maintenance Operations (CMO) that operate by a virtual address. - Broadcast cache coherency operations from another CPU in the system. This event does not count for the following conditions: 1. A cache refill invalidates a cache line. 2. A CMO which is executed on that CPU and invalidates a cache line specified by set/way. Note that CMOs that operate by set/way cannot be broadcast from one CPU to another.
0x0039	L1D_CACHE_LMISS_RD	Counts cache line refills into the level 1 data cache from any memory read operations, that incurred additional latency.
0x0040	L1D_CACHE_RD	Counts level 1 data cache accesses from any load operation. Near atomic operations that resolve in the CPUs caches counts as both a write access and read access.
0x0003	L1D_CACHE_REFILL	Counts level 1 data cache refills caused by speculatively executed load or store operations that missed in the level 1 data cache. This event only counts one event per cache line. This event does not count cache line allocations from preload instructions or from hardware cache prefetching.
0x0044	L1D_CACHE_REFILL_INNER	Counts level 1 data cache refills where the cache line data came from caches inside the immediate cluster of the core.

Event Num	Event Mnemonic	Description
0x0045	L1D_CACHE_REFILL_OUTER	Counts level 1 data cache refills for which the cache line data came from outside the immediate cluster of the core, like an SLC in the system interconnect or DRAM.
0x0042	L1D_CACHE_REFILL_RD	Counts level 1 data cache refills caused by speculatively executed load instructions where the memory read operation misses in the level 1 data cache. This event only counts one event per cache line.
0x0043	L1D_CACHE_REFILL_WR	Counts level 1 data cache refills caused by speculatively executed store instructions where the memory write operation misses in the level 1 data cache. This event only counts one event per cache line.
0x0015	L1D_CACHE_WB	Counts write-backs of dirty data from the L1 data cache to the L2 cache. This occurs when either a dirty cache line is evicted from L1 data cache and allocated in the L2 cache or dirty data is written to the L2 and possibly to the next level of cache. This event counts both victim cache line evictions and cache write-backs from snoops or cache maintenance operations. The following cache operations are not counted: 1. Invalidations which do not result in data being transferred out of the L1 (such as evictions of clean data), 2. Full line writes which write to L2 without writing L1, such as write streaming mode.
0x0047	L1D_CACHE_WB_CLEAN	Counts write-backs from the level 1 data cache that are a result of a coherency operation made by another CPU. Event count includes cache maintenance operations.
0x0046	L1D_CACHE_WB_VICTIM	Counts dirty cache line evictions from the level 1 data cache caused by a new cache line allocation. This event does not count evictions caused by cache maintenance operations.
0x0041	L1D_CACHE_WR	Counts level 1 data cache accesses generated by store operations. This event also counts accesses caused by a DC ZVA (data cache zero, specified by virtual address) instruction. Near atomic operations that resolve in the CPUs caches count as a write access and read access.

TABLE 12
Neoverse V1 Events: L1I_Cache

B1.5 L1I_Cache

Event Num	Event Mnemonic	Description
0x0014	L1I_CACHE	Counts instruction fetches which access the level 1 instruction cache. Instruction cache accesses caused by cache maintenance operations are not counted.
0x4006	L1I_CACHE_LMISS	Counts cache line refills into the level 1 instruction cache, that incurred additional latency.
0x0001	L1I_CACHE_REFILL	Counts cache line refills in the level 1 instruction cache caused by a missed instruction fetch. Instruction fetches may include accessing multiple instructions, but the single cache line allocation is counted once.

TABLE 13
Neoverse V1 Events: L2_Cache

B1.6 L2_Cache

Event Num	Event Mnemonic	Description
0x0016	L2D_CACHE	Counts level 2 cache accesses. Level 2 cache is a unified cache for data and instruction accesses. Accesses are for misses in the first level caches or translation resolutions due to accesses. This event also counts write back of dirty data from level 1 data cache to the L2 cache.
0x0020	L2D_CACHE_ALLOCATE	TBD
0x0058	L2D_CACHE_INVAL	Counts each explicit invalidation of a cache line in the level 2 cache by cache maintenance operations that operate by a virtual address, or by external coherency operations. This event does not count if either: 1. A cache refill invalidates a cache line or, 2. A Cache Maintenance Operation (CMO), which invalidates a cache line specified by set/way, is executed on that CPU. CMOs that operate by set/way cannot be broadcast from one CPU to another.

Event Num	Event Mnemonic	Description
0x4009	L2D_CACHE_LMISS_RD	Counts cache line refills into the level 2 unified cache from any memory read operations that incurred additional latency.
0x0050	L2D_CACHE_RD	Counts level 2 cache accesses due to memory read operations. Level 2 cache is a unified cache for data and instruction accesses, accesses are for misses in the level 1 caches or translation resolutions due to accesses.
0x0017	L2D_CACHE_REFILL	Counts cache line refills into the level 2 cache. level 2 cache is a unified cache for data and instruction accesses. Accesses are for misses in the level 1 caches or translation resolutions due to accesses.
0x0052	L2D_CACHE_REFILL_RD	Counts refills for memory accesses due to memory read operation counted by L2D_CACHE_RD. level 2 cache is a unified cache for data and instruction accesses, accesses are for misses in the level 1 caches or translation resolutions due to accesses.
0x0053	L2D_CACHE_REFILL_WR	Counts refills for memory accesses due to memory write operation counted by L2D_CACHE_WR. level 2 cache is a unified cache for data and instruction accesses, accesses are for misses in the level 1 caches or translation resolutions due to accesses.

Event Num	Event Mnemonic	Description
0x0018	L2D_CACHE_WB	Counts write-backs of data from the L2 cache to outside the CPU. This includes snoops to the L2 (from other CPUs) which return data even if the snoops cause an invalidation. L2 cache line invalidations which do not write data outside the CPU and snoops which return data from an L1 cache are not counted. Data would not be written outside the cache when invalidating a clean cache line.
0x0057	L2D_CACHE_WB_CLEAN	Counts write-backs from the level 2 cache that are a result of either: 1. Cache maintenance operations, 2. Snoop responses or, 3. Direct cache transfers to another CPU due to a forwarding snoop request.
0x0056	L2D_CACHE_WB_VICTIM	Counts evictions from the level 2 cache because of a line being allocated into the L2 cache.
0x0051	L2D_CACHE_WR	Counts level 2 cache accesses due to memory write operations. Level 2 cache is a unified cache for data and instruction accesses, accesses are for misses in the level 1 caches or translation resolutions due to accesses.

