RISC-V Infrastructure on an FPGA - Midterm Report

Michael Grieco, Prathmesh Patel, and Matthew Weingarten

I. Introduction

The ever-increasing demand for compute is threatened by the fact that Moore's Law and Denard's scaling are tapering out, forcing software to make the most from existing resources [1]. Feedback-directed optimization (FDO) has emerged as a successfully class of techniques to *specialize* software applications and is ubiquitously used for any hyper-scalar operating a datacenter [2, 3, 4, 5, 6, 7, 8, 9, 10]. Typically an FDO system collects data from a live production system and applies collected statistics to modify the software through the compiler, having a feedback directed dynamic runtime, or giving hints for manual code changes. Since profiles are collected from active processes, these profile collection systems are forced to have a low overhead, consequently relying on dedicated hardware components such as the Performance Monitoring Units (PMU) [11].

While there have been numerous works on FDO, innovation on the hardware side remains somewhat limited. Arguably the most promising hardware component dedicated to collecting profiles is Intel's Last Branch Record, that holds a history of the most recent 32 branches taken [12]. The LBR is undoubtedly a success story and a staple feature in modern data-center optimizations [3, 2, 4] Some prior work even shows that the profiles collected with an LBR result in significant performance improvement over non-LBR profiles [3]. This begs the question, what more is possible with more sophisticated profiling hardware. In this work, we aim to explore how we can innovate on the PMU hardware, further enabling FDO and improve application performance.

Furthermore, as hardware accelerators, for instance a graphics processing unit (GPU), continue to improve, a similiar pattern might come to fruition and FDO can be applied accelerators. We cannot predict the future and design a perfect hardware system that will process various applications in new languages designed for parallelism with high energy efficiency. In these cases, profiling hardware is often a secondary consideration, making it difficult for a system to understand the intricacies of the underlying hardware. We believe it worth exploring better infrastructure to support development of PMUs for new and promising accelerator development.

In this project, we design and implement a reconfigurable performance monitoring unit (PMU) built on top of the Ibex RISC-V core [13] to provide real-time feedback to an application. The Ibex core is a simple core to get started and has low overhead in development and evaluation. We argue that our PMU can provide meaningful metrics in a way that the application can react to improve software performance while providing hints for hardware to reconfigure itself to optimize the hardware architecture. In Section II, we define the specific problem to which we contribute along with the existing literature that guided our problem definition. This includes examining implementations of PMUs and how they interact with the processor hardware and the application binary. Section III details how we intend to approach the problem, noting what we want to implement, how we will design it, and the numerous considerations required in a design which builds upon a computer architecture. To show how we intend to evaluate the reliability and provided benefit of our solution, Section IV walks through the desired benchmarks and the physical steps we will perform to attain relevant metrics.

For this midterm report we make the following contributions:

- Simulated bare-metal benchmarks on Rocket-Chip [14], Boom [15], and Ibex [13] using Chipyard [16] to build a Verilator [17] simulation for benchmarks Dhrystone [18] and some custom PMU micro-benchmarks.
- Synthesized an Ibex core using Vivado for future Softcore tests on a ZedBoard FPGA.
- Performed a ASIC-Synthesis workflow for an Ibex core that allows us make sure hardware changes to the PMU do not affect core performance.

The artifact is available under a public GitHub repository (https://github.com/mattweingarten/riscv-pmu-core).

M. Grieco, P. Patel, and M. Weingarten were with the Department of Electrical Engineering, Columbia University, New York, NY, USA e-mail: {mag2346,pp2870,mew2260}@columbia.edu.

Manuscript received April 19, 2005; revised August 26, 2015.

II. Background & Related Work

A. Open-source RISC-V

As we argue in the introduction, RISC-V open source processors are a good target for developing new infrastructure. We analyzed three such designs, each with different levels of performance, complexity, flexibility, and resource usage. Eventually, we would like to implement a solution on a generic processor, but that is a reach goal. Each processor has a full design and has been through synthesis and physical design flows for both ASIC chips and FPGAs [19].

