Toki: Profiling HBM Performance on FPGA Systems with RISC-V Soft Cores and PCIe Host DMA Traffic

Andrea Galimberti
Politecnico di Milano
Milan, Italy
andrea.galimberti@polimi.it

Andrea Motta
Politecnico di Milano
Milan, Italy
andrea.motta@polimi.it

Gianni Antichi
Politecnico di Milano
Milan, Italy
gianni.antichi@polimi.it

Davide Zoni
Politecnico di Milano
Milan, Italy
davide.zoni@polimi.it

Abstract

Programmable RISC-V soft cores are becoming more widespread in data-center scenarios, making it crucial to design efficient systems that deploy them on FPGA chips with HBM memory. Toki, released as open source, is the first hardware-software framework that enables profiling the performance of HBM on FPGA accelerator cards by jointly considering (i) the execution of workloads on RISC-V soft cores instantiated on the FPGA and (ii) the injection of memory traffic from the host system via DMA over PCIe, providing insights that cannot be obtained with synthetic traffic generators alone. Extensive experiments target an AMD Alveo U55C card, deploying up to 60 RISC-V compute cores and stressing its HBM2 memory through real-world applications and microbenchmarks with user-defined access patterns. Results showcase how Toki can effectively profile the impact on HBM performance of the compute cores' organization, the workload's memory access patterns, data locality, contention over the memory controllers, and host traffic.

CCS Concepts

• **Hardware** → **Reconfigurable logic and FPGAs**; **Dynamic memory**; *Buses and high-speed links*; • **Computer systems organization** → *Multicore architectures.*

Keywords

FPGA, HBM, RISC-V, Performance profiling, Memory systems, Soft cores, PCI Express, Direct memory access

ACM Reference Format:
Andrea Galimberti, Andrea Motta, Gianni Antichi, and Davide Zoni. 2026. Toki: Profiling HBM Performance on FPGA Systems with RISC-V Soft Cores and PCIe Host DMA Traffic. In *63rd ACM/IEEE Design Automation Conference (DAC '26), July 26–29, 2026, Long Beach, CA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3770743.3803967

1 Introduction

Modern data centers are tasked with managing increasingly demanding workloads, such as AI, real-time data analytics, and HPC, that require both high computational power and efficient memory systems to ensure scalability, energy efficiency, and quality of service. FPGAs have emerged as essential components in this context due to their flexibility [6], enabling the deployment of custom accelerators that exploit massive parallelism and can be dynamically reprogrammed to adapt to evolving application requirements [21].

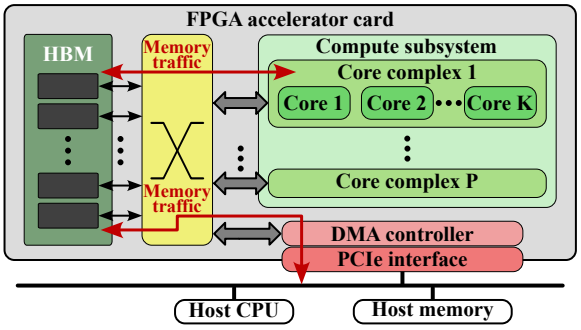


Figure 1: Toki's high-level architecture. Toki profiles HBM performance by concurrently stressing it with traffic from RISC-V soft cores and from the PCIe host system.

While computational throughput has increased rapidly, memory bandwidth has not scaled at the same rate, creating a critical bottleneck for data-intensive applications. HBM addresses this challenge by using 3D packaging to stack memory dies vertically and connect them to processing logic through a silicon interposer with wide I/O interfaces, achieving industry-leading data throughput while maintaining moderate latency [17]. Prior research has explored a wide range of HBM-enabled FPGA computing solutions. Application-specific architectures include accelerators for graph processing [13] and breadth-first search in graphs [18], as well as overlay processors for graph neural networks [27]. Other efforts have used HBM for accelerating sparse matrix-vector multiplication [16, 25, 26, 28, 31], matrix transposition [30], LLM inference [35], fully homomorphic encryption [29], and wave simulations [10].