B1.7 L3_Cache

TABLE 14
Neoverse V1 Events: L3_Cache

Event Num	Event Mnemonic	Description
0x002B	L3D_CACHE	Counts level 3 cache accesses. Level 3 cache is a unified cache for data and instruction accesses. Accesses are for misses in the lower level caches or translation resolutions due to accesses.
0x0029	L3D_CACHE_ALLOCATE	Counts level 3 cache line allocates that do not fetch data from outside the level 3 data or unified cache. For example, allocates due to streaming stores.
0x400B	L3D_CACHE_LMISS_RD	Counts any cache line refill into the level 3 cache from memory read operations that incurred additional latency.
0x00A0	L3D_CACHE_RD	TBD
0x002A	L3D_CACHE_REFILL	Counts level 3 accesses that receive data from outside the L3 cache.

TABLE 15
Neoverse V1 Events: LL_Cache

B1.8 LL_Cache

Event Num	Event Mnemonic	Description
0x0037	LL_CACHE_MISS_RD	Counts read transactions that were returned from outside the core cluster but missed in the system level cache. This event counts when the system register CPUECTLR.EXTLLC bit is set. This event counts read transactions returned from outside the core if those transactions are missed in the System level Cache. The data source of the transaction is indicated by a field in the CHI transaction returning to the CPU. This event does not count reads caused by cache maintenance operations.
0x0036	LL_CACHE_RD	Counts read transactions that were returned from outside the core cluster. This event counts when the system register CPUECTLR.EXTLLC bit is set. This event counts read transactions returned from outside the core if those transactions are either hit in the System Level Cache or missed in the SLC and are returned from any other external sources.

TABLE 16
Neoverse V1 Events: Memory

B1.9 Memory

Event Num	Event Mnemonic	Description
0x001A	MEMORY_ERROR	Counts any detected correctable or uncorrectable physical memory errors (ECC or parity) in protected CPUs RAMs. On the core, this event counts errors in the caches (including data and tag rams). Any detected memory error (from either a speculative and abandoned access, or an architecturally executed access) is counted. Note that errors are only detected when the actual protected memory is accessed by an operation.
0x0013	MEM_ACCESS	Counts memory accesses issued by the CPU load store unit, where those accesses are issued due to load or store operations. This event counts any memory access, no matter whether the data is received from any level of cache hierarchy or external memory. If memory accesses are broken up into smaller transactions than what were specified in the load or store instructions, then the event counts those smaller memory transactions.

Event Num	Event Mnemonic	Description
0x0066	MEM_ACCESS_RD	Counts memory accesses issued by the CPU due to load operations. The event counts any memory load access, no matter whether the data is received from any level of cache hierarchy or external memory. The event also counts atomic load operations. If memory accesses are broken up by the load/store unit into smaller transactions that are issued by the bus interface, then the event counts those smaller transactions.
0x0067	MEM_ACCESS_WR	Counts memory accesses issued by the CPU due to store operations. The event counts any memory store access, no matter whether the data is located in any level of cache or external memory. The event also counts atomic load and store operations. If memory accesses are broken up by the load/store unit into smaller transactions that are issued by the bus interface, then the event counts those smaller transactions.
0x0031	REMOTE_ACCESS	Counts accesses to another chip, which is implemented as a different CMN mesh in the system. If the CHI bus response back to the core indicates that the data source is from another chip (mesh), then the counter is updated. If no data is returned, even if the system snoops another chip/mesh, then the counter is not updated.

TABLE 17
Neoverse V1 Events: Retired

B1.10 Retired

Event Num	Event Mnemonic	Description
0x0022	BR_MIS_PRED_RETIRED	Counts branches counted by BR_RETIRED which were mispredicted and caused a pipeline flush.
0x0021	BR_RETIRED	Counts architecturally executed branches, whether the branch is taken or not. Instructions that explicitly write to the PC are also counted.

Event Num	Event Mnemonic	Description
0x000B	CID_WRITE_RETIRED	Counts architecturally executed writes to the CONTEXTIDR register, which usually contain the kernel PID and can be output with hardware trace.
0x0008	INST_RETIRED	Counts instructions that have been architecturally executed.
0x003A	OP_RETIRED	Counts micro-operations that are architecturally executed. This is a count of number of micro-operations retired from the commit queue in a single cycle.
0x0000	SW_INCR	Counts software writes to the PMSWINC_EL0 (software PMU increment) register. The PMSWINC_EL0 register is a manually updated counter for use by application software. This event could be used to measure any user program event, such as accesses to a particular data structure (by writing to the PMSWINC_EL0 register each time the data structure is accessed). To use the PMSWINC_EL0 register and event, developers must insert instructions that write to the PMSWINC_EL0 register into the source code. Since the SW_INCR event records writes to the PMSWINC_EL0 register, there is no need to do a read/increment/write sequence to the PMSWINC_EL0 register.
0x001C	TTBR_WRITE_RETIRED	Counts architectural writes to TTBR0/1_EL1. If virtualization host extensions are enabled (by setting the HCR_EL2.E2H bit to 1), then accesses to TTBR0/1_EL1 that are redirected to TTBR0/1_EL2, or accesses to TTBR0/1_EL12, are counted. TTBRn registers are typically updated when the kernel is swapping user-space threads or applications.

TABLE 18
Neoverse V1 Events: SPE

B1.11 SPE

Event Num	Event Mnemonic	Description
0x4003	SAMPLE_COLLISION	Counts statistical profiling samples that have collided with a previous sample and so therefore not taken.

Event Num	Event Mnemonic	Description
0x4001	SAMPLE_FEED	Counts statistical profiling samples taken for sampling.
0x4002	SAMPLE_FILTRATE	Counts statistical profiling samples taken which are not removed by filtering.
0x4000	SAMPLE_POP	Counts statistical profiling sample population, the count of all operations that could be sampled but may or may not be chosen for sampling.

B1.12 Spec_Operation

TABLE 19
Neoverse V1 Events: Spec_Operation

Event Num	Event Mnemonic	Description
0x8005	ASE_INST_SPEC	Counts speculatively executed Advanced SIMD operations.
0x0074	ASE_SPEC	Counts speculatively executed Advanced SIMD operations excluding load, store and move micro-operations that move data to or from SIMD (vector) registers.
0x0078	BR_IMMED_SPEC	Counts immediate branch operations which are speculatively executed.
0x007A	BR_INDIRECT_SPEC	Counts indirect branch operations including procedure returns, which are speculatively executed. This includes operations that force a software change of the PC, other than exception-generating operations. Eg: BR Xn, RET
0x0010	BR_MIS_PRED	Counts branches which are speculatively executed and mispredicted.
0x0012	BR_PRED	Counts branches speculatively executed and were predicted right.
0x0079	BR_RETURN_SPEC	Counts procedure return operations (RET) which are speculatively executed.
0x0077	CRYPTO_SPEC	Counts speculatively executed cryptographic operations except for PMULL and VMULL operations.
0x007E	DMB_SPEC	Counts DMB operations that are speculatively issued to the Load/Store unit in the CPU. This event does not count implied barriers from load acquire/store release operations.

