# Towards Open-Source and Automatic Performance Characterization Hardware

Matthew Edwin Weingarten and Tanvir Ahmed Khan*

## Introduction

Performance characterization is the key to unlocking efficient utilization of the underlying processing system. Rapid developments in specialized computing and hardware/software co-design make performance characterization more challenging — as the underlying hardware changes, so must the performance monitoring hardware and the accompanying performance models. CPU vendors have successfully popularized the top-down micro-architectural analysis (TMA) methodology that is effective in identifying true bottlenecks in a processor while abstracting away the hardware implementation [1]. Unfortunately, researchers and practitioners are often limited to open-source RISC-V processors that lack hardware support for TMA, or any other systematic performance characterization methodology. Even the simple scalar in-order RocketCore [2] has not implemented performance hardware capable of providing enough information to support TMA, let alone a more complex Out-of-Order (OoO) and super scalar core like SonicBOOM [3]. Furthermore, the challenges of performance characterization are compounded by the ever-increasing heterogeneity and specialization of hardware [4], and a wholistic performance characterization methodology for an entire System-on-Chip (SoC) remains open-ended. Overall, the lack of hardware supported performance characterization hamstrings the ability to evaluate new hardware designs, for performance tooling to adapt to modern hardware, or even programmers efficiently exploit the target hardware.

In this work, our aim is to close this gap and move towards open-source implementation of performance characterization methodology by adding hardware support for TMA on RocketChip and SonicBOOM. This hardware support is added manually, with an understanding of the micro-architecture details, and our next steps will be applying the manual approach to heterogeneous compute and accelerators. We hope to extract key insights from the manual approach and *automate* this process, so that performance monitoring can keep up, and even help improve on, the rapid development of hardware. Concretely, this work makes the following contributions:

- First open-source PMU support for TMA on RISC-V.
- An analysis showing that the current performance monitoring events on both RocketCore and SonicBOOM lack critical information for performance characterization.
- Identification and addition of missing key performance events alongside a new top-level TMA model. We show that this model requires merely *three* new performance events for both RocketCore and SonicBOOM respectively.
- Evaluation of TMA with our new PMU events by characterizing widely-used benchmarks such as EEMBC CoreMark [5] and Dhrystone [6].
- Evaluation infrastructure to measure power and area overhead of new performance events, showing that these are below %4.

*Columbia University
Corresponding Author: matthew.weingarten@columbia.edu

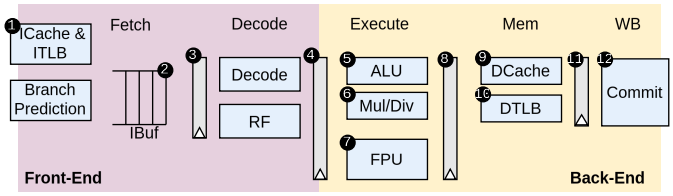


**Figure 1:** *RocketCore pipeline and sources of pipeline stalls marked with numbers* ❶ *-* ⑫.

## Lack of hardware support

The goal of this section is to illustrate that the current Performance Monitoring Unit (PMU) events on RocketCore lack the ability to properly identify the underlying bottlenecks. Assume we have a performance critical application and we wish to identify where in the RocketCore pipeline, highlighted in Figure 1, most of the inefficiencies occur — for example classifying how many cycles are lost in the frontend