The Ibex core from LowRISC [13] is a small, in-order core with two pipeline stages. It has incredible extensibility, as the SystemVerilog code includes many generics that allow customization of each part of the processor. The advantages of building upon this processor is that the onboarding cost is low; SystemVerilog is a common language so we can easily experiment with changes. Once we go to evaluation, collecting metrics is a relatively straightforward task given a robust logic synthesis and physical design flow. However, it is not a high performance core.

On the other hand, both the RocketChip [14]] and the Berkeley Out-of-Order Machine (BOOM) [15] are open source RISC-V cores implemented in the Chisel hardware construction language. They arguably have a better ecosystem for RISC-V development, as they leverage the Chipyard framework that makes it easy to evaluate new designs. This comes with a further advantadge for longer simulation runs, as the Chipyard framework is design to integrate well with the FPGA accelerated simulation platform FireSim [20], making it easier to simulate entire clusters of processors. RocketChip is a simpler in-order processor comparable to Ibex, while BOOM is a large step up from Ibex in most of the above metrics. Both cores already have a PMU (or Hardware Performance Monitor) implementation that we can use to springboard our project. While they do not share an implementation, they share a common register interface to interact read counter and collect statistics.

Finally, the Xiang Shan [21] core is another out-of-order processor, that is one step up on BOOM in terms of size. To the best of our knowledge, more engineering effort has gone into Xiang Shan to optimize the chip and generally outperforms BOOM, although this has not yet been experimentally validated by us yet. It is also a much larger design, meaning it would be more challenging to launch as a Softcore on an FPGA and our simulation experiments will be more involved. However, Xiang Shan is worth considering to put FDO experiments to the test, as this core more closely resembles a modern datacenter CPU than BOOM.

B. Processor architecture enhancements

Because of the open source nature, those processors provide rich platforms for developing enhancements to the architecture.

Common efforts to improve processor performance look at the cache structure, as memory accesses are the most expensive in terms of time and energy. On the BOOM processor, Anand et al. [22] implemented an advanced cache structure. Using a finite state machine, their cache protocol ensures coherence while allowing for more fine-grained software control. The co-design in this paper demonstrates that good designs should enhance processor performance for existing programs. However, should programmers wish to re-design applications to leverage the custom hardware, there will be an even larger improvement. This work is also proof of how enhancements integrate well with these open source processors. We will draw from their approach to understand how we can effectively design with an existing processor.

Several projects have also aimed to design new processors from the ground up, specifically targeted at improving the efficiency of database computers. Wu et al. [23] implemented one such design which held a configurable number of execution units. They mapped database queries to these elements to be able to efficiently process streams of data coming from memory. Because of the full custom design, this is a considerable and complex project. However, this type of work demonstrates how exploiting known application structure can allow for clever hardware design. We would like to implement infrastructure with such ideas. Without requiring application re-design, a monitoring unit can detect when stream-based applications are executing, and can activate specific hardware to enhance stream processing.

Similarly, Caminal et al. [24] design another fully-custom processor aimed at database processing. Their associative implementation targeted the RISC-V ISA, allowing for a wide variety of use cases in already compiled programs, allowing for the vast sea of production code to share the benefits without requiring changes (although changes can lead to even more benefit). This is the same reason that we want to target RISC-V designs. Their core is meant to run alongside other main cores, and its key feature is that it uses content addressable memory to quickly map between tables in a relational database. At runtime, specific programs benefit from using content addressable memories. Understanding when applications need that and providing it in a configurable manner can provide performance

boosts (primarily for energy by avoiding unnecessary instructions) whilst maintaining the reliability of other applications compiled to RISC-V.

One of the dangers of playing with micro-architecture is that it may compromise the existing design. All those processors, namely BOOM, have gone through multiple design revisions. Each time, the designers have addressed critical paths and carefully tuned everything from the RTL design to the synthesis and placement tuning to ensure that the processor meets timing constraints. Hence, putting different circuitry inside the processor will very likely affect the critical path. The ideas of modifying the register file or changing dispatch logic have all been critical points in those designs. A more feasible idea, however, is to design something to work alongside the processor to provide performance enhancements but staying off the critical path.