Event Num	Event Mnemonic	Description
0x0073	DP_SPEC	Counts speculatively executed logical or arithmetic instructions such as MOV/MVN operations.
0x007D	DSB_SPEC	Counts DSB operations that are speculatively issued to Load/Store unit in the CPU.
0x001B	INST_SPEC	Counts operations that have been speculatively executed.
0x007C	ISB_SPEC	Counts ISB operations that are executed.
0x006C	LDREX_SPEC	Counts Load-Exclusive operations (such as LDREX or LDX) that have been speculatively executed. Eg: LDREX, LDX
0x0070	LD_SPEC	Counts speculatively executed load operations including Single Instruction Multiple Data (SIMD) load operations.
0x003B	OP_SPEC	Counts micro-operations speculatively executed. This is the count of the number of micro-operations dispatched in a cycle.
0x0076	PC_WRITE_SPEC	Counts speculatively executed operations which cause software changes of the PC. Those operations include all taken branch operations.
0x0090	RC_LD_SPEC	Counts any load acquire operations that are speculatively executed. Eg: LDAR, LDARH, LDARB
0x0091	RC_ST_SPEC	Counts any store release operations that are speculatively executed. Eg: STLR, STLRH, STLRB'
0x006E	STREX_FAIL_SPEC	Counts store-exclusive operations that have been speculatively executed and have not successfully completed the store operation.
0x006D	STREX_PASS_SPEC	Counts store-exclusive operations that have been speculatively executed and have successfully completed the store operation.

Event Num	Event Mnemonic	Description
0x006F	STREX_SPEC	Counts store-exclusive operations that have been speculatively executed.
0x0071	ST_SPEC	Counts speculatively executed store operations including Single Instruction Multiple Data (SIMD) store operations.
0x006A	UNALIGNED_LDST_SPEC	Counts unaligned memory operations issued by the CPU. This event counts unaligned accesses (as defined by the actual instruction), even if they are subsequently issued as multiple aligned accesses.
0x0068	UNALIGNED_LD_SPEC	Counts unaligned memory read operations issued by the CPU. This event counts unaligned accesses (as defined by the actual instruction), even if they are subsequently issued as multiple aligned accesses. The event does not count preload operations (PLD, PLI).
0x0069	UNALIGNED_ST_SPEC	Counts unaligned memory write operations issued by the CPU. This event counts unaligned accesses (as defined by the actual instruction), even if they are subsequently issued as multiple aligned accesses.
0x0075	VFP_SPEC	Counts speculatively executed floating point operations. This event does not count operations that move data to or from floating point (vector) registers.

TABLE 20
Neoverse V1 Events: Stall

B1.13 Stall

Event Num	Event Mnemonic	Description
0x003C	STALL	Counts cycles when no operations are sent to the rename unit from the frontend or from the rename unit to the backend for any reason (either frontend or backend stall).
0x0024	STALL_BACKEND	Counts cycles whenever the rename unit is unable to send any micro-operations to the backend of the pipeline because of backend resource constraints. Backend resource constraints can include issue stage fullness, execution stage fullness, or other internal pipeline resource fullness. All the backend slots were empty during the cycle when this event counts.

Event Num	Event Mnemonic	Description
0x4005	STALL_BACKEND_MEM	Counts cycles when the backend is stalled because there is a pending demand load request in progress in the last level core cache.
0x0023	STALL_FRONTEND	Counts cycles when frontend could not send any micro-operations to the rename stage because of frontend resource stalls caused by fetch memory latency or branch prediction flow stalls. All the frontend slots were empty during the cycle when this event counts.
0x003F	STALL_SLOT	Counts slots per cycle in which no operations are sent to the rename unit from the frontend or from the rename unit to the backend for any reason (either frontend or backend stall).
0x003D	STALL_SLOT_BACKEND	Counts slots per cycle in which no operations are sent from the rename unit to the backend due to backend resource constraints.
0x003E	STALL_SLOT_FRONTEND	Counts slots per cycle in which no operations are sent to the rename unit from the frontend due to frontend resource constraints.

TABLE 21
Neoverse V1 Events: General

B1.14 General

Event Num	Event Mnemonic	Description
0x4004	CNT_CYCLES	Counts constant frequency cycles.
0x0011	CPU_CYCLES	Counts CPU clock cycles (not timer cycles). The clock measured by this event is defined as the physical clock driving the CPU logic.

TABLE 22
Neoverse V1 Events: TLB

B1.15 TLB

Event Num	Event Mnemonic	Description
0x0034	DTLB_WALK	Counts data memory translation table walks caused by a miss in the L2 TLB driven by a memory access. Note that partial translations that also cause a table walk are counted. This event does not count table walks caused by TLB maintenance operations.

Event Num	Event Mnemonic	Description
0x0035	ITLB_WALK	Counts instruction memory translation table walks caused by a miss in the L2 TLB driven by a memory access. Partial translations that also cause a table walk are counted. This event does not count table walks caused by TLB maintenance operations.
0x0025	L1D_TLB	Counts level 1 data TLB accesses caused by any memory load or store operation. Note that load or store instructions can be broken up into multiple memory operations. This event does not count TLB maintenance operations.
0x004E	L1D_TLB_RD	Counts level 1 data TLB accesses caused by memory read operations. This event counts whether the access hits or misses in the TLB. This event does not count TLB maintenance operations.
0x0005	L1D_TLB_REFILL	Counts level 1 data TLB accesses that resulted in TLB refills. If there are multiple misses in the TLB that are resolved by the refill, then this event only counts once. This event counts for refills caused by preload instructions or hardware prefetch accesses. This event counts regardless of whether the miss hits in L2 or results in a translation table walk. This event will not count if the translation table walk results in a fault (such as a translation or access fault), since there is no new translation created for the TLB. This event will not count on an access from an AT (Address Translation) instruction.
0x004C	L1D_TLB_REFILL_RD	Counts level 1 data TLB refills caused by memory read operations. If there are multiple misses in the TLB that are resolved by the refill, then this event only counts once. This event counts for refills caused by preload instructions or hardware prefetch accesses. This event counts regardless of whether the miss hits in L2 or results in a translation table walk. This event will not count if the translation table walk results in a fault (such as a translation or access fault), since there is no new translation created for the TLB. This event will not count on an access from an Address Translation (AT) instruction.

