Icicle: Open-Source Hardware Support for Top-Down Microarchitectural Analysis on RISC-V

Matthew Edwin Weingarten, Michael Grieco, Stephen A. Edwards, Tanvir Ahmed Khan
Columbia University, New York, USA
matthew.weingarten@columbia.edu, michael.grieco@columbia.edu, sedwards@cs.columbia.edu, tk3070@columbia.edu

***Abstract*—Performance characterization enables software to efficiently utilize the underlying hardware by pinpointing key performance bottlenecks. The need for specialization and hardware/software co-design continues to drive up the pace of hardware development, especially noticeable in open-source platforms. Unfortunately, performance characterization on these platforms remains challenging, as RISC-V processors do not support the industry standard characterization methodology, Top-Down Microarchitectural Analysis (TMA). This lack of support inhibits practitioners who rely on open-source tooling to understand workload bottlenecks and researchers proposing novel characterization methods.**

In this paper, we introduce Icicle,[1] the first full system-stack TMA implementation on widely-used open-source processors, Rocket and BOOM. Icicle enables TMA by adding three and seven new performance events to Rocket and BOOM respectively, while also revising the physical implementation of performance counters to support monitoring concurrent events. Icicle also includes a `perf`-like software tool and a trace-based validation infrastructure. We evaluate Icicle's efficacy with three case studies, accuracy against trace-based ground truth, and overhead of adding new events and counter architecture, measured in terms of post-synthesis power and timing analysis.

***Index Terms*—Computer architecture, Computer performance, Performance evaluation, Microarchitecture, RISC-V instruction set architecture**

I. Introduction

Characterization is essential in the post-Moore era to identify and remedy performance bottlenecks [37], [68], [98]. Accurate pinpointing of these bottlenecks enables optimizations such as manual tuning of workloads [36], [62], [104], [116], automated compiler techniques including Profile-Guided Optimizations [32], [77], [78], [89], or modifications to the hardware itself [57], [60], [61], [63], [64], [66], [91]. However, characterizing modern hardware is challenging due to the growing design complexity [50], [81]–[83]. Successfully locating performance bottlenecks often requires a detailed understanding of the underlying hardware [28], [51], [53].

To enable finding performance bottlenecks without knowing hardware details, hardware vendors [5], [55], [75] and software profilers [7], [38] adopt the industry-standard characterization methodology, Top-Down Microarchitectural Analysis (TMA) [107]. TMA feeds values of hardware performance events [40], [84] to predefined models [6] and classifies pipeline slots into high-level categories. Drilling down these categories further to pinpoint specific sources of pipeline stalls (*e.g.*, cache misses or execution unit contentions), TMA helps users and automated tools find effective optimizations [16], [23], [24], [69], [85], [92], [93], [95], [111], [112], [115]. Alas, while TMA is readily available on Intel [5]–[7], AMD [55], and ARM [75] processors, open-source RISC-V processors provide limited TMA support [15], [74], [96].

RISC-V has a robust and growing toolchain ecosystem [2]. Platforms such as Chipyard [18] enable rapid prototyping of processors and system on chips (SoCs), facilitating research into microarchitectural improvements [51], [53], custom processor designs [22], [114], and hardware accelerators [57], [60], [71], [72], [79], [86]. However, support for performance characterization remains limited. *Currently, conducting a TMA performance characterization is infeasible on open-source RISC-V cores due to insufficient hardware performance events.* Even for an in-order core like Rocket [22], existing performance events cannot pinpoint stalls, as we show in §III, resulting in an incomplete analysis. This challenge intensifies for superscalar and out-of-order designs such as BOOM [114].

The insufficiency of Performance Monitoring Units (PMUs) impedes both research and development efforts across open-source RISC-V platforms. Since industry drove most prior work in this field [3], [6], [74], [75], [107], the open-source RISC-V environment presents a unique opportunity to develop PMU architectures and performance characterization tools. Many researchers rely on this ecosystem to experiment with microarchitectural improvements [19], [46], [97], [113], hardware/software co-design [19], [27], [41], [88], [94], [108], or design space exploration [25], [56], [110]. These research efforts would benefit from reliable characterization as it provides insights during the design process [106], while also increasing confidence in the evaluation of workloads [111]. Additionally, despite the trend towards offloading an ever-increasing fraction of workloads to accelerators [49], CPU performance remains critical as the sequential code running on the CPU remains the dominating factor in runtime [21], [53]. Finally, monitoring hardware on heterogeneous systems remains an open research area [29], [54], [67], which is much harder to pursue without solid foundations of CPU characterization methods [83], [92], [117].

Missing performance events to capture and detect pathologies in the pipeline at run time is the main contributor to the insufficiency of characterization on open-source platforms. As we demonstrate a motivating example in §III, the insufficiency

[1]We open source all of our work at https://github.com/ice-rlab/Icicle

results in an incomplete understanding of how well software is utilizing the underlying hardware, harming the ability to build characterization tools. However, choosing *which* metrics to track on modern hardware is challenging as bottom-up events (*e.g.*, cache-miss events [1], [4], [8], [39]) provide an inaccurate performance picture on modern hardware [107].

Monitoring pipeline events for performance issues also requires careful consideration, as stringent requirements on physical design overhead drive the number of PMU counter registers down [70], [73]. Proprietary processors amortize the cost of monitoring by employing sampling and approximation techniques, ignoring non-determinism [26], [100], [101], [109]. Unfortunately, existing literature fails to quantify the physical overhead of performance events and counters in terms of post-logic synthesis metrics for power, area, and timing.

Finally, performance events and counter values provide only a snapshot of the processor pipeline across many cycles. Consequently, exhaustively identifying the root causes of pipeline stalls requires performance events that are too invasive [50], incurring an unreasonable overhead. Similarly, root causes of stalls also overlap in time in a way that is not always possible to capture with performance events, yet there is no way to quantify the inaccuracy [52].

Contributions. In this paper, we develop Icicle to address these issues, and make the following contributions:

- **Top-down microarchitectural analysis support** for both Rocket [22] and BOOM [114], by introducing 3 and 7 new performance events in Rocket and BOOM respectively, and an accompanying software toolchain to read counters and compute TMA metrics (§IV-A).
- **Superscalar counter implementations** with low physical overheads (post-placement increase of 4.15% in power, 1.54% in area, 9.93% in wirelength, at 200MHz) that allow tracking of multiple performance events per cycle, a key requirement for characterization on wide-issue designs (§IV-B).
- **Trace-based validation** to help design and evaluate the accuracy of our TMA implementation (§IV-C).
- **Comprehensive evaluation** of our TMA implementation on SPEC CPU2017 [31] and CoreMark [47] (§V-A), multiple case studies, trace-based validation (§V-B), and a detailed efficiency analysis of the new performance events and counters' physical implementations in terms of power, area, and timing (§V-C).

II. Background

To understand the motivating example in §III, we first cover the PMU events and counters architecture (§II-A), followed by the TMA methodology (§II-B), and finally an overview of the Rocket and BOOM microarchitectures as they relate to performance monitoring (§II-C).

A. PMU Counters Architecture

Chipyard has a single performance event interface shipped with Rocket and reused across all hardware. The principal component of the Performance Monitoring Unit (PMU) or Hardware Performance Monitor (HPM)[2] is a *counter* — the piece of hardware that increments and retains the value of the number of times the counter is triggered.

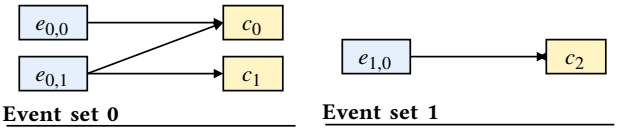


Fig. 1: Counter C_k and event $e_{i,j}$, where $e_{i,j}$ is event j in event set i.

While BOOM and Rocket use the same interface, they do not track the same events. We call the signal that triggers this counter an *event*, such as a cache miss or an instruction retirement signal. Events are grouped into *event sets*. Each event can be mapped to one or many counters as long as every event mapped to the same counter belongs to the same event set. Fig. 1 shows that c_0 increments whenever events $e_{0,0}$ or $e_{0,1}$ signal high. Events can be mapped to multiple counters; $e_{0,1}$ maps to both c_0 and c_1. However, $e_{0,1}$ could not map to c_2 as $e_{1,0}$ does not belong to the same event set [1], [2], [8]. *Importantly, if two events are mapped to the same counter signal high in the same cycle, the counter only increments by one.*

B. Top-Down Microarchitectural Analysis

TMA provides a methodology to interpret values collected from PMUs to pinpoint bottlenecks. Slots are the primary unit of abstraction. Intuitively, the number of slots represents the "work" required to process a specific workload. Each slot corresponds to one cycle spent at each stage of each lane in the pipeline. On a high level, TMA boils down to classifying every slot into a collection of hierarchical classes.

Drilling down into the class hierarchy helps to pinpoint the root cause of stalls. The top-level classes on any CPU typically include *Backend Bound*, *Frontend Bound*, *Bad Speculation*, and *Retiring*. The Retiring category constitutes the only category that represents useful work and is typically calculated by the number of micro-ops[3] (μ-ops) retired. A workload that is Frontend and I-cache Bound may see the highest impact from code size reductions or prefetching [24], [63], [77], [89], [111], a Bad Speculation Bound workload might benefit from improvements to branch prediction [65], [90], while Backend Bound workloads can be improved with strength reduction [34], instruction scheduling [48], or data prefetching [99] based on being Core or Memory Bound.

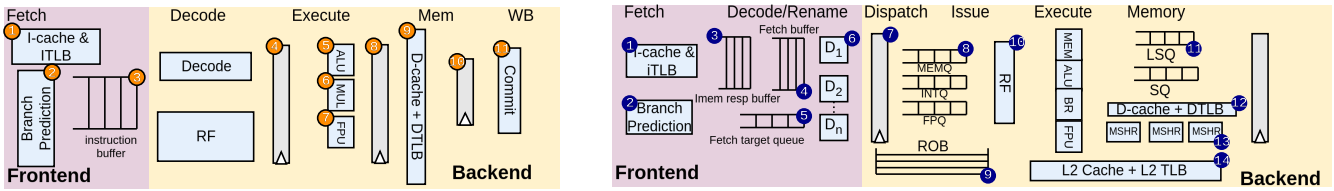
TMA outperforms traditional "bottom-up" approaches, that assign a static cost to each event. Bottom-up approaches fail to account for latency-hiding techniques in modern processors — for example, not every cache miss results in the same number of stalled cycles.

C. Performance Events in Rocket and BOOM

Performance characterization in TMA fashion entails identifying stalls and attributing each stall to a root cause.

[2]From now on we will refer to the counter architecture as PMUs.