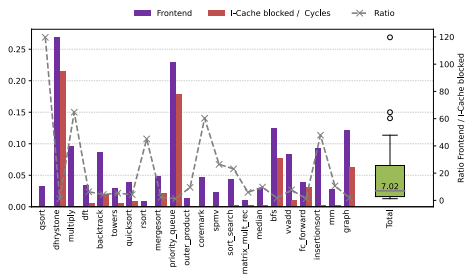


**Figure 2:** *Existing frontend PMU events do not give insight into the true frontend bound metric.*

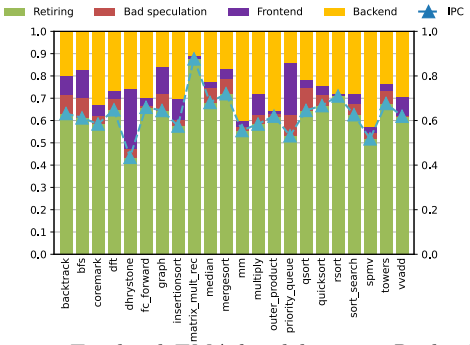


**Figure 3:** *Top-level TMA breakdown on RocketCore.*

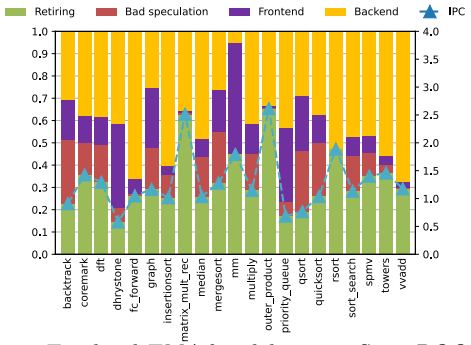


**Figure 4:** *Top-level TMA breakdown on SonicBOOM with core width of 4.*

of the processor. As of now, there are only two existing performance event that collect any information regarding the frontend, namely *I-Cache blocked* and *I-Cache misses*. However, critically, *these events are not enough* to accurately reflect how cycles are lost in the frontend. In Figure 2 we illustrate the relationship between the existing event called *I-Cache blocked* and our frontend bound metric, and clearly shows that bottom-up performance events like cache misses do not allow the model to pinpoint the precise bottleneck.

## Proposed Events & Evaluation

To solve the issue demonstrated in Figure 2, we add an event in the RocketCore pipeline at ❷ in Figure 1 to track the number of cycles lost because the instruction buffer does not have valid instructions to pass to the rest of the pipeline. More events are needed for deeper levels of the TMA model to further pinpoint the root cause. For brevity's sake we do not cover the rest of the events added, but a simple representation of the RocketCore TMA classification is shown in Table 1, where $\mathcal{C}_{\text{Recover}}$, and $\mathcal{C}_{\text{InstrIssued}}$, $\mathcal{C}_{\text{FetchStall}}$ are all newly added events.

Overall we have evaluated our TMA model on both RocketCore in Figure 3 and SonicBOOM in Figure 4. Furthermore, we have designed case studies to examine the accuracy of the new events and the associated TMA model. For example, we run a custom microbenchmark with many compulsory branch mispredictions, use the model to estimate the number of cycles lost due to bad speculation (in this branch mispredictions). Subsequently, we can validate the accuracy of our performance model by checking if the performance

**Table 1:** *TMA calculations with new Events*

| Top-level TMA | |
|---|---|
| Retiring | $\mathcal{C}_{\text{Retired}}$/ $\mathcal{C}_{\text{Total}}$ |
| $\mathcal{B}_{\text{Miss}}$ | $\mathcal{C}_{\text{BrMiss}}$/ ($\mathcal{C}_{\text{BrMiss}}$+ $\mathcal{C}_{\text{Flushes}}$+ $\mathcal{C}_{\text{Fence}}$) |
| Bad spec | (($\mathcal{C}_{\text{InstrIssued}}$- $\mathcal{C}_{\text{Retired}}$) * $\mathcal{B}_{\text{Miss}}$+ $\mathcal{C}_{\text{Recover}}$) / $\mathcal{C}_{\text{Total}}$ |
| Frontend | $\mathcal{C}_{\text{FetchStall}}$/ $\mathcal{C}_{\text{Total}}$ |
| Backend | 1 - Frontend - Bad speculation |

improvements of an *if-conversion* optimization pass, that eliminates every single branch misprediction, can be accurately predicted.

## Future efforts

We have shown that open-source performance monitoring hardware fails to pinpoint root causes and bottlenecks even on both RocketCore and SonicBOOM. Enabling performance characterization for these cores, our work helps researchers and practitioners evaluate hardware and software optimizations for open-source processors. As open-source support to profile and characterize workloads is critical for efficient utilization of next-generation hardware, in future, we will extend this work to out-of-order super-scalar processors and specialized accelerators. As a hardware becomes more heterogeneous, we require a *generalizable* and ideally *automatic* approach to performance characterization and the hardware support that is required. As hardware becomes more and more heterogeneous, we will propose a *generalizable* yet *automatic* approach to provide hardware support for performance characterization.

# References


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

[2] Krste Asanovic et al. "The rocket chip generator". In: *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17* 4 (2016), pp. 6–2.

[3] Jerry Zhao et al. "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.

[4] Alon Amid et al. "Chipyard: Integrated design, simulation, and implementation framework for custom socs". In: *IEEE Micro* 40.4 (2020), pp. 10–21.

[5] Shay Gal-On and Markus Levy. "Exploring coremark a benchmark maximizing simplicity and efficacy". In: *The Embedded Microprocessor Benchmark Consortium* (2012).

[6] Alan R Weiss. "Dhrystone benchmark". In: *History, Analysis, Scores and Recommendations, White Paper, ECL/LLC* (2002).