Event Num	Event Mnemonic	Description
0x004D	L1D_TLB_REFILL_WR	Counts level 1 data TLB refills caused by data side memory write operations. If there are multiple misses in the TLB that are resolved by the refill, then this event only counts once. This event counts for refills caused by preload instructions or hardware prefetch accesses. This event counts regardless of whether the miss hits in L2 or results in a translation table walk. This event will not count if the table walk results in a fault (such as a translation or access fault), since there is no new translation created for the TLB. This event will not count with an access from an Address Translation (AT) instruction.
0x004F	L1D_TLB_WR	Counts any L1 data side TLB accesses caused by memory write operations. This event counts whether the access hits or misses in the TLB. This event does not count TLB maintenance operations.
0x002D	L2D_TLB_REFILL	Counts level 2 TLB refills caused by memory operations from both data and instruction fetch, except for those caused by TLB maintenance operations and hardware prefetches.
0x005C	L2D_TLB_REFILL_RD	Counts level 2 TLB refills caused by memory read operations from both data and instruction fetch except for those caused by TLB maintenance operations or hardware prefetches.
0x005D	L2D_TLB_REFILL_WR	Counts level 2 TLB refills caused by memory write operations from both data and instruction fetch except for those caused by TLB maintenance operations.
0x005F	L2D_TLB_WR	Counts level 2 TLB accesses caused by memory write operations from both data and instruction fetch except for those caused by TLB maintenance operations.

Event Num	Event Mnemonic	Description
0x002D	L2D_TLB_REFILL	Counts level 2 TLB refills caused by memory operations from both data and instruction fetch, except for those caused by TLB maintenance operations and hardware prefetches.
0x005C	L2D_TLB_REFILL_RD	Counts level 2 TLB refills caused by memory read operations from both data and instruction fetch except for those caused by TLB maintenance operations or hardware prefetches.
0x005D	L2D_TLB_REFILL_WR	Counts level 2 TLB refills caused by memory write operations from both data and instruction fetch except for those caused by TLB maintenance operations.
0x005F	L2D_TLB_WR	Counts level 2 TLB accesses caused by memory write operations from both data and instruction fetch except for those caused by TLB maintenance operations.

TABLE 23
Neoverse V1 Events: SVE

B1.16 SVE

Event Num	Event Mnemonic	Description
0x80C1	FP_FIXED_OPS_SPEC	Counts speculatively executed non-scalable single precision floating point operations.
0x80C0	FP_SCALE_OPS_SPEC	Counts speculatively executed scalable single precision floating point operations.
0x8006	SVE_INST_SPEC	Counts speculatively executed operations that are SVE operations.
0x80BD	SVE_LDFF_FAULT_SPEC	Counts speculatively executed SVE first fault or non-fault load operations that clear at least one bit in the FFR.
0x80BC	SVE_LDFF_SPEC	Counts speculatively executed SVE first fault or non-fault load operations.
0x8075	SVE_PRED_EMPTY_SPEC	Counts speculatively executed predicated SVE operations with no active predicate elements.
0x8076	SVE_PRED_FULL_SPEC	Counts speculatively executed predicated SVE operations with all predicate elements active.
0x8077	SVE_PRED_PARTIAL_SPEC	Counts speculatively executed predicated SVE operations with at least one but not all active predicate elements.
0x8074	SVE_PRED_SPEC	Counts speculatively executed predicated SVE operations.

10. Appendix C1. Neoverse V1 Metrics

C.1.1 Metric Group: Topdown_L1

This metric group contains the first set of metrics to begin topdown analysis of application performance, which provide the percentage distribution of processor pipeline utilization.

C1.1.1 Metric Descriptions

TABLE 24
Neoverse V1 Metrics: Topdown_L1, Metric Descriptions

Metric Name	Metric Title	Metric Description
backend_bound	Backend Bound	This metric is the percentage of total slots that were stalled due to resource constraints in the backend of the processor.
bad_speculation	Bad Speculation	This metric is the percentage of total slots that executed operations and didn't retire due to a pipeline flush. This indicates cycles that were utilized but inefficiently.
frontend_bound	Frontend Bound	This metric is the percentage of total slots that were stalled due to resource constraints in the frontend of the processor.
retiring	Retiring	This metric is the percentage of total slots that retired operations, which indicates cycles that were utilized efficiently.

C1.1.2 Metric Formula

TABLE 25
Neoverse V1 Metrics: Topdown_L1, Metric Formula

Metric Name	Metric Formula	Unit
backend_bound	100 * STALL_SLOT_BACKEND/(CPU_CYCLES *8)	percent of slots
bad_speculation	100 * (((1 - (OP_RETIRED/OP_SPEC)) * (1 - (STALL_SLOT/(CPU_CYCLES * 8)))) + ((BR_MIS_PRED * 4)/CPU_CYCLES))	percent of slots
frontend_bound	100 * ((STALL_SLOT_FRONTEND/(CPU_CYCLES * 8)) - ((BR_MIS_PRED * 4)/CPU_CYCLES))	percent of slots
retiring	100 * (OP_RETIRED/OP_SPEC) * (1 - (STALL_SLOT/(CPU_CYCLES * 8)))	percent of slots

TABLE 26
Neoverse V1 Metrics: Topdown_L1, Metric Events

C1.1.3 Metric Events

Metric Name	Metric Events
backend_bound	CPU_CYCLES, STALL_SLOT_BACKEND
bad_speculation	CPU_CYCLES, OP_SPEC, BR_MIS_PRED, STALL_SLOT, OP_RETIRED
frontend_bound	CPU_CYCLES, BR_MIS_PRED, STALL_SLOT_FRONTEND
retiring	CPU_CYCLES, OP_SPEC, STALL_SLOT, OP_RETIRED

C1.2 Metric Group: Cycle_Accounting

This metric group contains a set of metrics that measure the percentage of processor cycles stalled in either frontend or backend of the processor.

TABLE 27
Neoverse V1 Metrics: Cycle_Accounting, Metric Descriptions

C1.2.1 Metric Descriptions

Metric Name	Metric Title	Metric Description
backend_stalled_cycles	Backend Stalled Cycles	This metric is the percentage of cycles that were stalled due to resource constraints in the backend unit of the processor.
frontend_stalled_cycles	Frontend Stalled Cycles	This metric is the percentage of cycles that were stalled due to resource constraints in the frontend unit of the processor.

TABLE 28
Neoverse V1 Metrics: Cycle_Accounting, Metric Formula

C1.2.2 Metric Formula

Metric Name	Metric Formula	Unit
backend_stalled_cycles	STALL_BACKEND / CPU_CYCLES * 100	percent of cycles
frontend_stalled_cycles	STALL_FRONTEND / CPU_CYCLES * 100	percent of cycles

TABLE 29
Neoverse V1 Metrics: Cycle_Accounting, Metric Events

C1.2.3 Metric Events

Metric Name	Metric Events
backend_stalled_cycles	CPU_CYCLES, STALL_BACKEND
frontend_stalled_cycles	CPU_CYCLES, STALL_FRONTEND

C1.3 Metric Group: General

This metric group contains general CPU metrics for performance analysis.

C1.3.1 Metric Descriptions

TABLE 30
Neoverse V1 Metrics: General, Metric Descriptions

Metric Name	Metric Title	Metric Description
ipc	Instructions Per Cycle	This metric measures the number of instructions retired per cycle.