[3]We use the term μ-ops exclusively without loss of generality, even though Rocket does not use μ-ops.



(a) Simplifed Rocket Core pipeline

(b) Simplified BOOM Core pipeline

Fig. 2: Core pipelines with marked performance event sources.

Characterization tools collect or sample strategically chosen performance events by interacting with PMU hardware. Fig. 2 shows a simplified diagram of the 5-stage in-order Rocket pipeline and 10-stage superscalar OoO BOOM pipeline. Tab. I lists all performance events. There are three existing event sets: Basic, Microarchitectural, and Memory. The TMA category contains the events added in this work. A stall may occur in the issue/execute stages due to structural- and data-hazards (4 or 8), a branch misprediction (2 or 2) that may cause an already issued μ-op to be flushed, or a miss in the instruction cache (I-cache) (1 or 1) to name a few examples. Dedicated events, like *I$-blocked* event, intend to capture the number of cycles lost due to a miss in the I-cache. Notice that the BOOM pipeline does not provide a comparable event to diagnose I-cache stalls, as it is challenging to implement such an event in an OoO pipeline.

III. Motivation: Why is event choice critical?

The primary motivation for our work is that an incorrect or incomplete set of performance events leads to poor performance characterization and ultimately inaccurate conclusions. To illustrate, we consider a scenario in which we aim to precisely identify how many cycles are lost in the Frontend of Rocket for a small bare-metal mergesort workload. Notice that the only existing performance events on Rocket related to the Frontend are `I$-miss` and `I$-blocked` 1. To diagnose Frontend pathologies on mergesort, an intuitive first approach might consider defining a model that uses these two events to gain insight into the number of Frontend stalls. Counting these two events is the only feasible approach to measure Frontend stalls. Alas, this approach is insufficient as Frontend stalls also originate from sources other than `I$-miss` and `I$-blocked`!

TABLE I: List of Rocket (above) and BOOM PMU (below) events. n/n = event origin, * = new event, † = Top-level TMA, ‡ = lower-level TMA.

Basic	Microarchitect.	Memory	TMA Events*
Cycles	Load-Use-inter.	I$ miss 1	**Instr. issued*** 4
Instr.R.	Long-latency inter.	D$ miss 9	**Fetch bubbles*** 3
Load	Csr-inter.	D$-release	**Recovering*** 2 3
Store	I$-blocked 1	ITLB-miss 1	
Atomic	D$-blocked 9	DTLB-miss 10	
System	Br-mispred.‡ 2 7	L2-TLB-miss	
Arith	CF-targ.mis.† 2		
Branch	Flush† 11		
Fence†	Replay 11		
...	Mul/Div-interlock 6		
	FP inter. 7		
Cycles	Br-mispred.†	I$-miss 1	**Uops-issued*†** 8
Instr. R.	CF-target-mispred.	D$-miss 12	**Fetch-bubbles*†** 2 4 6
Exception	Flush†	D$-release 12	**Recovering*†** 2 9
	Branch resolved	I-TLB$ miss 1	**Uops-retired*†** 9
		D-TLB$ miss 12	**I$-blocked*‡** 1 4
		L2-TLB$ miss 14	**D$-blocked*‡** 8 13

To demonstrate this insufficiency, we traced (§IV-C) a mergesort microbenchmark in simulation to collect the cycle-accurate state of 6 performance-critical Frontend events, visualized in Fig. 3. Focus on subfigure (a), where the trace is zoomed into a single I-cache miss event, highlighting the `I$-blocked` and `I$-miss` events. Each dot represents a signal being high at a specific clock cycle. Early in the runtime, instructions are loaded into the processor, triggering an `I$-miss` event. The miss is followed by around 40 cycles of the `I$-blocked` event.

Now, zooming out again, we add events to the trace, not part of the performance events supported by Rocket. The decode stage and instruction buffer operate a ready-valid handshake [80] at 3. We add both of these signals to the trace as `IBuf-valid` and `IBuf-ready`. We can determine if there is a stall that is caused by the Frontend if the decode stage is ready for a new μ-op , but the instruction buffer does not contain any valid instructions. One additional caveat is that the Frontend must not be `Recovering` from a branch misprediction 2. We define:

$$\texttt{FetchBubble} = \neg\texttt{Recovering} \wedge \left(\neg\texttt{IBuf-valid} \wedge \texttt{IBuf-ready}\right)$$

Focusing on the early parts of the trace below Fig. 3 (a), the `I$-blocked` appears to accurately track the `FetchBubble` signal. However, if we zoom into Fig. 3 (b), the I-cache is no longer cold and there are no `I$-miss` signals in sight, yet the Frontend of the pipeline is unable to supply instructions at every cycle, although the decode stage is ready. These fetch bubbles are also not due to branch misses or resteering, as the core is not in a recovery state. Most importantly, this

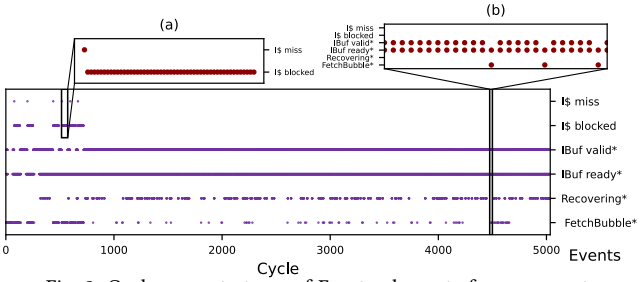


Fig. 3: Cycle-accurate trace of Frontend events for mergesort.

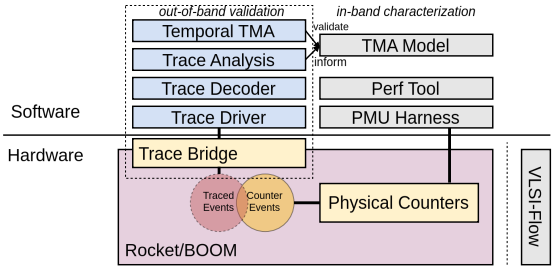


Fig. 4: Icicle overview.

phenomenon *is not an isolated incident*. Indeed, most Frontend stalls are not related to the I-cache for this workload, as its instruction footprint comfortably fits in L1 I-cache.

To summarize, *existing performance events of Rocket fail to detect the Frontend stall.* This makes it extremely challenging to properly characterize the performance of any workload on Rocket without understanding the implementation details of the pipeline. The same argument holds for BOOM, as BOOM has both a more complex microarchitecture, while simultaneously providing fewer performance events.

IV. Icicle's Implementation

We present Icicle, the first implementation of TMA on the Chipyard ecosystem to enable researchers and practitioners to seamlessly characterize workloads on open-source designs. This comprises the full system-stack, from the addition of new performance events to the processor pipelines, the PMU counters architecture, all the way to the software stack that includes a PMU harness and `perf`-tool to read counter values and a performance and characterization tool to apply the TMA model.

In particular, Icicle includes two types of components: (1) *in-band* and (2) *out-of-band* components. *In-band* components are all components native to the processor. This includes the performance events (§IV-A), counters implementation (§IV-B), and collecting and interpreting performance data inside the processor core (§IV-D). The in-band components target users that wish to characterize workloads using TMA both in and out of simulation. *Out-of-band* components help design and validate functionality and evaluate area and timing overhead. In particular, Icicle enables functional validation with a Chipyard extension to collect execution traces (§IV-C), while estimating area and timing overhead by integrating with various VLSI flows (§V-C). The out-of-band components target users that wish to evaluate characterization hardware itself. We show how all components of Icicle work together in Fig. 4.

A. Adding Events for TMA

The primary limitation preventing effective TMA on Rocket and BOOM cores is the lack of sufficient performance events. To support TMA, we introduce three new events to Rocket and seven events to BOOM (Tab. I). Fig. 5 visualizes first- and second-level TMA classes. The formulas of TMA used for each class is presented in Tab. II.

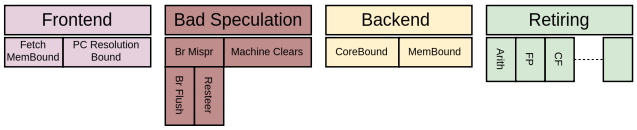


Fig. 5: TMA classes for BOOM and Rocket.

In this section, we detail the rationale for each event added to the BOOM pipeline. For each added event, we ensure it satisfies two Design Principles (DP):

DP 1: *New events should capture a performance pathology that is not observable by any of the existing events.*

An added event must be indispensable to categorize a slot into Top-down category. Each added event comes at cost in physical design that we aim to minimize. We discuss some further approximation possibilities in §V-A.

DP 2: *New events should be minimally invasive, requiring little to no additional state or complex logic to implement.*

We define *minimally-invasive* events as single-cycle events that do not require complex computing logic or finite-state machines with more than two states. Furthermore, the overhead should only scale with the pipeline width. For example, while it might be useful to track the runtime occupancy of the Fetch Buffer (4), doing so would introduce significant hardware overhead. Instead, we opt for a lightweight approximation using existing control signals, such as valid and ready.

1) Top-level Events: We begin with top-level categorization organized by category. The *Retiring* category is the simplest to capture and only requires counting retired μ-ops. Since BOOM provides only an instruction-retired counter by default, we add a `Uops-retired` event using the ROB commit signals (9). No events are added for the top-level *Backend* category, as

TABLE II: TMA model for BOOM with new performance events.

Derived Metrics	
M_{total}	$C_{\text{cycle}} \times W_C$
M_{tf} (All flushes)	$C_{\text{flush}} + C_{\text{bm}} + C^*_{\text{fence}}$
$M_{\text{br\_mr}}$ (Br miss rate)	$C_{\text{bm}}/M_{\text{tf}}$
$M_{\text{nf\_r}}$ (Non-fence flush)	$(C_{\text{bm}} + C^*_{\text{fence}})/M_{\text{tf}}$
$M_{\text{fl\_r}}$ (Machine-flush)	$C_{\text{flush}}/M_{\text{tf}}$
M_{rl} (Recover-length)	4 (Cycles from Decode stage (6) to being issued (8))
Top-level TMA	
Retiring	$C_{\text{ret}}/M_{\text{total}}$
BadSpec	$\frac{(C^*_{\text{issued}} - C_{\text{ret}})\, M_{\text{nf\_r}} \;+\; (C_{\text{rec}} + M_{\text{rl}}\, C^*_{\text{bm}})\, W_C}{M_{\text{total}}}$
Frontend	$C^*_{\text{fetch}}/M_{\text{total}}$
Backend	1 – Frontend – BadSpec – Retiring
Lower-level TMA	
MachCl.	$(C^*_{\text{issued}} - C_{\text{ret}})\, M_{\text{fl\_r}}/M_{\text{total}}$
BrMispr.	$((C^*_{\text{issued}} - C_{\text{ret}})\, M_{\text{br\_mr}} + C^*_{\text{rec}})/M_{\text{total}}$
Resteer.	$(C^*_{\text{issued}} - C_{\text{ret}})\, M_{\text{br\_mr}}/M_{\text{total}}$
RecovBub	$C^*_{\text{rec}}/M_{\text{total}}$
FetchLat.	$C^*_{\text{iblk}}\, W_C/M_{\text{total}}$
PCRes.	Frontend – FetchLat
CoreBound	Backend – $C^*_{\text{db}}/M_{\text{total}}$
MemBound	$C^*_{\text{db}}/M_{\text{total}}$

this is the most complicated to diagnose and can be inferred based on the values of the other three classes.