C. Processor infrastructure

Processor infrastructure refers to any hardware unit which is operates adjacent to the core (with internal visibility) but attempts to leave the datapath untouched. The unit has visibility to internal wires with taps inserted strategically to monitor bus lines for target events. The only modifications to the internal guts would be during the physical design flow, when fanout of a gate becomes a concern because gates would now have to drive an output to a unit off the data path. However, we expect that buffers to restore drive strength would not compromise placement.

There are various ways that a monitor can integrate with a processor. External accelerators can monitor the memory bus and respond to requests for its specific accelerated function, much like memory-mapped I/O. Power and temperature monitors use a combination of wire taps to count energy-specific events along with temperature sensors to generate real-time statistics. Listl et al. [25] used one such monitor to evaluate prototyped devices. However, this type of monitor can be extended to provide real-time feedback to the processor. As an example, a power controller can use such data determine where activity hot spots are to divert chip resources.

Performance monitoring units (PMUs) focus more on the tap wires and provide readily accessible information to software through registers. Emer and Clark [26] along with Ohja [27] compared various monitoring techniques. Their conclusion was that simple counters inside the processor did not provide enough granularity to understand specific performance in time for a processor. Even if software has access to them, these global counters cannot be isolated to specific applications, and have somewhat coarse granularity. Using more autonomous hardware units like PMUs allows for conditional tracking of events inside the processor. Depending on the complexity, PMUs can monitor specific programs and specific events, which is easily configurable through registers. Liu et al. [28] designed a system to manage how PMUs work with virtual machines. This co-design demonstrates the flexibility of the PMU and how simple state machines embedded in the operating system can elevate the hardware to provide real time fine-grained information. Our work wants to extend the idea of a PMU to be able to perform autonomous analysis based on configurable targets.

The above cores each integrate simple versions of PMUs, or hardware performance monitors as it is known in the RISC-V standards. However, these units are limited to simple performance counters that can be read in software. Our work intends to enhance the performance counters and add extra feedback paths to enable software to make better decisions along with adding autonomous responses.

III. Approach

This section details our approach to developing the performance monitoring unit (PMU) given our requirements and existing work.

A. Trade-offs

There are multiple trade-offs to consider when designing the PMU as a unit, even before evaluating the performance boost it may provide.

Development viability: The open-source cores have seen multiple design iterations, each refining datapaths and improving general performance. The integration and evaluation of these cores, especially BOOM, has a large on-boarding cost. Hence, we must first understand the prerequisites for developing on these platforms. Ibex is a smaller design written in Verilog, so starting a design on it is easily possible. However, simulating it requires Verilator, and the path to synthesizing the design to an FPGA is less clear. BOOM is the opposite in both ways. It is much larger design, so incremental development might be infeasible given the remaining project timeline. The simulation and synthesis of BOOM is heavily documented. It does require lots of infrastructure, which is a larger startup cost but once it begins running, the evaluation path is seamless.

Re-development of existing software: The goal with a PMU is to provide access to accurate performance counters. Existing programs have access to the basic performance counters that exist in processors. Some make use of these values to reconfigure themselves in real time. Hence, any modifications to the PMU have to

continue supporting those programs such that the core can be integrated into systems running legacy RISC-V code. It is infeasible to require developers to put effort into re-developing applications to support new infrastructure. Even if it is a step that can be in the compilation process, re-compiling the vast sea of existing code is not possible. However, that does not prevent programmers from designing future applications with a different lens. New designs can fully leverage new techniques and reap more benefit.