Emerging trends have demonstrated the viability of using soft-core RISC-V CPUs instantiated on FPGAs as the primary compute elements for systems where programmability, flexibility, and rapid development are prioritized over raw peak performance [5]. This design approach offers a unified software-programmable architecture that simplifies development and supports rapid iteration across a



This work is licensed under a Creative Commons Attribution 4.0 International License.
DAC '26, Long Beach, CA, USA
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2254-7/2026/07
https://doi.org/10.1145/3770743.3803967

broad range of applications. When deployed alongside HBM, these systems can exploit high memory parallelism while maintaining flexibility at the software level [15]. As a result, several recent architectures have explored different strategies for scaling RISC-V cores on HBM-equipped FPGAs. Notable examples include Vortex [5], a soft-core GPU with support for CUDA kernels; Phalanx [11, 12], a kilo-core overlay processor; SPARKLE [1], a 1024-core architecture operating at 400 MHz that leverages barrel processing; and the 32-core overlay processor in [8], where each pair of cores has a dedicated memory controller to maximize HBM bandwidth utilization.

Despite the diversity and scale of these architectures built around soft-core CPUs on FPGAs equipped with HBM, prior efforts have largely focused on design innovation, whereas no systematic framework or platform currently exists to profile the performance of such systems. Existing solutions focus on other memory technologies such as DDR4 [9], isolated aspects of memory performance [23], or accelerator workloads [14], but they all fail to capture the complex interactions between HBM, soft-core CPUs deployed in FPGA logic, and host systems. Conversely, with the growing adoption of RISC-V architectures on FPGA-based platforms, understanding how these architectures perform in conjunction with HBM and PCIe host traffic is critical for optimizing system-level performance.

Contributions

This paper introduces Toki, the first hardware-software (HW/SW) framework designed to profile the performance of HBM on FPGA accelerator cards by jointly considering two critical and interacting sources of memory traffic: *(i)* workloads executed on RISC-V soft cores instantiated on the FPGA and *(ii)* memory traffic injected from the host system via direct memory access (DMA) over the PCIe interface. The methodology, depicted in Fig. 1, allows evaluating a number of performance metrics, including memory bandwidth and latency, under diverse workloads and for different configurations of the RISC-V soft cores. Toki advances the current state of the art through three main contributions.

(1) *RISC-V soft cores* – Toki enables profiling the performance attainable on an FPGA by pairing HBM and a vast design space of programmable RISC-V soft cores, whose number, organization into core complexes, and architectural and microarchitectural parameters are defined at design time.
(2) *PCIe host DMA traffic* – Beyond on-chip compute tasks, Toki supports injection of user-configurable memory traffic originating from the host system via DMA over the PCIe interface of the FPGA accelerator card, enabling realistic concurrent stress on HBM.
(3) *Real-world application workloads* – The programmability of Toki's RISC-V compute cores makes it possible to execute any application and stress HBM with memory access patterns and workloads neglected by state-of-the-art solutions, which are instead limited to synthetic traffic generators.

An *extensive experimental campaign* targets an AMD Alveo U55C card [2] with a second-generation HBM (HBM2) memory, instantiating up to 60 32-bit RISC-V Snitch [34] compute cores and executing both microbenchmarks to stress specific memory access patterns and a set of real-world applications from the PolyBench/C [32] benchmark suite.

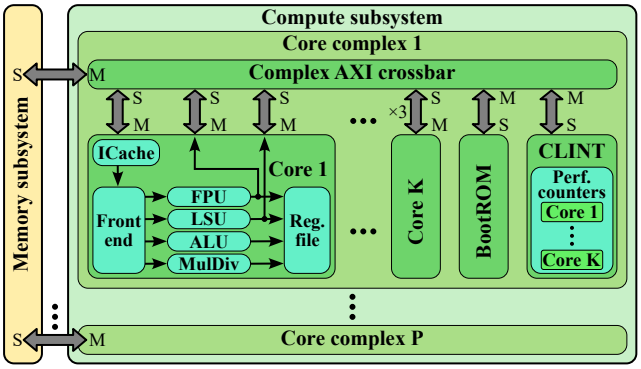


Figure 2: Detailed architecture of Toki's compute subsystem. *Legend: K* number of cores per core complex, *P* number of core complexes, *M* AXI master, *S* AXI slave.