Frontend. To capture Frontend-bound stalls, we implement `Fetch-bubbles`. These occur when the Frontend fails to supply enough μ-ops for the Backend to operate at full capacity. The `Fetch-bubble` event is equivalent to the one introduced in the motivating example in Fig. 3.

The Fetch Buffer supplies raw instruction data to the decoders and can deliver up to W_C instructions per cycle, where W_C is the core width. We model `Fetch-bubbles` as W_C per-lane events, where event i is asserted if the fetch packet is valid, but decoder lane D_i (6) does not successfully handshake. The decode stage itself never acts as a stall source, but can put backpressure on the Frontend.

A challenge arises in avoiding misclassification during pipeline flushes. For example, a mispredicted branch may trigger a Frontend flush, resulting in fetch-bubbles that are not attributable to the Frontend itself but rather to Bad Speculation. To prevent this, fetch bubble events are suppressed when the pipeline is in a recovery state. This is tightly connected to the *Bad Speculation* category, which we discuss subsequently.

Bad Speculation. To account for lost slots due to incorrect speculation, we implement two new events: `Uops-issued` and `Recovering`. The *Bad Speculation* category encompasses both slots lost to pipeline flushes caused by incorrect speculation along with the subsequent recovery phase. Common sources of Bad Speculation include branch mispredictions and machine clears, a "catch-all" for events such as memory disambiguation failures [87], [107]. Conceptually, the number of lost slots due to flushes must be computed by taking a start point in the pipeline, in our case `Uops-issued` (8), and an end-point, in our case `Uops-retiring` (9) – the difference in these gives us the flush count. The flush count is unreliable if buffers sit between start and end-point. This also informs the choice to take `Uops-issued` instead of `Uops-dispatched` (7); issue-queues (8) may overinflate the number of lost slots due to Bad Speculation. BOOM's issue queues follow a valid only protocol to their respective execution units, each of which may contain multiple functional units. All stalling logic is internal to the issue queue itself. BOOM typically includes one issue queue for integer operations, one for memory, and one for floating-point. The total issue width W_I often exceeds the commit width W_C; we use W_I valid signals to implement this event.

While `Uops-issued` tracks the Backend of pipeline, flushes will also affect the Frontend to recover the PC from a misspeculation. The `Recovering` event captures this phenomenon. Furthermore, it is relevant to the Frontend, allowing us to distinguish between fetch bubbles and recovery bubbles. This event works as follows: This counter begins incrementing at a flush event ((2), (9), (11)) and continues incrementing until a fetch packet is valid (4).

It is important to note that if a branch target misprediction results in an I-cache miss of the new PC, the recovery implementation attribute the lost slots to the Bad Speculation category. To distinguish the two, one would have to decide whether or not the miss would have happened if the target was predicted correctly, which could depend on the prefetcher implementation and we deem this approach to break **DP 2**.

Finally, strictly speaking, we want to avoid considering slots lost by intended pipeline flushes by fence instructions, which should not be considered performance pathology. Hence, we add a `Fence-retired` instruction counter, since BOOM does not have counters for each instruction type.

2) Low-level Events: In this section we discuss the events necessary for second- and third-level TMA classes.

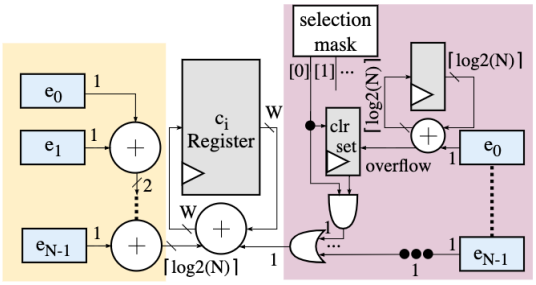
Low-level Frontend. To diagnose Frontend stalls, we introduce the `I$-blocked` event. This event quantifies slots lost specifically to I-cache misses, as opposed to stalls caused by unresolved program counters (e.g., during indirect calls) or any other Frontend issue. The main challenge in capturing this event stems from the presence of multiple pipeline stages and two buffers between the I-cache and decode: the I-mem response Buffer (3) and the Fetch Buffer (4). The Fetch Buffer typically holds two cycles of instruction data, while the I-mem response buffer holds a single fetch width. Due to this buffering, the visibility of an I-cache miss can vary depending on the PC timing and buffer occupancy, making stall attribution non-trivial. Furthermore, a prefetcher could make an I-cache request well before the instruction is used. We therefore adopt a simple heuristic which incurs a low cost: the `I$-blocked` event is asserted whenever a refill is in progress *and* the Fetch Buffer is empty.

Low-level Bad speculation. To refine the *Bad Speculation* category, we distinguish between lost slots from branch mispredictions and machine clears. Branch mispredictions are prioritized, as they are a frequent optimization target. We leverage existing events (`Flush` and `Br-mispredict`) and assume each flush type results in a fixed number of lost slots. More precise tracking would require attributing each flushed μ-op to its cause, which would violate **DP 2**.

At a third level, we use `Uops-issued` and `Recovering` to separate flushed μ-ops from Frontend recovery stalls. We conservatively assume that branch mispredictions ((2)) cause full pipeline flushes, while Backend-originating flushes ((9)) are limited to the Backend. Our model therefore assumes every recovery bubble is incurred only by a branch mispredict; thus overestimating its impact.

Low-level Backend. The low-level *Backend* categorization distinguishes between Core Bound stalls, caused by structural or data hazards, and Memory Bound stalls, where μ-ops are stuck in the issue queues waiting on cache misses. This distinction is notoriously difficult to make in out-of-order processors, where speculative execution is designed specifically to hide cache-miss latency. However, even an approximate classification between these two categories provides significant insight into workload behavior.

To address this, we introduce the `D$-blocked` event to capture stalls attributable to the memory system. Implementing this event is challenging due to the nature of modern issue queues. These queues rely on a wake up mechanism [76],



(a) Add wires. (b) Distributed counters.

Fig. 6: Counter implementations.

where a μ-op remains in the queue until all its source operands are ready, at which point it is "woken up." Wake-up signals may originate from either the bypass network or the register file [14]. The key difficulty is that the wake-up signals abstract away the cause of the stall, or the source of the wake-up signal. Whether the μ-op is stalled on a long-latency cache miss or waiting for another ALU μ-op due to a data dependency, the wake-up mechanism is the same. Disentangling these cases would require intrusive logic into the register file or bypass network, once more violating our minimal-invasiveness design principle (**DP 2**). Additionally, BOOM includes multiple types of issue unit implementations [14], which would potentially require specialized implementations for each.

To overcome this challenge we propose a simple heuristic. In most cases if an the issue queues contain valid μ-ops, their operands are not ready, and there is a D-cache refill in-flight, it is very likely that μ-ops are stuck waiting for the memory response. Matching this logic, for each commit width slot W_C, we define a corresponding `D$-blocked` event that is high if: (1) the issue queue failed to produce a valid instruction, (2) the queue was not empty, and (3) at least one MSHR is currently handling a cache miss. This heuristic may occasionally misattribute a stall to the Mem Bound category that is stalling for another reason. For example, if the stall is actually due to a long-latency functional unit like a pipelined multiplier, and an unrelated MSHR is active.

Rocket Events. Most of the proposed event categories apply directly to Rocket, though implementation is considerably simpler due to its in-order, non-speculative design. Notably, Rocket already includes I$-blocked and D$-blocked counters.

Limitations. This work focuses on top-level and second-level TMA categories and does not yet consider the impact of TLB behavior, we leave for future work. Similarly, deeper levels of the TMA hierarchy, while potentially insightful, require more dedicated analysis implementation.

B. Counters Architecture

There is a lack of hardware support for efficiently monitoring concurrent events, which has contributed to an overall lack of robust performance monitoring infrastructure in Chipyard. A naïve approach monitors each concurrent event with a separate counter. The advantage is that designers can easily map new events to counters (Fig. 1). In our fetch bubbles example at the top level of TMA (Fig. 2), each lane's valid signal would map to a separate counter. We denote this approach as the scalar counters implementation. However, this not always feasible as it exerts pressure on the already limited number of counters [70], [73]. Hence, we design approaches to combine information into the same counter mapping to reduce the pressure on a limited set counters.

Additionally, there are implications in the physical design process as tools must route wires from an event source to a distant counter. This challenge is not immediately apparent to an RTL designer. As we will see, the place and route tools tend to place the counters in the middle of the die. These tools find this to be optimal because the counters monitor pieces of the entire design. The placement in the middle finds the cheapest aggregate routing from event to counter. With TMA, designs need many more events. Routing these event signals on new wires to the counters can introduce a new critical path, which would be a non-critical function limiting performance. We address these challenges by implementing new increment logic in the RTL design that reduces pressure on the place and route process by simplifying the path from event to the counter. Namely, we design two novel approaches with awareness of wire length and combinational delay as shown in Fig. 6.

Add wires. The first, more straightforward option is to aggregate the number of sources that experienced the same event in a cycle with a multi-bit increment signal routed to a single counter. Fig. 6a depicts the incremental aggregation, in which the adders are placed locally before the main counter, thus reducing the amount of wires going between events and the counter registers. However, the partial sums must reach the next adder over longer wires. Additionally, as the number of event sources increases, there are more combinational adders in between each event and the counter register. Those two factors may sum to create a new critical path that compromises the entire design. Implementing an adder network lacks modularity as the adder architecture highly depends on the number of events and sources. While adder trees would be more optimal, we prioritized simplifying the Chisel-based implementation in Chipyard that compiled into a sequential chain to aggregate events. Regardless of the adder network optimality, the multi-bit increment signals lead to complications when using event sets such as in BOOM. This is because the counters can multiplex the sources as discussed in §II-A. In the case where the increment signals have different widths (i.e. if the events have a different number of sources), the logic must pad the smaller increment signal with extra bits to match the larger increment width.