Autonomy versus controllability: Existing RISC-V code must function as expected with any modifications. The PMU, however, can integrate autonomy to boost performance without compromising functionality. This task is difficult because realizing significant performance gains likely involves modifications to the internal microarchitecture. As discussed in Section II, that could negatively impact the processor architecture if we are not careful. Hence, these decisions should be made external to the processor's core microarchitecture. In terms of benefit to new programs, this would involve using control directives to issue commands to the PMU. Triggering counts then reading them is the principal functionality which the performance counters already support on Ibex and BOOM.

IV. Evaluation

There are three levels of evaluation we use to prove our design is reliable and that it provides performance improvement. The first is simulation of our digital design to provide functional verification and run small benchmarks. Then, we run applications on the design synthesized to a field programmable gate array (FPGA). Finally, we put the design through the synthesis and physical design flow for an application-specific integrated circuit (ASIC). To be able to determine the consequences of our design, each step must run for the base processor then the processor into which our designed infrastructure is integrated. Comparing the metrics allows for insights into trade-offs, limitations, and further steps.

A. Software benchmarks

Crucial to both functional verification and physical design are the software applications run on the core. The results from synthesis and physical design provide preliminary results on how many resources the core uses before and after any modifications. However, evaluating the core with live software demonstrates what benefits systems can yield from the PMU. RISC-V has various benchmarks that evaluate computational efficiency for standard metrics. Ideally, we expect to see performance improvement on the basic benchmarks as the autonomous feedback from the PMU may help applications execute better. Some of the benchmarks make use of the performance counters. To demonstrate the value of further development, we want to develop programs that integrate real-time feedback from the PMU.

B. RTL simulation

In this section, we will briefly describe the setup for simulating some simple benchmarks in Verilator for Ibex, RocketChip, and BOOM cores. Currently, our simulation setup uses Chipyard to generate a Verilator simulation binary that can run a bare-metal RISC-V executable. The instructions to recreate our runs are described in the GitHub repository linked in the introduction. Currently, the simulations can successfully run the Dhrystone benchmark, and we have designed a few simple examples that read PMU counters. This is mostly to test the PMU interface that will allow us to incrementally make changes to the PMU and test for correctness.

The functional simulation of the core is a starting sanity check. This provides functional verification for the design, ensuring that it works using a number of unit tests. Once we are confident in the functionality, we can then use benchmark software to verify software's connection to the extra design. Because simulation is limited in speed, these benchmarks must be small or the simulator must be able to save states as a checkpoints so that it is possible to load new software once the boot-up sequence is complete.

We have also used Verilator to run simulations on the Ibex infrastructure and execute a simple c program to get preliminary benchmarks on the processor. This would also allow us to experiment with infrastructural changes on Ibex and test them in a concrete pipeline.

C. ASIC synthesis and physical design

Ibex, along with the other open source processors examined in Section II-A, are optimized for deployment onto custom ASICs. Using FPGAs allows for rapid development and concrete proof that our design can work on a physical board. However, so much of the design effort for these processors went into optimizing critical paths once the design was freed from the constraints of an already laid-out FPGA. Hence, we want to see what effects our design has on the products of that effort.

Using Cadence Genus for logical synthesis and Cadence Innovus for placement and routing, we can collect pre-clock tree synthesis metrics of the base processor and the processor with our design when laid-out with the ASAP7 physical design kit and standard cell library [29]. Figure 1 shows the top-level system after running