C1.3.2 Metric Formula

TABLE 31
Neoverse V1 Metrics: General, Metric Formula

Metric Name	Metric Formula	Unit
ipc	INST_RETIRED / CPU_CYCLES	per cycle

C1.3.3 Metric Events

TABLE 32
Neoverse V1 Metrics: General, Metric Events

Metric Name	Metric Events
ipc	CPU_CYCLES, INST_RETIRED

C1.4 Metric Group: MPKI

This metric group contains metrics for different CPU resources that can be measured as misses per kilo instructions.

C1.4.1 Metric Descriptions

TABLE 33
Neoverse V1 Metrics: MPKI, Metric Descriptions

Metric Name	Metric Title	Metric Description
branch_mpki	Branch MPKI	This metric measures the number of branch mispredictions per thousand instructions executed.
dtlb_mpki	DTLB MPKI	This metric measures the number of data TLB Walks per thousand instructions executed.
itlb_mpki	ITLB MPKI	This metric measures the number of instruction TLB Walks per thousand instructions executed.
l1d_cache_mpki	L1D Cache MPKI	This metric measures the number of level 1 data cache accesses missed per thousand instructions executed.
l1d_tlb_mpki	L1 Data TLB MPKI	This metric measures the number of level 1 instruction TLB accesses missed per thousand instructions executed.

Metric Name	Metric Title	Metric Description
l1d_tlb_mpki	L1 Data TLB MPKI	This metric measures the number of level 1 instruction TLB accesses missed per thousand instructions executed.
l1i_cache_mpki	L1I Cache MPKI	This metric measures the number of level 1 instruction cache accesses missed per thousand instructions executed.
l1i_tlb_mpki	L1 Instruction TLB MPKI	This metric measures the number of level 1 instruction TLB accesses missed per thousand instructions executed.
l2_cache_mpki	L2 Cache MPKI	This metric measures the number of level 2 unified cache accesses missed per thousand instructions executed. Note that cache accesses in this cache are either data memory access or instruction fetch as this is a unified cache.
l2_tlb_mpki	L2 Unified TLB MPKI	This metric measures the number of level 2 unified TLB accesses missed per thousand instructions executed.
ll_cache_read_mpki	LL Cache Read MPKI	This metric measures the number of last level cache read accesses missed per thousand instructions executed.

TABLE 34
Neoverse V1 Metrics: MPKI, Metric Formula

C1.4.2 Metric Formula

Metric Name	Metric Formula	Unit
branch_mpki	BR_MIS_PRED_RETIRED / INST_RETIRED * 1000	MPKI
dtlb_mpki	DTLB_WALK / INST_RETIRED * 1000	MPKI
itlb_mpki	ITLB_WALK / INST_RETIRED * 1000	MPKI
l1d_cache_mpki	L1D_CACHE_REFILL / INST_RETIRED * 1000	MPKI
l1d_tlb_mpki	L1D_TLB_REFILL / INST_RETIRED * 1000	MPKI
l1i_cache_mpki	L1I_CACHE_REFILL / INST_RETIRED * 1000	MPKI
l1i_tlb_mpki	L1I_TLB_REFILL / INST_RETIRED * 1000	MPKI
l2_cache_mpki	L2D_CACHE_REFILL / INST_RETIRED * 1000	MPKI
l2_tlb_mpki	L2D_TLB_REFILL / INST_RETIRED * 1000	MPKI
ll_cache_read_mpki	LL_CACHE_MISS_RD / INST_RETIRED * 1000	MPKI

Metric Name	Metric Formula	Unit
branch_mpki	BR_MIS_PRED_RETIRED / INST_RETIRED * 1000	MPKI
dtlb_mpki	DTLB_WALK / INST_RETIRED * 1000	MPKI
itlb_mpki	ITLB_WALK / INST_RETIRED * 1000	MPKI
l1d_cache_mpki	L1D_CACHE_REFILL / INST_RETIRED * 1000	MPKI
l1d_tlb_mpki	L1D_TLB_REFILL / INST_RETIRED * 1000	MPKI
l1i_cache_mpki	L1I_CACHE_REFILL / INST_RETIRED * 1000	MPKI

TABLE 35
Neoverse V1 Metrics: MPKI, Metric Events

C1.4.3 Metric Events

Metric Name	Metric Events
branch_mpki	BR_MIS_PRED_RETIRED, INST_RETIRED
dtlb_mpki	INST_RETIRED, DTLB_WALK
itlb_mpki	INST_RETIRED, ITLB_WALK
l1d_cache_mpki	L1D_CACHE_REFILL, INST_RETIRED
l1d_tlb_mpki	INST_RETIRED, L1D_TLB_REFILL
l1i_cache_mpki	INST_RETIRED, L1I_CACHE_REFILL
l1i_tlb_mpki	INST_RETIRED, L1I_TLB_REFILL
l2_cache_mpki	L2D_CACHE_REFILL, INST_RETIRED
l2_tlb_mpki	INST_RETIRED, L2D_TLB_REFILL
ll_cache_read_mpki	INST_RETIRED, LL_CACHE_MISS_RD

C1.5 Metric Group: Miss_Ratio

This metric group contains metrics to measure miss ratios of different processor resources.

C1.5.1 Metric Descriptions

TABLE 36
Neoverse V1 Metrics: Miss_Ratio, Metric Descriptions

Metric Name	Metric Title	Metric Description
branch_misprediction_ratio	Branch Misprediction Ratio	This metric measures the ratio of branches mispredicted to the total number of branches architecturally executed. This gives an indication of the effectiveness of the branch prediction unit.
dtlb_walk_ratio	DTLB Walk Ratio	This metric measures the ratio of instruction TLB Walks to the total number of data TLB accesses. This gives an indication of the effectiveness of the data TLB accesses.
itlb_walk_ratio	ITLB Walk Ratio	This metric measures the ratio of instruction TLB Walks to the total number of instruction TLB accesses. This gives an indication of the effectiveness of the instruction TLB accesses.
l1d_cache_miss_ratio	L1D Cache Miss Ratio	This metric measures the ratio of level 1 data cache accesses missed to the total number of level 1 data cache accesses. This gives an indication of the effectiveness of the level 1 data cache.
l1d_tlb_miss_ratio	L1 Data TLB Miss Ratio	This metric measures the ratio of level 1 data TLB accesses missed to the total number of level 1 data TLB accesses. This gives an indication of the effectiveness of the level 1 data TLB.