Distributed counters. Our final strategy attempts to completely remove the PMU logic from a potential critical path. Fig. 6b depicts our implementation of local counters that count local instances of an event near each source. Each local counter sets a register bit once it overflows. The global counter (as read by software) arbitrates the set of these registers using a rotating one-hot mask to select one overflow signal in each cycle. If that selected signal is high, the principal counter increments by one. Each local counter's overflow register

also clears when it sees its select signal, like a clear-on-read register, to prevent double counting. The advantage is that all counters can still use a one-bit increment signal. This, along with the locality of smaller counters contributes to the improved modularity of this design because the source of the event only needs to route signals to a nearby counter register, thus taking the increment logic off a potential critical path as in the adders approach.

In this approach, we count concurrent events by incrementing based on the counter overflow bit, which represents the event that has occurred by 2^N times, with N as the width of the counter. Since only one counter passes its overflow per cycle, each counter waits for a maximum number of cycles corresponding to the number of event sources. One drawback is that at the end of execution if the counter has not overflowed, the final count will not reflect the leftover events in each local counter. The principal counter undercounts at most by the product of the number of event sources multiplied and the maximum value of the counter. When considering BOOM with a fetch width of four, there are four corresponding fetch bubble event sources. Each slot implements a local counter that waits for four cycles between selections, so each counter must be able to count up to three before signaling overflow. At the end of execution, the worst case is that each counter holds the value of four, leading to a total undercount of twelve. This number is negligible when running longer benchmarks. In our smallest benchmark, the fetch bubbles count was 929, hence the worst case error would be $\frac{12}{929+12} = 1.28\%$.

C. Microarchitectural Event Trace

In this section, we describe Icicle's custom extension to FirePerf that enables collecting traces of custom microarchitectural events at the finest granularity (*i.e.*, every cycle). As we show in §III, fine-grained traces of microarchitectural events also help identify critical events to detect performance issues such as Frontend stalls. FireSim supports out-of-band instruction tracing with TracerRV [58] using a Target-to-Host bridge [13]. In particular, FireSim streams TracerRV data over PCIe from the target FPGA simulating the design to the host processor. Unfortunately, TracerRV produces a large amount of data, slowing the simulation speed. For example, tracing a single SPEC CPU2017 benchmark with TracerRV produces hundreds of terabytes of data [52]. Other alternatives (*e.g.*, waveform debugging) are not feasible as FireSim does not support them in FPGA-accelerated simulation.

To solve this, we customize the TraceRV implementation to optionally trace performance events and send dynamic signals over the bridge and PCIe to the host machine for every simulated cycle (Fig. 4), as opposed to instruction data. The extension includes a custom DMA driver to read trace data, interpreted as raw binary data. Each event must be chosen manually in the BOOM core. A trace analyzer is needed to parse and interpret the trace, and contains a matching type definition for each bit in the trace to the TraceBundle. The trace analyzer can apply the TMA model on raw trace data.

TABLE III: Simulation & Benchmark Configuration

Part	Configuration
Simulator	Firesim@141bff7 on Xilinx VCU118
Compiler	riscv64-gcc 13.2.0 (-O2)
Benchmarks	Coremark [47]; Dhrystone [103]; riscv-tests [11]; SPEC CPU2017 [31] -O3, Intrate, Threads=1, BOOM:ref, Rocket:test
OS / FW	Buildroot + Linux 6.6.0; OpenSBI v1.2

We refer to trace-based TMA as temporal TMA, which can look for performance event windows.

D. Perf Software Harness

Our software harness supports both baremetal and Linux simulations. We first describe how our harness supports baremetal simulations. Then, we explain the extensions to support Linux simulations.

The harness configures different counters to track a group of performance events using Control and Status Register (CSR) instructions. In particular, our harness sets up the counters in four steps: (1) enabling the CSR registers, (2) writing an 8-bit event ID into the control register of each counter (3) setting a 56 event-bit mask to choose the tracked events in the event set, and (4) clearing the inhibit bit to let the counters begin incrementing.

As all four steps require M-mode, workloads on Linux require (1)-(4) to be done in the openSBI bootloader [9]. Every workload and hardware configuration has varying performance event requirements, especially for performance characterization research, making it cumbersome to manually update openSBI for each configuration. To solve this, we provide a wrapper around FireMarshal build commands with optional parameters of preset CSR booting instructions depending on the event requirements into openSBI and triggering a rebuild. *Only one command is necessary to experiment with new event and counter setups show TMA data from simulated benchmarks.*

V. Evaluation

Icicle implements TMA for BOOM and Rocket. We evaluate it on SPEC CPU2017, Coremark, Dhrystone, and a collection of microbenchmarks. We present three Case Studies (CS) that demonstrate TMA's sensitivity to software, compiler, and architectural changes. We also show an example of trace-based validation using a temporal TMA model to bound inaccuracies of TMA our implementation. Finally, we report area, timing, power, and wire-length overheads for the added events and counter architecture.

Simulation methodology. All simulations use cycle-accurate FireSim [58]. We compile all benchmarks with `gcc -O2` unless noted as `-O3` does not provide any substantial benefit over `-O2` [35]. We list different simulation parameters in Tab. III and parameters for different processor cores in Tab. IV. We show TMA only for LargeBOOMV3 and Rocket for brevity.

A. Top–Down Analysis Results

Fig. 7 shows all TMA results for Rocket (subfigures a-f). In particular, subfigure (a) shows Rocket's top–level TMA breakdown, and (b) zooms into the backend.

TABLE IV: Core & Simulation Configuration

Common	RV64IMAFDCZICSR, 31 Perf Counters, 3.2 GHz, FASED@1 GHz, No LLC, 32 KiB, 8-way, 64 B block L1D/I, 512 KiB, 8-way, 64 B block L2					
Component	Rocket	SmallBOOMV3	MediumBOOMV3	LargeBOOMV3	MegaBOOMV3	GigaBOOMV3
Pipeline	2-fe/1-de/1-iss	4-fe/1-de/3-iss	4-fe/2-de/4-iss	8-fe/3-de/5-iss	8-fe/4-de/8-iss	8-fe/5-de/9-iss
Execute	–	32-entry ROB	64-entry ROB	96-entry ROB	128-entry ROB	130-entry ROB
IQ (I/M/F)	–	8/8/8	12/20/16	16/32/24	24/40/32	24/40/32
LQ/STQ/nMSHR	–/-/-	8/8/2	16/16/2	24/24/4	32/32/8	32/32/8
Branch Pred.	512-entry BHT, 28-entry BTB			TAGE+BTB, ⟨14,14,28,28,28⟩ KiB		

Rocket. We highlight the benchmarks `qsort`, where lost slots are dominated by Bad Speculation. `qsort` exhibits bad branch prediction accuracy due to an unpredictable branch for pivot comparison. By contrast, `rsort` achieves near-ideal IPC since its control flow is loop-centric and `-O2` removes expensive operations (mul/div). Most Rocket benchmarks are small, yielding negligible Frontend stalls. The benchmark `memcpy` exhibits the largest number of backend stalls. As we show in Fig. 7 (b), roughly half of all backend stalls are Memory Bound stalls.

BOOM. We demonstrate the top-level TMA characterization of BOOM for SPEC CPU2017 Intrate and microbenchmarks in Fig. 7 (g) and Fig. 7 (k) respectively. For SPEC benchmarks, we show all second-level TMA results in (h), (i), (j), and only the Backend category for the microbenchmarks (l). `525.x264_r` stands out with a high retire rate matching IPC, while `505.mcf_r` and `523.xalancbmk_r` are almost 80% Backend Bound. Frontend remains minimal across all benchmarks. Bad Speculation is most considerable for `525.x264_r`. Machine Clears overall represent a small portion of the Bad Speculation category, which is mostly due to branch misses. Microbenchmarks follow a similar breakdown to Rocket, where Dhrystone and Coremark have high IPCs, on BOOM this in the range of 2. `Memcpy` again stands out for being memory bound.

Rocket CS1 : L1D-cache size (c) We run `531.deepsjeng_r` with 16 KiB and 32 KiB L1D caches on Rocket. Reducing cache size causes a 7% slowdown. Fig. 7 (a) shows Backend-bound rises from almost 0% to around 12% with a smaller cache size. Some of the lost slots are caught by Bad Speculation category, signifying stall overlap.

Rocket CS2: Branch inversion (d) We synthesize a branch-heavy benchmark that executes a chain of branch instructions without a loop: `brmiss` (always mispredicted) versus `brmiss_inv` (always predicts correct). Retiring rises from 20% to 33% while Bad-speculation falls from 17% to 6%. The remaining discrepancy reflects a slight misattribution between Bad Speculation and Frontend Bound.

BOOM CS1: Branch inversion (n) The same case study on BOOM shows the opposite effect, where the inverted benchmark is slower than the baseline by 3%. This is because the branch prediction implementation is different. The Bad Speculation category (0% in the base case) explains the slowdown as there is no branch target misprediction.

Rocket CS3: Coremark (e) and (f) Coremark is Core Bound in the Backend. We compare two `-O1` builds: one without instruction scheduling pass, one with `-fschedule-insns -fschedule-insns2`. Both binaries have identical instruction counts, only instruction ordering differs. We observe a ∼4% IPC and runtime improvement, fully explained by a ∼4% reduction in the Backend and Core Bound categories.

BOOM CS1: Coremark (m) Similarly to the previous case study, we can apply the same optimization on BOOM. Here the performance increase is slim as instruction scheduling is less effective on superscalar and OoO pipelines. Nevertheless the runtime improves by 0.3% with the same compiler scheduling pass, with the Backend and specifically the Core Bound category reflecting this improvement. This demonstrates the fidelity of the model

How important are per-lane events? In this work, we model events per lane and argue each lane's signals must be tracked. We explore the impact of not supporting per-lane events. To achieve this, we count every fetch lane individually. BOOM's three-wide fetch produces one `Fetch-bubble` event per lane, and we report per-lane totals for selected benchmarks in Tab. V. For `Fetch-bubble`, lanes are correlated: lane 1 has the fewest bubbles, then lane 2, then lane 3. Thus, as a lightweight heuristic, we could approximate total fetch bubbles as $3 \times$ (`Fetch-bubble1`), yielding Frontend category errors within about $\pm 10\%$ across our suite compared to the full model. The simpler evaluation also realizes gains in the physical design, if a processor designer wanted to implement the event only from one lane as opposed to placing circuitry to monitor all lanes. As we implemented BOOM configurations in a physical design flow (§V-C), we also observed that the 10% decrease in accuracy trades with a reduction in the length of the longest PMU-specific wire by 11.39%.

By contrast, per-lane approximation fails for events like `Uops-issued` or `D$-blocked`. Issue queues are not symmetric (e.g., only the fourth queue handles floating-point μ-ops), so each lane must be tracked separately.