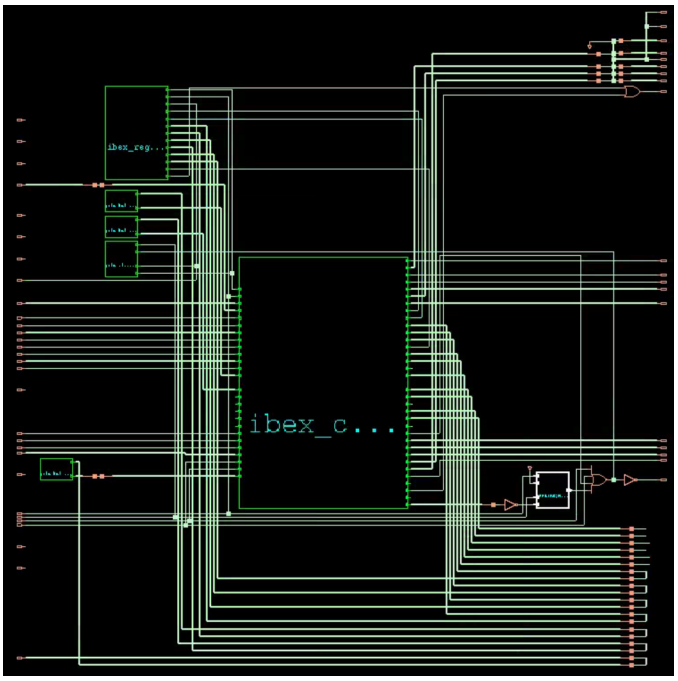

Fig. 1. Top-level schematic of the core after synthesis.

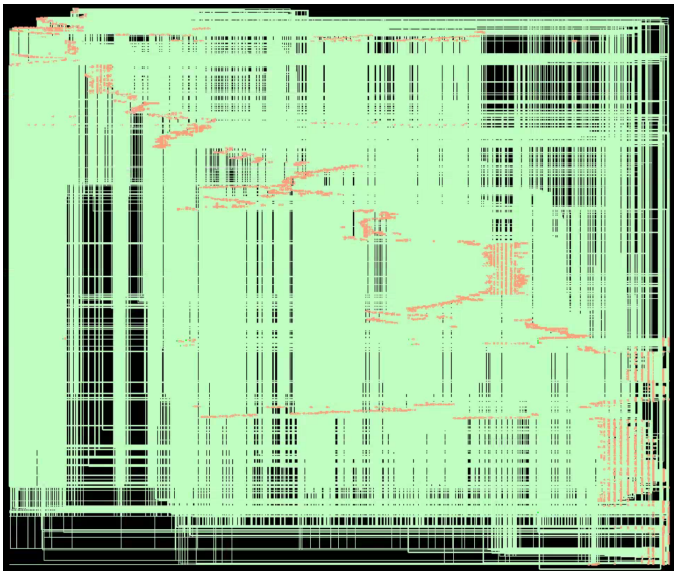

Fig. 2. Internal schematic of the core after synthesis. The orange shows the gates and the green shows the wires connecting them.

the base Ibex core through a synthesis flow and Figure 2 shows the internal makeup of the core. Finally, Figure 3 shows the result of running the placement flow. The only limitation is that we have currently run the flow up to clock tree synthesis. After this point, there are a host of errors which would require thorough debugging effort in Ibex to adapt it to the new flow and physical design kit.

Since we do not intend to tape out this design or

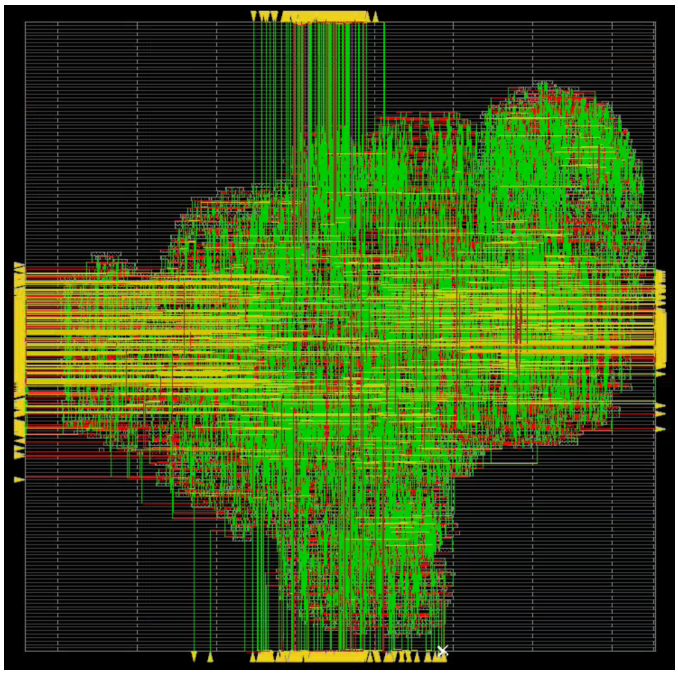

Fig. 3. Top-level visualization of the core after placement.

fabricate it, the physical design results are limited to the models in the tool flow. It is possible to upload activity logs (from RTL simulation) to the performance model which contains switching activity. This way, performance modeling can account for switching execution during live application execution.