The whole Toki framework, encompassing all its hardware and software components and with support for widely available commercial tools and devices, is *released as open source* under Apache License Version 2.0 to foster further research and ensure full reproducibility of the experiments and analyses.[1]

2 Toki Profiling Framework

The Toki HW/SW framework is designed for deployment on FPGA accelerator cards with a PCIe interface that connects them to the host system, which can access HBM via DMA. Toki's architecture can be split into three main parts, namely, the compute, memory, and DMA subsystems.

2.1 Compute Subsystem

The compute subsystem, shown in more detail in Fig. 2, comprises a cluster of CPU cores organized into multi-core complexes. The number of complexes (P) and the number of cores per complex (K) are configurable parameters defined at design time, offering flexibility to tailor the architecture to the user's requirements.

The CPU cores of each complex are connected to an AXI crossbar that manages memory accesses. The integrated cores must expose an AXI4 master interface to communicate with the crossbar. Depending on the core design, they can feature multiple separate AXI4 channels for instruction fetch, integer load-store unit (LSU), and floating-point load-store instructions of the floating-point unit (FPU). The AXI4 interface between the cores and the crossbar operates at the core clock frequency, avoiding clock domain crossings, and is parametric in terms of data width and address width. The compute subsystem's flexibility allows heterogeneous configurations where different core designs can be instantiated either within the same complex or across different complexes. Cores may support various cache architectures, including cacheless designs or configurations with separate or unified instruction and data caches.

Each core complex contains all the necessary support logic to operate the cores, including a BootROM and a core-local interrupt (CLINT) controller that manages timer and software interrupts. The BootROM read-only memory stores, for each core, the boot

[1] Sources available at https://github.com/hardware-fab/Toki.

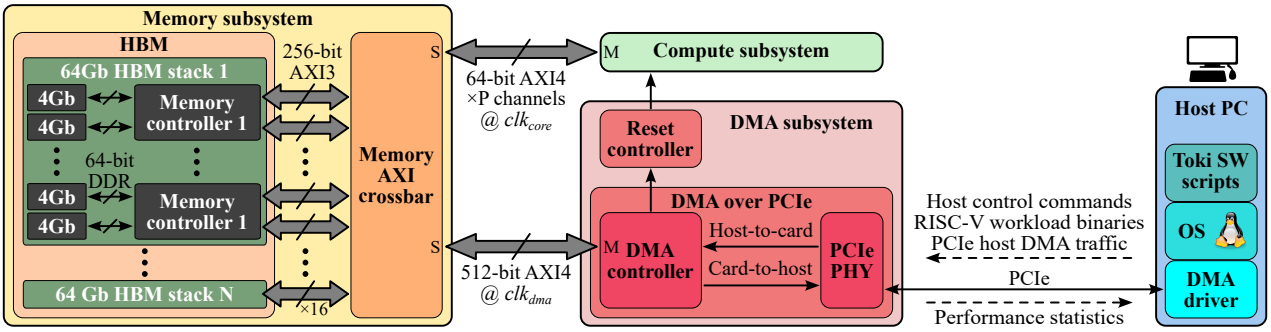


Figure 3: Detailed architecture of Toki's memory and DMA subsystems. *Legend: P* number of core complexes, *N* number of HBM stacks, *M* AXI master, *S* AXI slave.

code that initializes peripherals and jumps to the memory region assigned to the core, including both instructions and data. In addition, each core is allocated a memory region for data exchange with the host system. The CLINT controller of each core complex includes user-configurable per-core counters dedicated to measuring performance-related statistics such as throughput, latency, and execution time. Both the BootROM and the CLINT controller are connected to the crossbar as AXI4 slaves with 64-bit data buses and 32-bit address buses. The AXI crossbar in each core complex connects the compute cores to the memory subsystem and accordingly routes the memory transactions. It supports burst transactions, atomic operations, and up to 16 in-flight transactions per master and per slave to scale with increasing numbers of compute cores.