B. Accuracy of Icicle's TMA Implementation

When performance bottlenecks overlap with each other, they introduce inaccuracy in TMA results. The key challenge of identifying such inaccuracy is the lack of ground truth,

TABLE V: Per-lane events per total cycles.

	`Fetch-bubble`			`D$-blocked`			`Uops-issued`				
Benchmark	0	1	2	0	1	2	0	1	2	3	4
505.mcf_r	0.03	0.05	0.09	0.41	0.06	0.14	0.34	0.30	0.12	0.05	0.00
523.xalancbmk_r	0.03	0.07	0.11	0.47	0.09	0.18	0.29	0.31	0.12	0.02	0.00
541.leela_r	0.04	0.07	0.12	0.04	0.08	0.10	0.44	0.09	0.38	0.20	0.00
525.x264_r	0.02	0.05	0.08	0.03	0.06	0.08	0.24	0.16	0.43	0.29	0.00
548.exchange2_r	0.02	0.04	0.06	0.00	0.00	0.00	0.84	0.61	0.26	0.13	0.00
500.perlbench_r	0.04	0.07	0.11	0.05	0.08	0.10	0.02	0.13	0.14	0.05	0.00
mm	0.02	0.03	0.04	0.01	0.04	0.06	0.47	0.09	0.03	0.01	0.87
memcpy	0.03	0.04	0.05	0.13	0.67	0.74	0.69	0.13	0.06	0.00	0.00

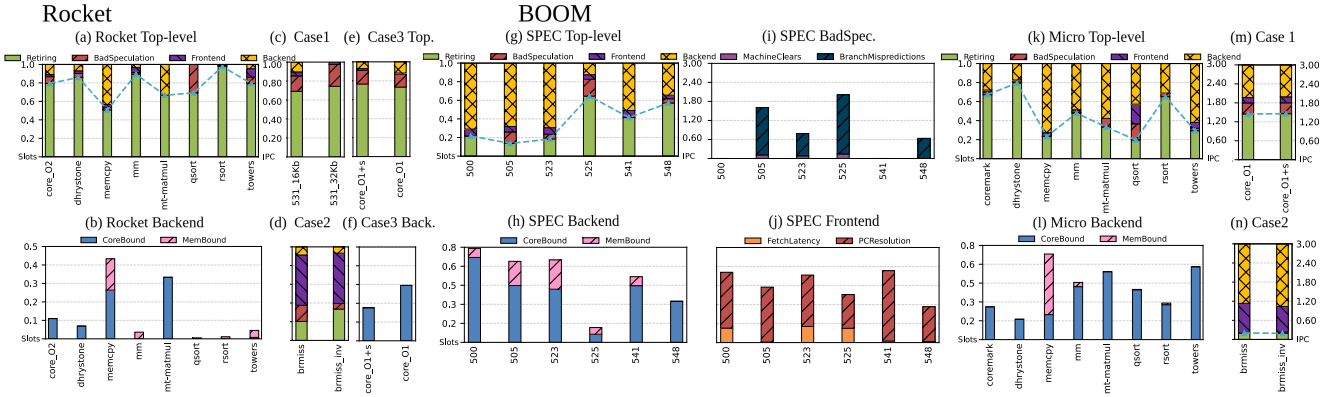


Fig. 7: All TMA results for Rocket and BOOM on SPEC CPU2017 Intrate, microbenchmarks, and selected case studies.

which includes all overlapping performance bottlenecks. We address this challenge by using Icicle's tracing infrastructure to record all overlapping bottlenecks. Specifically, we leverage traces to build a temporal TMA model that helps us quantify the upper bound of TMA inaccuracies.

How much do Bad Speculation and Frontend overlap? A full microarchitectural trace lets us quantify overlaps. For example, Frontend and Bad Speculation can both bottleneck and mask each other's lost slots; something counter values alone cannot reveal. Fig. 8a shows one such overlap: an I-cache miss triggers a branch misprediction before the miss completes. Even with a detailed trace, slot classification remains challenging. We cannot prove solely from the trace that the I-cache miss or branch miss is responsible for the `Fetch-bubbles`. Most likely, this I-cache miss is not responsible, and is only the prefetcher missing, since Recovering perfectly overlaps them. However, we can use the trace to provide an upper bound on the number of overlapping slots.

Temporal TMA		
Overlap Frontend,`I$-miss` & Bad Speculation	0.01%	
Frontend	3.33%	± 0.30%
Bad Speculation	18.15%	± 0.06%

TABLE VI: Quantifying upper-bound for TMA class overlap.

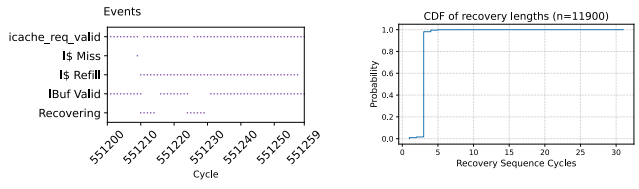
To measure this, we sampled trace sequences for a total 1.5 million cycles across all benchmarks. Our trace analyzer scans for overlaps between I-Cache Refills and Recovering, using a rolling window padded by 50 cycles to conservatively bound the overlap. Any fetch bubble within that window could count toward either category. Tab. VI reports that about 0.01% of all total slots may be an overlapping slot. If we assume the worst case and all of these 0.01% of slots were to be placed the Frontend instead, the perturbation would be $\frac{0.01}{3.33} * 100 = 0.30\%$. The same calculation can be made for Bad Speculation perturbation.

Can we approximate Recovering with a constant? We use Icicle's tracing infrastructure to explore further approximations. Specifically, we identify the Recovery sequence by measuring the number of consecutive cycles the processor Frontend requires to recover after a branch misprediction, as we show in Fig. 8a. Fig. 8b shows the CDF of all such Recovery sequences. Almost every sequence lasts exactly four cycles, showing that the Frontend needs four cycles to resume producing valid instructions after a misprediction. However, a long tail extends beyond 30 cycles: Icicle's trace reveals the single longest Recovery occurs when a fence instruction immediately follows the misprediction. Conversely, the shortest Recovering sequences arise from two back-to-back pipeline flushes.

C. Physical design overhead

We wanted to establish confidence in the practicality of these designs in a synthesized design. Hence, we implemented each of the counter implementations in the five different sizes of the BOOM processor, small, medium, large, mega, and giga Tab. IV. Our closed-source Cadence-based flow passed each configuration through logic synthesis, floorplanning, and placement with the open-source ASAP7 physical design kit as its technology node [33]. Our infrastructure allowed us to rapidly prototype different configurations. Any future modifications, such as adding levels of TMA with new events, would quickly realize metrics for physical overheads through our infrastructure. Designers can then evaluate physical cost tradeoffs.

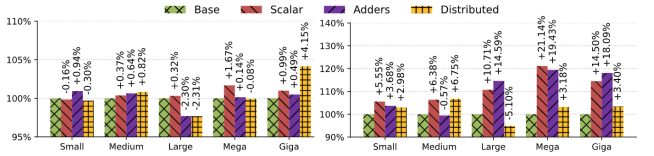
We collect power, area, and wirelength metrics for Icicle. Icicle incurs a maximum overhead of 4.15% in power (Fig. 9a), 1.54% in area, and 9.93% in total wirelength. Additionally, all designs pass constraints at a 200MHz clock frequency. This matches the desired frequency of the base processor without TMA events or their corresponding increment logic. Hence, we conclude that our implementation is off the existing critical path and does not introduce a new critical path. While



(a) Overlap example.

(b) Sequence Distribution

Fig. 8: Temporal TMA examples.



(a) Total core power. (b) Largest CSR combinational delay.

Fig. 9: Post-placement metrics (lower is better).

this frequency may be lower than previous taped-out BOOM designs, we did not have access to an ASAP7 memory compiler. Consequently, the flow unrolled all memories into register arrays which complicated the placement and routing of the designs. However, that frequency stability, along with the low overhead numbers for area, demonstrate that these designs are physically practical.

We also collected wirelength and maximum combinational delay statistics, specifically scoped to the CSR file, as that contains all PMU-relevant logic. Fig. 9b shows the normalized longest combinational delay in a path between two registers that crosses the CSR file (thus containing our modifications and added interfaces with the entire core). In the small and medium configurations, we observe that the adders implementation performs equal to or better than the distributed counters. However, the likely reason is that there are fewer sources for the events in these smaller sizes. As a result, the circuit overhead of distributed counters outweighs its scalability. However, as the size increases, there are more sources for each event. The longer adder network of the adders approach extends the combinational delay. This metric highlights the scalability of distributed counters.

VI. Related Work

Performance characterization. Ahmad Yasin first proposed Top-down analysis approach on Intel processors [107] and laid the foundations of the hierarchical structure that categorizes the slot breakdown, later adopted by AMD [55] and ARM [75]. TMA improves on earlier approaches that assign a static cost to events, for instance a cache miss [17], [20], [44]. To the best of our knowledge, SiFive is the only industry group that has applied top-down characterization to a RISC-V core, specifically the P470 [12], [74], but only at top level. XiangShan is the only open-source out-of-order core that currently supports TMA [96], [105], but their implementation is not directly transferable to Rocket and BOOM, and no accuracy studies exist. To the best of our knowledge, their implementation is out-of-band only and is not synthesizable. A RISC-V working group is currently developing a specification for a standardized set of PMU events to support TMA [10]. However, this specification only standardizes the name of the event if such an event is implemented. Providing an open-source implementation and evaluation infrastructure, our tool would facilitate such standardization efforts. Closely related to Top-down, Eyerman et al. [30], [30], [42] introduce Cycles Per Instruction (CPI) stacks to uncover stalls to uncover sources of stalls. More recent works improve this insight by proposing CPI stacks per stage [43], [45] or per instruction cycle stacks by Gottschall et al. [51], [53].

Counters architecture. Weaver et al. [102] diagnosed inaccuracies of commercial processors as issues with instruction-level counters that stem from benchmark and operating system design. Other works [84] have synthesized superscalar PMU designs to FPGAs, but lack a quantitative analysis of design choices that would not integrate well with Chipyard.

Out-of-band tools. Our out-of-band tools expand on Fireperf, which includes TraceRV and AutoCounter [59]. AutoCounter allows for annotating boolean signals and producing counter values at the end of simulation, whereas Fireperf offers a suite of pre-defined, well-evaluated events that enable accurate in-band characterization. Our out-of-band evaluation tracing is most comparable to TEA's and TIP's evaluation [51], [53] that use TraceDoctor, however, TraceDoctor was built before a major redesign of FireSim and is incompatible with new FireSim versions.