Metric Name	Metric Title	Metric Description
l1i_cache_miss_ratio	L1I Cache Miss Ratio	This metric measures the ratio of level 1 instruction cache accesses missed to the total number of level 1 instruction cache accesses. This gives an indication of the effectiveness of the level 1 instruction cache.
l1i_tlb_miss_ratio	L1 Instruction TLB Miss Ratio	This metric measures the ratio of level 1 instruction TLB accesses missed to the total number of level 1 instruction TLB accesses. This gives an indication of the effectiveness of the level 1 instruction TLB.
l2_cache_miss_ratio	L2 Cache Miss Ratio	This metric measures the ratio of level 2 cache accesses missed to the total number of level 2 cache accesses. This gives an indication of the effectiveness of the level 2 cache, which is a unified cache that stores both data and instruction. Note that cache accesses in this cache are either data memory access or instruction fetch as this is a unified cache.
l2_tlb_miss_ratio	L2 Unified TLB Miss Ratio	This metric measures the ratio of level 2 unified TLB accesses missed to the total number of level 2 unified TLB accesses. This gives an indication of the effectiveness of the level 2 TLB
ll_cache_read_miss_ratio	LL Cache Read Miss Ratio	This metric measures the ratio of last level cache read accesses missed to the total number of last level cache accesses. This gives an indication of the effectiveness of the last level cache for read traffic. Note that cache accesses in this cache are either data memory access or instruction fetch as this is a system level cache.

C1.5.2 Metric Formula

TABLE 37
Neoverse V1 Metrics: Miss_Ratio, Metric Formula

Metric Name	Metric Formula	Unit
branch_misprediction_ratio	BR_MIS_PRED_RETIRED / BR_RETIRED	per branch
dtlb_walk_ratio	DTLB_WALK / L1D_TLB	per TLB access
itlb_walk_ratio	ITLB_WALK / L1I_TLB	per TLB access
l1d_cache_miss_ratio	L1D_CACHE_REFILL / L1D_CACHE	per cache access
l1d_tlb_miss_ratio	L1D_TLB_REFILL / L1D_TLB	per TLB access
l1i_cache_miss_ratio	L1I_CACHE_REFILL / L1I_CACHE	per cache access
l1i_tlb_miss_ratio	L1I_TLB_REFILL / L1I_TLB	per TLB access
l2_cache_miss_ratio	L2D_CACHE_REFILL / L2D_CACHE	per cache access
l2_tlb_miss_ratio	L2D_TLB_REFILL / L2D_TLB	per TLB access
ll_cache_read_miss_ratio	LL_CACHE_MISS_RD / LL_CACHE_RD	per cache access

C1.5.3 Metric Events

TABLE 38
Neoverse V1 Metrics: Miss_Ratio, Metric Events

Metric Name	Metric Events
branch_misprediction_ratio	BR_MIS_PRED_RETIRED, BR_RETIRED
dtlb_walk_ratio	L1D_TLB, DTLB_WALK
itlb_walk_ratio	ITLB_WALK, L1I_TLB
l1d_cache_miss_ratio	L1D_CACHE_REFILL, L1D_CACHE
l1d_tlb_miss_ratio	L1D_TLB, L1D_TLB_REFILL
l1i_cache_miss_ratio	L1I_CACHE, L1I_CACHE_REFILL
l1i_tlb_miss_ratio	L1I_TLB_REFILL, L1I_TLB
l1i_tlb_miss_ratio	L2D_CACHE_REFILL, L2D_CACHE
l2_tlb_miss_ratio	L2D_TLB, L2D_TLB_REFILL
ll_cache_read_miss_ratio	LL_CACHE_MISS_RD, LL_CACHE_RD

C1.6 Metric Group: Branch_Effectiveness

This metric group contains metrics to evaluate the effectiveness of branch instruction execution on this processor.

C1.6.1 Metric Descriptions

TABLE 39
Neoverse V1 Metrics: Branch_Effectiveness, Metric Description

Metric Name	Metric Title	Metric Description
branch_misprediction_ratio	Branch Misprediction Ratio	This metric measures the ratio of branches mispredicted to the total number of branches architecturally executed. This gives an indication of the effectiveness of the branch prediction unit.
branch_mpki	Branch MPKI	This metric measures the number of branch mispredictions per thousand instructions executed.

C1.6.2 Metric Formula

TABLE 40
Neoverse V1 Metrics: Branch_Effectiveness, Metric Formula

Metric Name	Metric Formula	Unit
branch_misprediction_ratio	BR_MIS_PRED_RETIRED / BR_RETIRED	per branch
branch_mpki	BR_MIS_PRED_RETIRED / INST_RETIRED * 1000	MPKI

C1.6.3 Metric Events

TABLE 41
Neoverse V1 Metrics: Branch_Effectiveness, Metric Events

Metric Name	Metric Events
branch_misprediction_ratio	BR_MIS_PRED_RETIRED, BR_RETIRED
branch_mpki	BR_MIS_PRED_RETIRED, INST_RETIRED

C1.7 Metric Group: ITLB_Effectiveness

This metric group contains metrics to evaluate the effectiveness of instruction TLB on this processor.

C1.7.1 Metric Descriptions

TABLE 42
Neoverse V1 Metrics: ITLB_Effectiveness, Metric Descriptions

Metric Name	Metric Title	Metric Description
itlb_mpki	ITLB MPKI	This metric measures the number of instruction TLB Walks per thousand instructions executed.
itlb_walk_ratio	ITLB Walk Ratio	This metric measures the ratio of instruction TLB Walks to the total number of instruction TLB accesses. This gives an indication of the effectiveness of the instruction TLB accesses.
l1i_tlb_miss_ratio	L1 Instruction TLB Miss Ratio	This metric measures the ratio of level 1 instruction TLB accesses missed to the total number of level 1 instruction TLB accesses. This gives an indication of the effectiveness of the level 1 instruction TLB.
l1i_tlb_mpki	L1 Instruction TLB MPKI	This metric measures the number of level 1instruction TLB accesses missed per thousand instructions executed.
l2_tlb_miss_ratio	L2 Unified TLB Miss Ratio	This metric measures the ratio of level 2 unified TLB accesses missed to the total number of level 2 unified TLB accesses. This gives an indication of the effectiveness of the level 2 TLB.
l2_tlb_mpki	L2 Unified TLB MPKI	This metric measures the number of level 2 unified TLB accesses missed per thousand instructions executed.

C1.7.2 Metric Formula

TABLE 43
Neoverse V1 Metrics: ITLB_Effectiveness, Metric Formula

Metric Name	Metric Formula	Unit
itlb_mpki	ITLB_WALK / INST_RETIRED * 1000	MPKI
itlb_walk_ratio	ITLB_WALK / L1I_TLB	per TLB access
l1i_tlb_miss_ratio	L1I_TLB_REFILL / L1I_TLB	per TLB access
l1i_tlb_mpki	L1I_TLB_REFILL / INST_RETIRED * 1000	MPKI
l2_tlb_miss_ratio	L2D_TLB_REFILL / L2D_TLB	per TLB access
l2_tlb_mpki	L2D_TLB_REFILL / INST_RETIRED * 1000	MPKI

TABLE 44
Neoverse V1 Metrics: ITLB_Effectiveness, Metric Events

C1.7.3 Metric Events

Metric Name	Metric Events
itlb_mpki	INST_RETIRED, ITLB_WALK
itlb_walk_ratio	ITLB_WALK, L1I_TLB
l1i_tlb_miss_ratio	L1I_TLB_REFILL, L1I_TLB
l1i_tlb_mpki	INST_RETIRED, L1I_TLB_REFILL
l2_tlb_miss_ratio	L2D_TLB, L2D_TLB_REFILL
l2_tlb_mpki	INST_RETIRED, L2D_TLB_REFILL

C1.8 Metric Group: DTLB_Effectiveness

This metric group contains metrics to evaluate the effectiveness of data TLB on this processor.