The compute subsystem supports the execution of any application, enabling diverse workloads and profiling scenarios. The Toki framework includes in its open-source release a suite of user-configurable memory microbenchmarks that combine RISC-V assembly and C to implement specific access patterns, e.g., sequential, sequential with stride, and random, akin to those produced by synthetic traffic generators. Conversely, real-world applications can also be executed to profile performance in realistic scenarios. The compute subsystem is designed to support multi-threaded and non-uniformly parallel workloads with both structured and unstructured computational patterns, though the experimental analysis in this work focuses on applications from the PolyBench/C [32] 4.2 suite.

The software stack of the Toki framework notably allows selectively disabling at run time core complexes and cores without reprogramming the FPGA. This enables the straightforward profiling of configurations that are smaller than the ones physically instantiated on the FPGA. For instance, experiments on a compute subsystem configuration with four dual-core complexes can be equivalently carried out on a compute subsystem configuration with six quad-core complexes by disabling two core complexes as well as two cores of each enabled core complex.

2.2 Memory Subsystem

The memory subsystem includes the HBM with memory controllers and an AXI crossbar, as shown on the left in Fig. 3.

The HBM consists of one or more 64Gb stacks, each containing sixteen 4Gb DRAM blocks. Each block communicates with its associated memory controller via a 64-bit pseudo channel, with all sixteen pseudo channels combining to form a 1024-bit wide HBM interface. Eight memory controllers per stack handle the translation of AXI transactions into memory transactions and issue the necessary control signals to the DRAM cells. Each controller manages two pseudo channels, corresponding to a pair of DRAM blocks.

The memory subsystem connects to the compute subsystem via the AXI crossbar, that is responsible for routing memory transactions to the appropriate memory controller. An AXI4 channel is instantiated for each multi-core complex in the compute subsystem. These channels share consistent parameter settings, including data width, address width, and burst capabilities, and operate at the core clock frequency. The crossbar interfaces with the memory controllers via two 256-bit AXI3 channels per controller.

The AXI crossbar handles protocol, data width, and clock domain conversions between the compute subsystem and the memory controllers' AXI interfaces. Moreover, it allows fine-grained control over the memory address space, enabling selective access to memory regions by different core complexes. Finally, it facilitates communication between the DMA subsystem and the HBM controllers, enabling the PCIe host to read from and write to the HBM. This functionality supports uploading compiled binaries for the compute subsystem's cores and retrieving performance statistics.

2.3 DMA Subsystem

The DMA subsystem, depicted at the center of Fig. 3, facilitates the communication between the host PC and the FPGA's memory subsystem. It consists of a DMA controller, a reset controller, and a PCIe physical layer (PHY) interface.

The DMA controller enables the transfer of data between the host and the FPGA's HBM, as well as the transmission of reset commands to the reset controller, which initializes the compute subsystem. It connects to the memory subsystem through an AXI4 master interface, which is 512-bit wide and supports burst transfers and multiple in-flight transactions, and communicates with the host via the PCIe interface, connected to the FPGA's PCIe PHY layer. The host operating system (OS) orchestrates data transfers via a dedicated DMA driver that leverages scatter-gather mechanisms to optimize large transactions, thereby avoiding the need for contiguous memory allocations. The HBM is exposed to the host as two character devices, namely, one serving host-to-card read and write requests and the other handling card-to-host transfers.

The host system can generate memory access traffic to the HBM by issuing read and write requests with various patterns, such as random, sequential, and sequential with stride, via DMA over the PCIe interface. This interface connects the FPGA and its HBM memory to the rest of the system. The DMA subsystem enables byte granularity for read and write operations, allowing the host to interact with the HBM at runtime. Moreover, it supports loading software payloads for the compute cores and reinitializing the compute subsystem when switching workloads during profiling, with the reset controller generating accordingly a reset signal when triggered by a dedicated host command.