VII. Conclusion

We have enabled TMA on Rocket and BOOM with new performance events and shown that our model can identify performance bottlenecks. To support BOOM, we implemented two new counters architectures for single-cycle, multi-increment tracking of concurrent events. We extended TraceRV to enable microarchitectural tracing that helps design and quantify overlaps in characterization models. In future work, we aim to extend the TMA hierarchy to third- and fourth levels, improve accuracy, explore performance characterization on heterogeneous systems on Chipyard, and expand the temporal TMA model to quantify inaccuracies for every overlap.

VIII. Acknowledgments

We thank the anonymous reviewers for their insightful feedback. This work was supported by generous gifts from Google. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the funding agencies. We thank Prathmesh Patel for helping us on earlier work on the Ibex processor.

IX. Artifact Appendix

A. Abstract

This artifact provides all resources required to reproduce top-down and VLSI results of Icicle. It includes raw logs from FireSim FPGA simulations, analysis scripts, physical design flow scripts, and Icicle's full code base. To reproduce microbenchmark results without FPGA hardware, we provide a meta-simulation workflow based on Verilator. While larger benchmarks require specialized FPGA infrastructure, the Verilator flow enables reproduction of results at a smaller scale.

B. Artifact check-list (meta-information)

- **Run-time environment:** Ubuntu 22.04
- **Hardware:** x86_64 machine with $>$ 8 cores and $>$ 32 GB RAM
- **Output:** Table 5, Figures 7(a), 7(b), 7(d), 7(k), 7(n), and 9
- **Experiments:** Meta-simulation (Verilator) with new performance counters and top-down analysis
- **Disk space required:** 200 GB
- **Time to prepare workflow:** 2 hours
- **Time to complete experiments:** 10 hours
- **Publicly available:** Yes
- **Code licenses:** Apache 2.0
- **Archived:** https://doi.org/10.5281/zenodo.17059077, https://doi.org/10.5281/zenodo.16916499

C. Description

1) How to access:

- **Icicle:** https://doi.org/10.5281/zenodo.17059077
- **FPGA dataset:** https://doi.org/10.5281/zenodo.16916499

Only the first Zenodo link needs to be downloaded; the rest will follow automatically.

Before beginning, please ensure that you have SSH key access to GitHub.

2) Hardware dependencies: An x86_64 machine with at least 8 cores and more than 32 GB of RAM running Ubuntu 22.04 with 200GB of storage.

D. Installation

First copy the artifact:

```
$ cd ~/
$ wget -O Icicle-main.zip
https://zenodo.org/records/17059077/files/Icicle.zip
$ unzip Icicle-main.zip
```

Follow the setup instructions in the repository README to install conda, system packages, and prepare sudo privileges.

Then, run the setup script. This step should take around 1-2 hours.

```
$ cd ~/Icicle-main
$ bash setup.sh --skip=fpga
```

It is common that the setup fails with a guestmount error, rerun the command:

```
sudo chmod +r /boot/vmlinuz-*
```

Once done, the setup should print `Init completed.` The setup script will download our fork of Chipyard, RocketChip, and BOOM. Icicle is built in the top-level repository around the Chipyard code base.

From this point on, run all scripts inside the built environment.

```
$ source ./env.sh
```

E. Experiment workflow: FPGA dataset

Inside the Icicle repository, run:

```
bash ./plots-iiswc-2025-ae.sh
```

This step should take a few minutes to complete.

1) Expected results: This will download all FPGA simulation results used in the Icicle evaluation section and use the `tma_tool` to generate plots and tables. Results can be found in:

```
~/Icicle-main/iiswc-2025-ae-out/results/
```

and include all subfigures in Figure 7 (and additional TMA levels) and Table 5.

Raw data is in:

```
~/Icicle-main/iiswc-2025-ae-out/data/
```

which includes collected counters, uartlog, and binary traces. Further analysis can be done independently using the `tma_tool` commands.

F. Experiment workflow: Microbenchmark meta-simulation

This can take 8-10 hours to complete. Inside the Icicle repository, run:

```
make init
bash run-iiswc-2025-ae.sh
```

`make init` sets up the build recipes with performance characterization configurations that include performance counters and counters architectures.

If an `instance liveness` error is encountered, rerun the Firesim ssh agent setup from the Readme.

1) Expected results: This script will initialize the FireSim configuration templates and run Verilator-based meta-simulation for the RISC-V microbenchmarks suite. This script will first build FireMarshal workloads with the `perf` harness and subsequently run the workloads on both Rocket and BOOM. We use the LargeBoom configuration with `Scalar`, `AddWires`. We compare counter values of `AddWires` and `DistributedCounters`. The latter requires post-processing based on counter width and the artifact harness is set up for add wires.

All simulation outputs including uartlogs, counter values, TMA plots, and TMA numbers will be in:

```
~/Icicle-main/iiswc-2025-ae-out/<rocket|boom>
```

Counter values comparison will be in:

```
~/Icicle-main/iiswc-2025-ae-out/counters-comparison
```

G. Experiment workflow: Physical Design

This section walks through implementing different Chipyard configurations in a Cadence-based physical design flow and extracting metrics of interest. The script below iterates through each of the counter configurations in the Mega size of Boom. Running from scratch, the four designs should run through placement in around a day.

```
source ./env.sh
bash ./run-iiswc-2025-vlsi-ae.sh
```

This produces various plots that evaluate tradeoffs in physical metrics. The two files listed below correspond to Figures 9(a) and 9(b) in our paper.

```
find ${PC_DIR}/iiswc-2025-ae-out/vlsi -name "*power*"
find ${PC_DIR}/iiswc-2025-ae-out/vlsi -name
"*longest_csr_path*"
```

H. Experiment customization

We only provide commands to run meta-simulation. However, if an FPGA is available all commands work identically to meta-simulation, allowing for running SPEC and Coremark benchmarks.

I. Notes

Note that results from meta-simulation and FPGA simulation may differ slightly due to differences in how memory behavior is modeled. While exact numbers can vary, overall bottlenecks remain consistent.

References

[1] "SiFive U54-MC Core Complex Manual v1p0," October 2017, available at https://static.dev.sifive.com/U54-MC-RVCoreIP.pdf.

[2] "The RISC-V Instruction Set Manual Volume I," https://lf-riscv.atlassian.net/wiki/spaces/HOME/pages/16154769/RISC-V+Technical+Specifications, April 2024.

[3] "AMD uProf User Guide," https://www.amd.com/content/dam/amd/en/documents/developer/uprof-v4.0-gaGA-user-guide.pdf, n.d., [Accessed 05-09-2025].

[4] "CVA6 performance counters," https://docs.openhwgroup.org/projects/cva6-user-manual/01_cva6_user/CSR_Performance_Counters.html, n.d., [Accessed 17-05-2025].

[5] "Intel perfmon," https://github.com/intel/perfmon, n.d., [Accessed: 2025-09-05].

[6] "Intel pmu profiling tools," https://github.com/andikleen/pmu-tools, n.d., [Accessed 10-06-2025].

[7] "Intel VTune Profiler Performance Analysis Cookbook: Top-down Microarchitecture Analysis Method," https://www.intel.com/content/www/us/en/docs/vtune-profiler/cookbook/2023-0/top-down-microarchitecture-analysis-method.html, n.d., [Accessed 14-12-2024].

[8] "Micro-architectural event tracking," https://docs.boom-core.org/en/latest/sections/uarch-counters.html, n.d., [Accessed 10-12-2024].

[9] "OpenSBI," https://github.com/riscv-software-src/opensbi, n.d., [Accessed 14-06-2025].

[10] "Performance event sampling rvs-2770," https://github.com/riscv/riscv-performance-events/pull/18, n.d., [Accessed 10-12-2024].

[11] "riscv-tests," https://github.com/riscv-software-src/riscv-tests, n.d., [Accessed 14-12-2024].

[12] "SiFive P400-Series Datasheet," https://www.sifive.com/document-file/p400-series-datasheet, n.d., [Accessed 13-06-2025].

[13] "Target-to-Host Bridges documentation," https://docs.fires.im/en/latest/Golden-Gate/Bridges.html, n.d., [Accessed 14-06-2025].

[14] "The Berkeley Out-of-Order Machine," https://docs.boom-core.org/en/latest/sections/intro-overview/boom.html, n.d., [Accessed 23-06-2025].

[15] "Xiangshan top-down," https://github.com/OpenXiangShan/XiangShan/blob/master/scripts/top-down/top_down.py, n.d., [Accessed: 2025-09-05].

[16] A. Abel, Y. Li, R. O'Grady, C. Kennelly, and D. Gove, "A profiling-based benchmark suite for warehouse-scale computers," in *International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2024, pp. 325–327.

[17] A. Ailamaki, D. J. DeWitt, M. D. Hill, and D. A. Wood, "DBMSs on a modern processor: Where does time go?" in *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK*, 1999, pp. 266–277.

[18] A. Amid, D. Biancolin, A. Gonzalez, D. Grubb, S. Karandikar, H. Liew, A. Magyar, H. Mao, A. Ou, N. Pemberton *et al.*, "Chipyard: Integrated design, simulation, and implementation framework for custom socs," in *International Symposium on Microarchitecture (MICRO)*, vol. 40, no. 4, 2020, pp. 10–21.

[19] S. Anand, M. Friedman, M. Giardino, and G. Alonso, "Skip it: Take control of your cache!" in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2024, pp. 1077–1094.

[20] J. M. Anderson, L. M. Berc, J. Dean, S. Ghemawat, M. R. Henzinger, S.-T. A. Leung, R. L. Sites, M. T. Vandevoorde, C. A. Waldspurger, and W. E. Weihl, "Continuous profiling: Where have all the cycles gone?" *Transactions on Computer Systems (TOCS)*, vol. 15, no. 4, pp. 357–390, 1997.

[21] M. Arora, S. Nath, S. Mazumdar, S. B. Baden, and D. M. Tullsen, "Redefining the role of the cpu in the era of cpu-gpu integration," *IEEE Micro*, vol. 32, no. 6, pp. 4–16, 2012.

[22] K. Asanovic, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, D. Dabbelt, J. Hauser, A. Izraelevitz *et al.*, "The rocket chip generator," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17*, vol. 4, pp. 6–2, 2016.

[23] G. Ayers, J. H. Ahn, C. Kozyrakis, and P. Ranganathan, "Memory hierarchy for web search," in *International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2018, pp. 643–656.

[24] G. Ayers, N. P. Nagendra, D. I. August, H. K. Cho, S. Kanev, C. Kozyrakis, T. Krishnamurthy, H. Litz, T. Moseley, and P. Ranganathan, "Asmdb: understanding and mitigating front-end stalls in warehouse-scale computers," in *International Symposium on Computer Architecture (ISCA)*, 2019, pp. 462–473.