TABLE 45
Neoverse V1 Metrics: DTLB_Effectiveness, Metric Descriptions

C1.8.1 Metric Descriptions

Metric Name	Metric Title	Metric Description
dtlb_mpki	DTLB MPKI	This metric measures the number of data TLB Walks per thousand instructions executed.
dtlb_walk_ratio	DTLB Walk Ratio	This metric measures the ratio of instruction TLB Walks to the total number of data TLB accesses. This gives an indication of the effectiveness of the data TLB accesses.
l1d_tlb_miss_ratio	L1 Data TLB Miss Ratio	This metric measures the ratio of level 1 data TLB accesses missed to the total number of level 1 data TLB accesses. This gives an indication of the effectiveness of the level 1 data TLB.
l1d_tlb_mpki	L1 Data TLB MPKI	This metric measures the number of level 1 instruction TLB accesses missed per thousand instructions executed.
l2_tlb_miss_ratio	L2 Unified TLB Miss Ratio	This metric measures the ratio of level 2 unified TLB accesses missed to the total number of level 2 unified TLB accesses. This gives an indication of the effectiveness of the level 2 TLB.
l2_tlb_mpki	L2 Unified TLB MPKI	This metric measures the number of level 2 unified TLB accesses missed per thousand instructions executed.

Metric Name	Metric Title	Metric Description
l1d_tlb_miss_ratio	L1 Data TLB Miss Ratio	This metric measures the ratio of level 1 data TLB accesses missed to the total number of level 1 data TLB accesses. This gives an indication of the effectiveness of the level 1 data TLB.
l1d_tlb_mpki	L1 Data TLB MPKI	This metric measures the number of level 1 instruction TLB accesses missed per thousand instructions executed.

TABLE 46
Neoverse V1 Metrics: DTLB_Effectiveness, Metric Formula

C1.8.2 Metric Formula

Metric Name	Metric Formula	Unit
dtlb_mpki	DTLB_WALK / INST_RETIRED * 1000	MPKI
dtlb_walk_ratio	DTLB_WALK / L1D_TLB	per TLB access
l1d_tlb_miss_ratio	L1D_TLB_REFILL / L1D_TLB	per TLB access
l1d_tlb_mpki	L1D_TLB_REFILL / INST_RETIRED * 1000	MPKI
l2_tlb_miss_ratio	L2D_TLB_REFILL / L2D_TLB	per TLB access
l2_tlb_mpki	L2D_TLB_REFILL / INST_RETIRED * 1000	MPKI

TABLE 47
Neoverse V1 Metrics: DTLB_Effectiveness, Metric Events

C1.8.3 Metric Events

Metric Name	Metric Events
dtlb_mpki	INST_RETIRED, DTLB_WALK
dtlb_walk_ratio	L1D_TLB, DTLB_WALK
l1d_tlb_miss_ratio	L1D_TLB, L1D_TLB_REFILL
l1d_tlb_mpki	INST_RETIRED, L1D_TLB_REFILL
l2_tlb_miss_ratio	L2D_TLB, L2D_TLB_REFILL
l2_tlb_mpki	INST_RETIRED, L2D_TLB_REFILL

C1.9 Metric Group: L1I_Cache_Effectiveness

This metric group contains metrics to evaluate the effectiveness of L1 Instruction cache on this processor.

TABLE 48
Neoverse V1 Metrics: L1I_Cache_Effectiveness, Metric Descriptions

C1.9.1 Metric Descriptions

Metric Name	Metric Title	Metric Description
l1i_cache_miss_ratio	L1I Cache Miss Ratio	This metric measures the ratio of level 1 instruction cache accesses missed to the total number of level 1 instruction cache accesses. This gives an indication of the effectiveness of the level 1 instruction cache.
l1i_cache_mpki	L1I Cache MPKI	This metric measures the number of level 1 instruction cache accesses missed per thousand instructions executed.

TABLE 49
Neoverse V1 Metrics: L1I_Cache_Effectiveness, Metric Formula

C1.9.2 Metric Formula

Metric Name	Metric Formula	Unit
l1i_cache_miss_ratio	L1I_CACHE_REFILL / L1I_CACHE	per cache access
l1i_cache_mpki	L1I_CACHE_REFILL / INST_RETIRED * 1000	MPKI

TABLE 50
Neoverse V1 Metrics: L1I_Cache_Effectiveness, Metric Events

C1.9.3 Metric Events

Metric Name	Metric Events
l1i_cache_miss_ratio	L1I_CACHE, L1I_CACHE_REFILL
l1i_cache_mpki	INST_RETIRED, L1I_CACHE_REFILL

C1.10 Metric Group: L1D_Cache_Effectiveness

This metric group contains metrics to evaluate the effectiveness of L1 Data Cache on this processor.

TABLE 51
Neoverse V1 Metrics: L1D_Cache_Effectiveness, Metric Descriptions

C1.10.1 Metric Descriptions

Metric Name	Metric Title	Metric Description
l1d_cache_miss_ratio	L1D Cache Miss Ratio	This metric measures the ratio of level 1 data cache accesses missed to the total number of level 1 data cache accesses. This gives an indication of the effectiveness of the level 1 data cache.

Metric Name	Metric Title	Metric Description
l1d_cache_mpki	L1D Cache MPKI	This metric measures the number of level 1 data cache accesses missed per thousand instructions executed.

TABLE 52
Neoverse V1 Metrics: L1D_Cache_Effectiveness, Metric Formula

C1.10.2 Metric Formula

Metric Name	Metric Formula	Unit
l1d_cache_miss_ratio	L1D_CACHE_REFILL / L1D_CACHE	per cache access
l1d_cache_mpki	L1D_CACHE_REFILL / INST_RETIRED * 1000	MPKI

TABLE 53
Neoverse V1 Metrics: L1D_Cache_Effectiveness, Metric Events

C1.10.3 Metric Events

Metric Name	Metric Events
l1d_cache_miss_ratio	L1D_CACHE_REFILL, L1D_CACHE
l1d_cache_mpki	L1D_CACHE_REFILL, INST_RETIRED

C1.11 Metric Group: L2_Cache_Effectiveness

This metric group contains metrics to evaluate the effectiveness of L2 Unified Cache on this processor.