3 Experimental Setup

Hardware Setup. Our experiments targeted an AMD Alveo U55C card [2], that integrates an UltraScale+ FPGA and 16GB of HBM2, split into two 64Gb stacks each with eight memory controllers, and that connects to the host PC via a PCIe Gen3x16 interface.

The host PC is configured with an Intel i7-10700 CPU and 64GB of DDR4-2933 memory, and runs Ubuntu 22.04.5 LTS. The AMD Vivado ML 2023.2 toolchain was employed for RTL synthesis and implementation, bitstream generation, and programming of the AMD Alveo U55C card.

Software Setup. The workload executed on the CPU cores instantiated on the FPGA included a number of real-world applications from the PolyBench/C [32] 4.2 benchmark suite. The sixteen PolyBench/C applications used in the experimental campaign were compiled with gcc 13.2.0 from the RISC-V GNU compiler toolchain. Workload execution is performed bare-metal, with each core of each complex running one thread of the application under execution.

The applications' binaries are loaded at run time from the host PC through the PCIe connection into the HBM, from which instructions are fetched by the CPU cores to be executed. PCIe data transfers from the host system leverage AMD's XDMA driver to store the application binaries in the HBM, retrieve performance-related statistics from the U55C card, and stress memory with host traffic incoming through DMA over PCIe.

Toki Setup. The experimental campaign considered a variety of configurations of the Toki architecture that differed in the number of core complexes and in the number of RISC-V cores per complex. Snitch [34], a 32-bit RISC-V core that supports the RV32IMAFDC extensions and that features the OpenHW Group's CVFPU [20], is used as the compute core, configured with a 4kB instruction cache and no data cache. Synthesis and implementation targeted operating frequencies of 50MHz (clk_{core} in Fig. 5) for the compute subsystem and 250MHz (clk_{dma}) for the memory and DMA subsystems. The Toki architecture also makes use of AMD's XDMA [4] and HBM [3] IPs as its DMA and memory controllers.

Toki supports instantiating on the Alveo U55C card up to 60 Snitch cores organized into up to twelve clusters, providing a maximum memory throughput of 4.8GB/s and with a maximum resource utilization of 1 184 903 lookup tables (LUTs), 875 554 flip-flops (FFs), 900 DSPs, 820 36kB blocks of block RAM (BRAM), and 5 288kB blocks of UltraRAM (URAM). As an alternative compute core, the application-class 64-bit RISC-V CVA6 [33] core features both configurable instruction and data caches and operates at 100MHz.

Table 1: Resource utilization breakdown of Toki architecture.

Design component	kLUT	kFF	DSP	BRAM	URAM
Compute subsystem	382.0	252.4	360	288	0
→ Core complex (4×)	95.5	63.1	90	72	0
→ Snitch core (6×)	13.5	6.3	15	12	0
→ BootROM	0.3	0.1	0	0	0
→ CLINT controller	0.2	0.2	0	0	0
→ Complex AXI crossbar	14.0	25.0	0	0	0
Memory subsystem	16.0	29.0	0	6	0
→ Memory controllers	1.5	1.6	0	6	0
→ Memory AXI crossbar	14.5	27.4	0	0	0
DMA subsystem	59.6	66.0	0	112	5
→ DMA over PCIe	59.5	65.9	0	112	5
→ Reset controller	0.1	0.1	0	0	0
Total	457.6	347.4	360	406	5

Note: Toki architecture deployed on AMD Alveo U55C card with compute subsystem in $S_{6\times4}$ configuration, i.e., 24 Snitch cores organized into four core complexes.