[25] C. Bai, Q. Sun, J. Zhai, Y. Ma, B. Yu, and M. D. Wong, "BOOM-explorer: RISC-V BOOM microarchitecture design space exploration," *Design Automation of Electronic Systems*, vol. 29, no. 1, pp. 1–23, 2023.

[26] S. S. Banerjee, S. Jha, Z. Kalbarczyk, and R. K. Iyer, "Bayesperf: minimizing performance monitoring errors using bayesian statistics," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2021, pp. 832–844.

[27] L. Berger-Vergiat, S. G. Cardwell, B. Feinberg, S. D. Hammond, C. Hughes, M. Levenhagen, and K. Pedretti, "Evaluation of HPC workloads running on open-source RISC-V hardware," in *International Conference on High Performance Computing*. Springer, 2023, pp. 538–551.

[28] Björn Gottschall and Lieven Eeckhout and Magnus Jahre, "Per-instruction cycle stacks through time-proportional event analysis," in *International Symposium on Microarchitecture (MICRO)*, vol. 44, no. 4. IEEE, 2024, pp. 27–33.

[29] B. Boroujerdian, Y. Jing, D. Tripathy, A. Kumar, L. Subramanian, L. Yen, V. Lee, V. Venkatesan, A. Jindal, R. Shearer *et al.*, "FARSI: An early-stage design space exploration framework to tame the domain-specific system-on-chip complexity," *Transactions on Embedded Computing Systems*, vol. 22, no. 2, pp. 1–35, 2023.

[30] M. Breughe, S. Eyerman, and L. Eeckhout, "A mechanistic performance model for superscalar in-order processors," in *International Symposium on Performance Analysis of Systems & Software (ISPASS)*. IEEE, 2012, pp. 14–24.

[31] J. Bucek, K.-D. Lange, and J. v. Kistowski, "SPEC CPU2017: Next-generation compute benchmark," in *International Conference on Performance Engineering*, 2018, pp. 41–42.

[32] D. Chen, T. Moseley, and D. X. Li, "Autofdo: Automatic feedback-directed optimization for warehouse-scale applications," in *Code Generation and Optimization (CGO)*. IEEE, 2016.

[33] L. T. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, C. Ramamurthy, and G. Yeric, "ASAP7: A 7-nm finFET predictive process design kit," *Microelectronics Journal*, vol. 53, pp. 105–115, 2016.

[34] K. D. Cooper, L. T. Simpson, and C. A. Vick, "Operator strength reduction," *Transactions on Programming Languages and Systems (TOPLAS)*, vol. 23, no. 5, pp. 603–625, 2001.

[35] C. Curtsinger and E. D. Berger, "Stabilizer: Statistically sound performance evaluation," *ACM SIGARCH Computer Architecture News*, vol. 41, no. 1, pp. 219–228, 2013.

[36] Curtsinger, Charlie and Berger, Emery D, "Coz: Finding code that counts with causal profiling," in *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2015, pp. 184–197.

[37] W. J. Dally, Y. Turakhia, and S. Han, "Domain-specific hardware accelerators," *Communications of the ACM*, vol. 63, no. 7, pp. 48–57, 2020.

[38] A. C. De Melo, "The new linux'perf'tools," in *Slides from Linux Kongress*, vol. 18, 2010, pp. 1–42.

[39] J. M. Domingos, T. Rocha, N. Neves, N. Roma, P. Tomás, and L. Sousa, "Supporting RISC-V Performance Counters Through Linux Performance Analysis Tools," in *International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 2023, pp. 94–101.

[40] J. S. Emer and D. W. Clark, "A characterization of processor performance in the VAX-11/780," *ACM SIGARCH Computer Architecture News*, vol. 12, no. 3, pp. 301–310, 1984.

[41] V. Espindola, L. Zago, H. Yviquel, and G. Araujo, "Source matching and rewriting for MLIR using string-based automata," *Transactions on Architecture and Code Optimization*, vol. 20, no. 2, pp. 1–26, 2023.

[42] S. Eyerman, L. Eeckhout, T. Karkhanis, and J. E. Smith, "A mechanistic performance model for superscalar out-of-order processors," *Transactions on Computer Systems (TOCS)*, vol. 27, no. 2, pp. 1–37, 2009.

[43] S. Eyerman, W. Heirman, K. Du Bois, and I. Hur, "Multi-stage cpi stacks," *IEEE Computer Architecture Letters*, vol. 17, no. 1, pp. 55–58, 2017.

[44] S. Eyerman, J. E. Smith, and L. Eeckhout, "Characterizing the branch misprediction penalty," in *International Symposium on Performance Analysis of Systems & Software (ISPASS)*. IEEE, 2006, pp. 48–58.

[45] Eyerman, Stijn and Heirman, Wim and Du Bois, Kristof and Hur, Ibrahim, "Extending the performance analysis tool box: Multi-stage CPI stacks and FLOPS stacks," in *International Symposium on Performance Analysis of Systems & Software (ISPASS)*. IEEE, 2018, pp. 179–188.

[46] Z. Fu, R. Tedeschi, G. Ottavi, N. Wistoff, C. Fuguet, D. Rossi, and L. Benini, "Ramping up open-source RISC-V cores: Assessing the energy efficiency of superscalar, out-of-order execution," in *International Conference on Computing Frontiers*, 2025.

[47] S. Gal-On and M. Levy, "Exploring coremark a benchmark maximizing simplicity and efficacy," *The Embedded Microprocessor Benchmark Consortium*, 2012.

[48] P. B. Gibbons and S. S. Muchnick, "Efficient instruction scheduling for a pipelined architecture," in *SIGPLAN Symposium on Compiler Construction*, 1986, pp. 11–16.

[49] A. Gonzalez, A. Kolli, S. Khan, S. Liu, V. Dadu, S. Karandikar, J. Chang, K. Asanovic, and P. Ranganathan, "Profiling hyperscale big data processing," in *International Symposium on Computer Architecture (ISCA)*, 2023, pp. 1–16.

[50] B. Gottschall, S. C. de Santana, and M. Jahre, "Balancing accuracy and evaluation overhead in simulation point selection," in *International Symposium on Workload Characterization (IISWC)*. IEEE, 2023, pp. 43–53.

[51] B. Gottschall, L. Eeckhout, and M. Jahre, "TIP: Time-proportional instruction profiling," in *International Symposium on Microarchitecture (MICRO)*, 2021, pp. 15–27.

[52] B. Gottschall and M. Jahre, "Tracedoctor: Versatile high-performance tracing for firesim," in *The First FireSim and Chipyard User and Developer Workshop at ASPLOS*, 2023.

[53] Gottschall, Björn and Eeckhout, Lieven and Jahre, Magnus, "TEA: Time-proportional event analysis," in *International Symposium on Computer Architecture (ISCA)*, 2023, pp. 1–13.

[54] M. Hill and V. J. Reddi, "Gables: A roofline model for mobile socs," in *International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2019, pp. 317–330.

[55] M. Jarus and A. Oleksiak, "Top-down characterization approximation based on performance counters architecture for amd processors," *Simulation Modelling Practice and Theory*, vol. 68, pp. 146–162, 2016.

[56] Z. Jiang, K. Yang, N. Fisher, N. Guan, N. C. Audsley, and Z. Dong, "Hopscotch: A hardware-software co-design for efficient cache resizing on multi-core SoCs," *Transactions on Parallel and Distributed Systems*, vol. 35, no. 1, pp. 89–104, 2023.

[57] S. Karandikar, C. Leary, C. Kennelly, J. Zhao, D. Parimi, B. Nikolic, K. Asanovic, and P. Ranganathan, "A hardware accelerator for protocol buffers," in *International Symposium on Microarchitecture*, 2021, pp. 462–478.

[58] S. Karandikar, H. Mao, D. Kim, D. Biancolin, A. Amid, D. Lee, N. Pemberton, E. Amaro, C. Schmidt, A. Chopra *et al.*, "Firesim: Fpga-accelerated cycle-exact scale-out system simulation in the public cloud," in *International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 29–42.

[59] S. Karandikar, A. Ou, A. Amid, H. Mao, R. Katz, B. Nikolić, and K. Asanović, "Fireperf: Fpga-accelerated full-system hardware/software performance profiling and co-design," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2020, pp. 715–731.

[60] S. Karandikar, A. N. Udipi, J. Choi, J. Whangbo, J. Zhao, S. Kanev, E. Lim, J. Alakuijala, V. Madduri, Y. S. Shao *et al.*, "CDPU: Co-designing compression and decompression processing units for hyperscale systems," in *International Symposium on Computer Architecture (ISCA)*, 2023, pp. 1–17.

[61] T. A. Khan, N. Brown, A. Sriraman, N. K. Soundararajan, R. Kumar, J. Devietti, S. Subramoney, G. A. Pokam, H. Litz, and B. Kasikci, "Twig: Profile-guided btb prefetching for data center applications," in *International Symposium on Microarchitecture (MICRO)*, 2021, pp. 816–829.

[62] T. A. Khan, I. Neal, G. Pokam, B. Mozafari, and B. Kasikci, "DMon: Efficient detection and correction of data locality problems using selective profiling," in *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, Jul. 2021.

[63] T. A. Khan, A. Sriraman, J. Devietti, G. Pokam, H. Litz, and B. Kasikci, "I-SPY: Context-driven conditional instruction prefetching with coalescing," in *International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 146–159.

[64] T. Khan, M. Ugur, K. Nathella, D. Sunwoo, H. Litz, D. A. Jiménez, and B. Kasikci, "Whisper: Profile-guided branch misprediction elimination for data center applications," in *International Symposium on Microarchitecture (MICRO)*, 2022.

[65] T. A. Khan, M. Ugur, K. Nathella, D. Sunwoo, H. Litz, D. A. Jiménez, and B. Kasikci, "Whisper: Profile-guided branch misprediction elimination for data center applications," in *International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 19–34.

[66] T. A. Khan, D. Zhang, A. Sriraman, J. Devietti, G. Pokam, H. Litz, and B. Kasikci, "Ripple: Profile-guided instruction cache replacement for data center applications," in *International Symposium on Computer Architecture (ISCA)*, Jun. 2021.

[67] G. Kornaros and D. Pnevmatikatos, "A survey and taxonomy of on-chip monitoring of multicore systems-on-chip," *Design Automation of Electronic Systems (TODAES)*, vol. 18, no. 2, pp. 1–38, 2013.

[68] C. E. Leiserson, N. C. Thompson, J. S. Emer, B. C. Kuszmaul, B. W. Lampson, D. Sanchez, and T. B. Schardl, "There's plenty of room at the top: What will drive computer performance after moore's law?" *Science*, vol. 368, no. 6495, 2020.