V. Conclusion & Future Steps

We have shown in this report our initial efforts on the RISC-V PMU infrastructure, including ASIC synthesis and Verilator simulation of benchmarks. We have successfully simulated PMU access for simple benchmarks. Additionally, we have outlined design consideration for updating the PMU.

Our current direct next steps include:

- Make a minor change to the PMU and see if we can observe changed behaviour in a simulation run. This will give us an end-to-end pipeline to experiment, where can modify the PMU and run a test-suite to test for correctness.
- Understand in depth the details of the PMU implementations on RocketChip, Boom, and Ibex. This will help us understand what events we want to be able to track (for example, adding a register that holds last accessed memory addresses).

References

[1] 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. DOI: 10.1109/MCSE.2017.29.

[2] D. Chen, D. X. Li, and T. Moseley, "Autofdo: Automatic feedback-directed optimization for warehouse-scale applications," in *Proceedings of the 2016 International Symposium on Code Generation and Optimization*, 2016, pp. 12–23.

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

[4] 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 *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, 2023, pp. 617–631.

[5] M. E. Weingarten, T. Theodoridis, and A. Prokopec, "Inlining-benefit prediction with interprocedural partial escape analysis," in *Proceedings of the 14th ACM SIGPLAN International Workshop on Virtual Machines and Intermediate Languages*, 2022, pp. 13–24.

[6] A. Prokopec, G. Duboscq, D. Leopoldseder, and T. Wirthinger, "An optimization-driven incremental inline substitution algorithm for just-in-time compilers," in *2019 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, IEEE, 2019, pp. 164–179.

[7] G. Ayers *et al.*, "Asmdb: Understanding and mitigating front-end stalls in warehouse-scale computers," in *Proceedings of the 46th International Symposium on Computer Architecture*, 2019, pp. 462–473.

[8] S. Lee, T. Johnson, and E. Raman, "Feedback directed optimization of tcmalloc," in *Proceedings of the workshop on Memory Systems Performance and Correctness*, 2014, pp. 1–8.

[9] A. H. Hunter, C. Kennelly, P. Turner, D. Gove, T. Moseley, and P. Ranganathan, "Beyond malloc efficiency to fleet efficiency: A hugepage-aware memory allocator," in *15th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 21)*, 2021, pp. 257–273.

[10] Y. Zhang, T. A. Khan, G. Pokam, B. Kasikci, H. Litz, and J. Devietti, "Ocolos: Online code layout optimizations," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, IEEE, 2022, pp. 530–545.

[11] G. Ren, E. Tune, T. Moseley, Y. Shi, S. Rus, and R. Hundt, "Google-wide profiling: A continuous profiling infrastructure for data centers," *IEEE micro*, vol. 30, no. 4, pp. 65–79, 2010.

[12] R. Rajwar, P. Lachner, L. A. Knauth, and K. K. Lai, *Processor with last branch record register storing transaction indicator*, US Patent 8,479,053, 2013.

[13] *Ibex risc-v core*. [Online]. Available: https://github.com/lowRISC/ibex.

[14] K. Asanovic *et al.*, "The rocket chip generator," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17*, vol. 4, pp. 6–2, 2016.

[15] J. Zhao, B. Korpan, A. Gonzalez, and K. Asanovic, "Sonicboom: The 3rd generation berkeley out-of-order machine," 2020. [Online]. Available: https://people.eecs.berkeley.edu/~krste/papers/SonicBOOM-CARRV2020.pdf.