4 Experimental Results

Without loss of generality, we showcase Toki's effectiveness with an experimental analysis that profiles the performance achievable in a worst-case scenario, where all memory accesses stress solely one memory controller of one HBM stack of the U55C card, for instance, in Fig. 3, Memory controller 1 of the 64Gb HBM stack 1. The experiments execute memory microbenchmarks and real-world applications on the compute cores and include the injection of memory traffic from the PCIe host. We refer to Toki instances with K Snitch cores per complex and P core complexes as $S_{K\times P}$. For instance, the $S_{6\times4}$ configuration features four core complexes each of whom includes six Snitch cores. Table 1 lists the post-implementation resource utilization of Toki in its $S_{6\times4}$ configuration, breaking it down among the three subsystems and their components.

Memory Microbenchmarks – Execution Time. The experimental campaign makes use of four of the memory microbenchmarks included in Toki, stressing HBM from the compute cores with sequential and random reads and sequential and random writes. The microbenchmarks involve reading and writing 256MB of data, executing 256 iterations of read and write operations on a 1MB data buffer. All memory sections dedicated to instructions and data of the application under execution are stored in HBM blocks associated to the same memory controller and each core has its own data buffer.

Fig. 4 depicts the execution time on each core of the four microbenchmarks running on a set of Toki configurations with different numbers of core complexes and cores per complex. Such metric tends in general to increase with the number of compute cores, as the latter contend access to the shared memory. The per-core execution time is almost equivalent among sequential and random reads up to twelve total compute cores, while sequential and random writes perform similarly up to eight. Conversely, once such thresholds are exceeded, the gap in performance between sequential and random accesses starts rapidly increasing, with the random accesses microbenchmark being up to 3.10× slower than the sequential one in the write use case on the $S_{6\times4}$ configuration. Write operations are notably up to 4.94× faster than read ones, with the largest gap achieved with the lowest core count, i.e., $S_{1\times1}$.

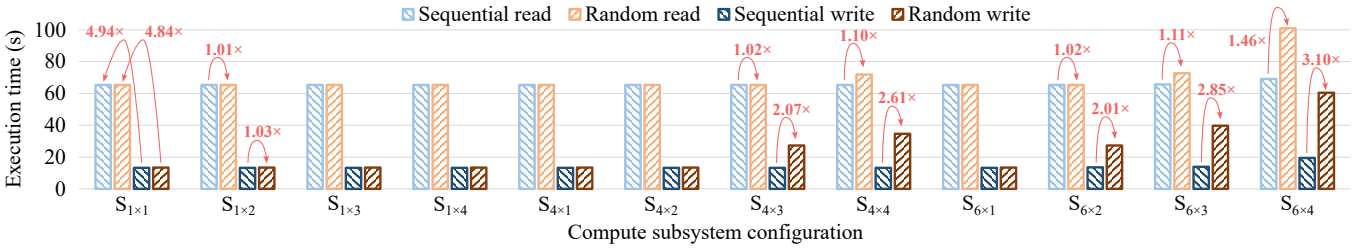


Figure 4: Execution time, expressed in seconds (s), on each compute core of memory microbenchmarks running on $S_{K\times P}$ Toki configurations with *K* Snitch cores per complex and *P* core complexes. *Note:* arrows denote performance degradation.

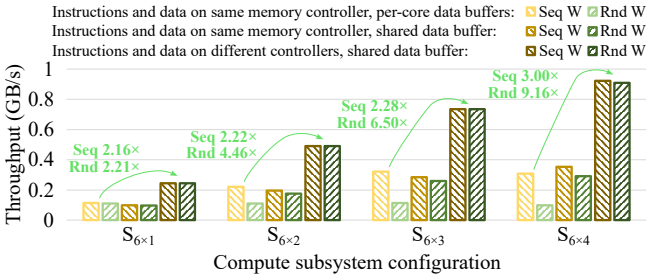


Figure 5: Memory throughput when executing sequential and random write microbenchmarks with data buffer shared among cores and with separate buffers, and with instructions and data sections mapped on same memory controller and on different ones. *Legend: Seq W* sequential write, *Rnd W* random write. *Note:* arrows denote throughput increase.