[69] H. Li, D. S. Berger, L. Hsu, D. Ernst, P. Zardoshti, S. Novakovic, M. Shah, S. Rajadnya, S. Lee, I. Agarwal *et al.*, "Pond: Cxl-based memory pooling systems for cloud platforms," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2023, pp. 574–587.

[70] Y. Lv, B. Sun, Q. Luo, J. Wang, Z. Yu, and X. Qian, "Counterminer: Mining big performance data from hardware counters," in *International Symposium on Microarchitecture (MICRO)*. IEEE, 2018, pp. 613–626.

[71] M. Maas, K. Asanović, and J. Kubiatowicz, "A hardware accelerator for tracing garbage collection," in *International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 138–151.

[72] H. Mao, R. H. Katz, and K. Asanovic, "Hardware acceleration for memory to memory copies," *Master's thesis*, 2017.

[73] J. M. May, "MPX: Software for multiplexing hardware performance counters in multithreaded programs," in *International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2001.

[74] C.-Y. Mou, C.-C. Hsiao, and J. Chou, "Top-down microarchitecture analysis approximation based on performance counter architecture for sifive risc-v processors."

[75] J. Mundichipparakkal, K. Nathella, and T. A. Khan, "Arm neoverse n1 core: Performance analysis methodology," https://armkeil.blob.core.windows.net/developer/Files/pdf/white-paper/neoverse-n1-core-performance-v2.pdf, 2021.

[76] Palacharla, Subbarao and Jouppi, Norman P and Smith, James E, "Complexity-effective superscalar processors," in *International Symposium on Computer Architecture (ISCA)*, 1997, pp. 206–218.

[77] M. Panchenko, R. Auler, B. Nell, and G. Ottoni, "Bolt: a practical binary optimizer for data centers and beyond," in *Code Generation and Optimization (CGO)*. IEEE, 2019, pp. 2–14.

[78] M. Panchenko, R. Auler, L. Sakka, and G. Ottoni, "Lightning BOLT: powerful, fast, and scalable binary optimization," in *SIGPLAN International Conference on Compiler Construction*, 2021, pp. 119–130.

[79] N. Pemberton, J. D. Kubiatowicz, and R. H. Katz, "Enabling efficient and transparent remote memory access in disaggregated datacenters," Ph.D. dissertation, Ph. D. Dissertation. University of California at Berkeley, Berkeley, CA, 2018.

[80] R. B. Reese and M. A. Thornton, *Introduction to logic synthesis using Verilog HDL*. Springer Nature, 2022.

[81] J. Rogers, L. Eeckhout, and M. Jahre, "HILP: Accounting for workload-level parallelism in system-on-chip design space exploration," in *International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2025, pp. 1275–1288.

[82] J. Rogers, L. Eeckhout, T. Soliman, and M. Jahre, "Neoscope: How resilient is my soc to workload churn?" in *International Symposium on Computer Architecture (ISCA)*, 2025, pp. 1296–1310.

[83] J. Rogers, T. Soliman, and M. Jahre, "AIO: An abstraction for performance analysis across diverse accelerator architectures," in *International Symposium on Computer Architecture (ISCA)*. IEEE, 2024, pp. 487–500.

[84] V. Salapura, K. Ganesan, A. Gara, M. Gschwind, J. C. Sexton, and R. E. Walkup, "Next-generation performance counters: Towards monitoring over thousand concurrent events," in *International Symposium on Performance Analysis of Systems and software (ISPASS)*. IEEE, 2008, pp. 139–146.

[85] D. Schall, A. Sandberg, and B. Grot, "Warming up a cold front-end with ignite," in *International Symposium on Microarchitecture (MICRO)*, 2023, pp. 254–267.

[86] C. Schmidt and A. Izraelevitz, "A fast parameterized sha3 accelerator," in *tech. rep.* EECS Department, University of California, 2015.

[87] S. Sethumadhavan, R. Desikan, D. Burger, C. R. Moore, and S. W. Keckler, "Scalable hardware memory disambiguation for high ilp processors," in *International Symposium on Microarchitecture (MICRO)*. IEEE, 2003, pp. 399–410.

[88] S. Sheikhpour, D. Z. Metz, E. Jellum, M. Själander, and L. Eeckhout, "Sustainable high-performance instruction selection for superscalar processors," in *International Conference on Computer-Aided Design (ICCAD)*, 2024, pp. 1–9.

[89] H. Shen, K. Pszeniczny, R. Lavaee, S. Kumar, S. Tallam, and X. D. Li, "Propeller: A profile guided, relinking optimizer for warehouse-scale applications," in *International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, 2023, pp. 617–631.

[90] J. E. Smith, "A study of branch prediction strategies," in *International Symposium on Computer Architecture (ISCA)*, 1998, pp. 202–215.

[91] N. K. Soundararajan, P. Braun, T. A. Khan, B. Kasikci, H. Litz, and S. Subramoney, "Pdede: Partitioned, deduplicated, delta branch target

buffer," in *International Symposium on Microarchitecture (MICRO)*, 2021, pp. 779–791.

[92] A. Sriraman and A. Dhanotia, "Accelerometer: Understanding acceleration opportunities for data center overheads at hyperscale," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2020, pp. 733–750.

[93] A. Sriraman, A. Dhanotia, and T. F. Wenisch, "Softsku: Optimizing server architectures for microservice diversity scale," in *International Symposium on Computer Architecture*, 2019, pp. 513–526.

[94] R. Starc, T. Kuchler, M. Giardino, and A. Klimovic, "Serverless? risc more!" in *Proceedings of the 2nd Workshop on SErverless Systems, Applications and MEthodologies*, 2024, pp. 15–24.

[95] W. Su, A. Dhanotia, C. Torres, J. Gandhi, N. Gholkar, S. Kanaujia, M. Naumov, K. Subramanian, V. Andrei, Y. Yuan *et al.*, "DCPerf: An open-source, battle-tested performance benchmark suite for datacenter workloads," in *International Symposium on Computer Architecture*, 2025, pp. 1717–1730.

[96] X. Team, "Xiangshan: An open-source high-performance risc-v processor and infrastructure for architecture research," in *High Performance Computer Architecture (HPCA)*. IEEE, 2025.

[97] R. Tedeschi, G. Ottavi, C. Allart, N. Wistoff, Z. Fu, F. Grillotti, F. De Ambroggi, E. Guidetti, J.-B. Rigaud, O. Potin *et al.*, "CVA6S+: A superscalar RISC-V core with high-throughput memory architecture," *arXiv preprint arXiv:2505.03762*, 2025.

[98] T. N. Theis and H.-S. P. Wong, "The end of moore's law: A new beginning for information technology," *Computing in Science & Engineering*, vol. 19, no. 2, pp. 41–50, 2017.

[99] S. P. Vanderwiel and D. J. Lilja, "Data prefetch mechanisms," *ACM Computing Surveys (CSUR)*, vol. 32, no. 2, pp. 174–199, 2000.

[100] V. M. Weaver and S. A. McKee, "Can hardware performance counters be trusted?" in *2008 IEEE International Symposium on Workload Characterization*. IEEE, 2008, pp. 141–150.

[101] V. M. Weaver, D. Terpstra, and S. Moore, "Non-determinism and overcount on modern hardware performance counter implementations," in *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2013, pp. 215–224.

[102] Weaver, Vincent M and Terpstra, Dan and Moore, Shirley, "Non-determinism and overcount on modern hardware performance counter implementations," in *International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2013, pp. 215–224.

[103] A. R. Weiss, "Dhrystone benchmark," *History, Analysis, Scores and Recommendations, White Paper, ECL/LLC*, 2002.

[104] L. Weng, Y. Hu, P. Huang, J. Nieh, and J. Yang, "Effective performance issue diagnosis with value-assisted cost profiling," in *Proceedings of the Eighteenth European Conference on Computer Systems*, 2023, pp. 1–17.

[105] Y. Xu, Z. Yu, D. Tang, G. Chen, L. Chen, L. Gou, Y. Jin, Q. Li, X. Li, Z. Li *et al.*, "Towards developing high performance risc-v processors using agile methodology," in *International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 1178–1199.

[106] J. Yang, M. Wen, D. Chen, Z. Chen, Z. Xue, Y. Li, J. Shen, and Y. Shi, "HyFiSS: A hybrid fidelity stall-aware simulator for gpgpus," in *International Symposium on Microarchitecture (MICRO)*. IEEE, 2024, pp. 168–185.

[107] A. Yasin, "A top-down method for performance analysis and counters architecture," in *International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2014, pp. 35–44.

[108] D. You, J. Jiang, X. Wang, Y. Du, Z. Tan, W. Xu, H. Wang, J. Guan, R. Wei, S. Zhao *et al.*, "MERE: Hardware-software co-design for masking cache miss latency in embedded processors." ACM, 2025.

[109] D. Zaparanuks, M. Jovic, and M. Hauswirth, "Accuracy of performance counter measurements," in *2009 IEEE international symposium on performance analysis of systems and software*. IEEE, 2009, pp. 23–32.

[110] J. Zhai and Y. Cai, "Microarchitecture design space exploration via pareto-driven active learning," *Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 31, no. 11, pp. 1727–1739, 2023.

[111] Y. Zhang, T. A. Khan, G. Pokam, B. Kasikci, H. Litz, and J. Devietti, "Ocolos: Online code layout optimizations," in *International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 530–545.

[112] Z. Zhang, M. K. Ramanathan, P. Raj, A. Parwal, T. Sherwood, and M. Chabbi, "{CRISP}: Critical path analysis of {Large-Scale} microservice architectures," in *USENIX Annual Technical Conference (USENIX ATC)*, 2022, pp. 655–672.

[113] J. Zhao, A. Gonzalez, A. Amid, S. Karandikar, and K. Asanović, "COBRA: A framework for evaluating compositions of hardware branch predictors," in *International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2021, pp. 310–320.

[114] J. Zhao, B. Korpan, A. Gonzalez, and K. Asanovic, "Sonicboom: The 3rd generation berkeley out-of-order machine," in *Fourth Workshop on Computer Architecture Research with RISC-V*, vol. 5, 2020, pp. 1–7.

[115] Y. Zhong, D. S. Berger, C. Waldspurger, R. Wee, I. Agarwal, R. Agarwal, F. Hady, K. Kumar, M. D. Hill, M. Chowdhury *et al.*, "Managing memory tiers with {CXL} in virtualized environments," in *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2024, pp. 37–56.

[116] F. Zhou, Y. Gan, S. Ma, and Y. Wang, "wperf: Generic off-cpu analysis to identify bottleneck waiting events," in *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2018, pp. 527–543.

[117] T. Zidenberg, I. Keslassy, and U. Weiser, "MultiAmdahl: How should i divide my heterogenous chip?" *IEEE Computer Architecture Letters*, vol. 11, no. 2, pp. 65–68, 2012.