TABLE 54
Neoverse V1 Metrics: L2_Cache_Effectiveness, Metric Descriptions

C1.11.1 Metric Descriptions

Metric Name	Metric Title	Metric Description
l2_cache_miss_ratio	L2 Cache Miss Ratio	This metric measures the ratio of level 2 cache accesses missed to the total number of level 2 cache accesses. This gives an indication of the effectiveness of the level 2 cache, which is a unified cache that stores both data and instruction. Note that cache accesses in this cache are either data memory access or instruction fetch as this is a unified cache.
l2_cache_mpki	L2 Cache MPKI	This metric measures the number of level 2 unified cache accesses missed per thousand instructions executed. Note that cache accesses in this cache are either data memory access or instruction fetch as this is a unified cache.

TABLE 55
Neoverse V1 Metrics: L2_Cache_Effectiveness, Metric Formula

C1.11.2 Metric Formula

Metric Name	Metric Formula	Unit
l2_cache_miss_ratio	L2D_CACHE_REFILL / L2D_CACHE	per cache access
l2_cache_mpki	L2D_CACHE_REFILL / INST_RETIRED * 1000	MPKI

TABLE 56
Neoverse V1 Metrics: L2_Cache_Effectiveness, Metric Events

C1.11.3 Metric Events

Metric Name	Metric Events
l2_cache_miss_ratio	L2D_CACHE_REFILL, L2D_CACHE
l2_cache_mpki	L2D_CACHE_REFILL, INST_RETIRED

C1.12 Metric Group: LL_Cache_Effectiveness

This metric group contains metrics to evaluate the effectiveness of Last Level Cache on this processor.

TABLE 57
Neoverse V1 Metrics: LL_Cache_Effectiveness, Metric Descriptions

C1.12.1 Metric Descriptions

Metric Name	Metric Title	Metric Description
ll_cache_read_hit_ratio	LL Cache Read Hit Ratio	This metric measures the ratio of last level cache read accesses hit in the cache to the total number of last level cache accesses. This gives an indication of the effectiveness of the last level cache for read traffic. Note that cache accesses in this cache are either data memory access or instruction fetch as this is a system level cache.
ll_cache_read_miss_ratio	LL Cache Read Miss Ratio	This metric measures the ratio of last level cache read accesses missed to the total number of last level cache accesses. This gives an indication of the effectiveness of the last level cache for read traffic. Note that cache accesses in this cache are either data memory access or instruction fetch as this is a system level cache.
ll_cache_read_mpki	LL Cache Read MPKI	This metric measures the number of last level cache read accesses missed per thousand instructions executed.

TABLE 58
Neoverse V1 Metrics: LL_Cache_Effectiveness, Metric Formula

C1.12.2 Metric Formula

Metric Name	Metric Formula	Unit
ll_cache_read_hit_ratio	(LL_CACHE_RD - LL_CACHE_MISS_RD) / LL_CACHE_RD	per cache access
ll_cache_read_miss_ratio	LL_CACHE_MISS_RD / LL_CACHE_RD	per cache access
ll_cache_read_mpki	LL_CACHE_MISS_RD / INST_RETIRED * 1000	MPKI

TABLE 59
Neoverse V1 Metrics: LL_Cache_Effectiveness, Metric Events

C1.12.3 Metric Events

Metric Name	Metric Events
ll_cache_read_hit_ratio	LL_CACHE_MISS_RD, LL_CACHE_RD
ll_cache_read_miss_ratio	LL_CACHE_MISS_RD, LL_CACHE_RD
ll_cache_read_mpki	INST_RETIRED, LL_CACHE_MISS_RD

C1.13 Metric Group: Operation_Mix

This metric group provides the distribution of micro-operation types executed for the program.

TABLE 60
Neoverse V1 Metrics: Operation_Mix, Metric Descriptions

C1.13.1 Metric Descriptions

Metric Name	Metric Title	Metric Description
branch_percentage	Branch Operations Percentage	This metric measures branch operations as a percentage of operations speculatively executed.
crypto_percentage	Crypto Operations Percentage	This metric measures crypto operations as a percentage of operations speculatively executed.
integer_dp_percentage	Integer Operations Percentage	This metric measures scalar integer operations as a percentage of operations speculatively executed.
load_percentage	Load Operations Percentage	This metric measures load operations as a percentage of operations speculatively executed.
scalar_fp_percentage	Floating Point Operations Percentage	This metric measures scalar floating point operations as a percentage of operations speculatively executed.

Metric Name	Metric Title	Metric Description
simd_percentage	SIMD Operations Percentage	This metric measures SIMD operations as a percentage of total operations speculatively executed.
store_percentage	Store Operations Percentage	This metric measures store operations as a percentage of operations speculatively executed.

C1.13.2 Metric Formula

TABLE 61
Neoverse V1 Metrics: Operation_Mix, Metric Formula

Metric Name	Metric Formula	Unit
branch_percentage	(BR_IMMED_SPEC + BR_INDIRECT_SPEC) / INST_SPEC * 100	percent of operations
crypto_percentage	CRYPTO_SPEC / INST_SPEC * 100	percent of operations
integer_dp_percentage	DP_SPEC / INST_SPEC * 100	percent of operations
load_percentage	LD_SPEC / INST_SPEC * 100	percent of operations
scalar_fp_percentage	VFP_SPEC / INST_SPEC * 100	percent of operations
simd_percentage	ASE_SPEC / INST_SPEC * 100	percent of operations
store_percentage	ST_SPEC / INST_SPEC * 100	percent of operations

C1.13.3 Metric Events

TABLE 62
Neoverse V1 Metrics: Operation_Mix, Metric Events

Metric Name	Metric Events
branch_percentage	INST_SPEC, BR_INDIRECT_SPEC, BR_IMMED_SPEC
crypto_percentage	INST_SPEC, CRYPTO_SPEC
integer_dp_percentage	INST_SPEC, DP_SPEC
load_percentage	INST_SPEC, LD_SPEC
scalar_fp_percentage	INST_SPEC, VFP_SPEC
simd_percentage	INST_SPEC, ASE_SPEC
store_percentage	INST_SPEC, ST_SPEC

© ARM LTD. 2023 All brand names or product names are the property of their respective holders. Neither the whole nor any part of the information contained in, or the product described in, this document may be adapted or reproduced in any material form except with the prior written permission of the copyright holder. The product described in this document is subject to continuous developments and improvements. All particulars of the product and its use contained in this document are given in good faith. All warranties implied or expressed, including but not limited to implied warranties of satisfactory quality or fitness for purpose are excluded. This document is intended only to provide information to the reader about the product. To the extent permitted by local laws Arm shall not be liable for any loss or damage arising from the use of any information in this document or any error or omission in such information.