Memory Microbenchmarks – Throughput. We evaluate then how memory throughput varies by sharing first the 1MB data buffer among all the compute cores to increase the data locality, and then by moving the data buffer to memory blocks managed by a different memory controller to remove the contention between instruction and data accesses. The experiments in Fig. 5 focus, without loss of generality, on configurations with six Snitch cores per complex and a number between one and four of complexes stressed only with writes. The experiments compare the total memory throughput, expressed in GB/s, achieved first by executing the sequential and random write microbenchmarks in the baseline conditions as in Fig. 4, then sharing the 1MB data buffer among all the cores, and finally with separate memory controllers for data and instructions.

Notably, in the baseline scenario with instructions and data on the same memory controller and with per-core 1MB data buffers, the single memory controller is unable to provide a larger throughput as the core count increases with the least favorable memory access pattern, i.e., random writes. Memory throughput even slightly decreases from $S_{6\times1}$'s 0.11GB/s down to $S_{6\times4}$'s 0.10GB/s due to the controller's saturation coupled with the increasing contention.

Sharing the data buffer among the cores is shown to raise the throughput with a more meaningful improvement at higher core counts and with random accesses, e.g., the $S_{6\times4}$ configuration's random write throughput is 2.93× higher with a shared data buffer than with per-core ones. Such modification also reduces the gap between sequential and random accesses at higher core counts.

Separating instructions and data sections on different memory controllers further increases the memory throughput, with a higher improvement at higher core counts and with random accesses. For instance, such modification increases the $S_{6\times4}$ configuration's throughput by 2.61× and 3.13× for sequential and random writes, respectively, with a total improvement on the baseline scenario of 3.00× and 9.16×. In this final scenario, scaling from one to four the number of six-core complexes produces an almost linear increase in the total memory throughput, with the $S_{6\times2}$, $S_{6\times3}$, and $S_{6\times4}$ configurations having throughput values 2.0×, 3.0×, and more than 3.8× higher than the $S_{6\times1}$ one for both sequential and random writes. The improvement is indeed equal to the maximum theoretical one with two and three clusters and close to it with four.

Real-World Applications – Execution Time. Fig. 6 depicts the execution time, on the Toki $S_{1\times1}$ configuration, of each of the sixteen considered PolyBench/C applications when there is no traffic from the host system and conversely when the host system generates and sends to HBM a write-only traffic of 6GB/s via DMA over PCIe, contending the shared memory resource with the compute subsystem. When executing on one Snitch core, the execution of PolyBench/C applications takes in the order of tens to hundreds of milliseconds. When 6GB/s traffic is incoming from the host, their execution takes 2.82× on average and up to 3.57× longer than in the no-traffic scenario, with the largest slowdown occurring for the *durbin*, *bicg*, and *atax* applications.

Real-World Applications – PCIe Host Traffic. Fig. 7 depicts the relative average time for the execution on each compute core of the sixteen applications on a wide range of Toki instances, compared to the no-traffic scenario on the $S_{1\times1}$ configuration which acts as the baseline and whose bar has a value of 1× in the chart.

The injection of traffic from the host system is shown to drastically slow down the execution on the compute cores, also due to the higher throughput of 6GB/s via DMA over PCIe compared to the lower one of compute cores even when many of them are instantiated, e.g., the total throughput of compute cores is lower than 1GB/s with both optimizations applied as shown in Fig. 5.

In particular, the impact of PCIe host traffic ranges from a 1.92× increase in the average execution time of PolyBench/C applications up to a 3.83× slowdown. The latter value corresponds to a $S_{24\times1}$ compute subsystem, i.e., the configuration considered in the experimental evaluation with the highest total core count and number of cores per complex, a hierarchical organization of the compute cores that produces the highest contention of shared resources.