[16] A. Amid *et al.*, "Chipyard: Integrated design, simulation, and implementation framework for custom socs," *IEEE Micro*, vol. 40, no. 4, pp. 10–21, 2020.

[17] W. Snyder, P. Wasson, D. Galbi, and et al, *Verilator*. [Online]. Available: https://github.com/verilator/verilator.

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

[19] A. Dörflinger *et al.*, "A comparative survey of open-source application-class risc-v processor implementations," in *Proceedings of the 18th ACM International Conference on Computing Frontiers*, ser. CF '21, Virtual Event, Italy: Association for Computing Machinery, 2021, pp. 12–20, ISBN: 9781450384049. DOI: 10.1145/3457388.3458657. [Online]. Available: https://doi.org/10.1145/3457388.3458657.

[20] S. Karandikar *et al.*, "Firesim: Fpga-accelerated cycle-exact scale-out system simulation in the public cloud," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, IEEE, 2018, pp. 29–42.

[21] Y. Xu *et al.*, "Towards developing high performance risc-v processors using agile methodology," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2022, pp. 1178–1199. DOI: 10.1109/MICRO56248.2022.00080.

[22] S. Anand, M. Friedman, M. Giardino, and G. Alonso, "Skip it: Take control of your cache!" In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming*

Languages and Operating Systems, Volume 2, ser. ASPLOS '24, La Jolla, CA, USA: Association for Computing Machinery, 2024, pp. 1077–1094, ISBN: 9798400703850. DOI: 10.1145/3620665.3640407. [Online]. Available: https://doi.org/10.1145/3620665.3640407.

[23] L. Wu, A. Lottarini, T. K. Paine, M. A. Kim, and K. A. Ross, "Q100: The architecture and design of a database processing unit," in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '14, Salt Lake City, Utah, USA: Association for Computing Machinery, 2014, pp. 255–268, ISBN: 9781450323055. DOI: 10.1145/2541940.2541961. [Online]. Available: https://doi.org/10.1145/2541940.2541961.

[24] H. Caminal, Y. Chronis, T. Wu, J. M. Patel, and J. F. Martínez, "Accelerating database analytic query workloads using an associative processor," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ser. ISCA '22, New York, New York: Association for Computing Machinery, 2022, pp. 623–637, ISBN: 9781450386104. DOI: 10.1145/3470496.3527435. [Online]. Available: https://doi.org/10.1145/3470496.3527435.

[25] A. Listl, D. Mueller-Gritschneder, F. Kluge, and U. Schlichtmann, "Emulation of an asic power, temperature and aging monitor system for fpga prototyping," in *2018 IEEE 24th International Symposium on On-Line Testing And Robust System Design (IOLTS)*, 2018, pp. 220–225. DOI: 10.1109/IOLTS.2018.8474284.

[26] J. S. Emer and D. W. Clark, "A characterization of processor performance in the vax-11/780," *SIGARCH Comput. Archit. News*, vol. 12, no. 3, pp. 301–310, Jan. 1984, ISSN: 0163-5964. DOI: 10.1145/773453.808199. [Online]. Available: https://doi.org/10.1145/773453.808199.

[27] A. Ojha, "Techniques in least-intrusive computer system performance monitoring," in *Proceedings. IEEE SoutheastCon 2001 (Cat. No.01CH37208)*, 2001, pp. 150–154. DOI: 10.1109/SECON.2001.923105.

[28] P. Liu, R. Yang, J. Sun, and X. Liu, "Sysoptic: A fine-grained monitoring system for virtual machines based on pmu," in *2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, 2019, pp. 244–2446. DOI: 10.1109/SOSE.2019.00042.

[29] L. T. Clark *et al.*, "Asap7: A 7-nm finfet predictive process design kit," *Microelectronics Journal*, vol. 53, pp. 105–115, 2016, ISSN: 1879-2391. DOI: https://doi.org/10.1016/j.mejo.2016.04.006. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S002626921630026X.