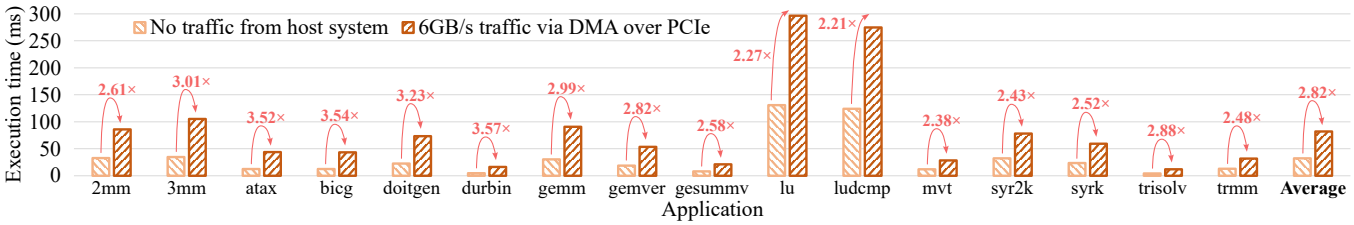


Figure 6: Execution time, expressed in milliseconds (ms), of each PolyBench/C application in $S_{1\times1}$ configuration without and with 6GB/s memory traffic from PCIe host system. *Note:* arrows denote performance degradation due to PCIe host traffic.

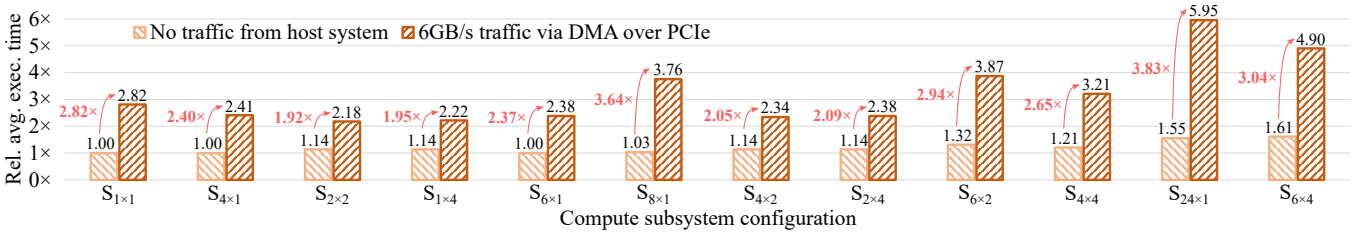


Figure 7: Relative average time for execution on each compute core of PolyBench/C applications without and with memory traffic from PCIe host system. *Note:* normalized against $S_{1\times1}$ no-traffic scenario (Fig. 6), arrows denote performance degradation.

Table 2: Feature comparison of HBM profiling frameworks.

Feature	Shuhai	Memory Sandbox	HBM Connect	DRAM Bender	Toki
Synthetic traffic generation	●	●	●	●	●
Multiple pseudo-channel support	◉	●	●	◉	●
Random access patterns	○	●	●	●	●
Programmable multi-core compute	○	○	○	●	●
RISC-V open-source cores	○	○	○	○	●
Real-world application workloads	○	○	○	○	●
Host PCIe/DMA traffic injection	○	○	○	○	●
Concurrent on-chip, host accesses	○	○	○	○	●

Legend: ● fully supported, ◉ partially supported, ○ not supported.
Note: Toki and DRAM Bender generate synthetic traffic through programmable cores, allowing reconfigurable access patterns distinct from fixed hardware generators.

5 Related Work

The existing solutions for the evaluation of HBM performance, compared with Toki in Table 2, consider synthetic scenarios with traffic generated according to some configurable memory access patterns and do not target real-world applications executing on multi- or many-core CPUs, including RISC-V-based ones. Moreover, they treat FPGAs with HBM as standalone devices rather than as part of more complex heterogeneous systems, ignoring for instance the impact given by concurrent accesses to HBM from logic instantiated on the FPGA and from the host system.

Shuhai [14] and Memory Sandbox [24] both use synthetic traffic generators to benchmark HBM2 performance on AMD's Alveo U280 card. Shuhai [14] focuses on a single HBM pseudo-channel with fixed-stride sequential read-only or write-only accesses, failing to consider coherent data traffic, random access patterns, memory accesses from the host system, and realistic workloads. Memory Sandbox [24] stresses multiple pseudo-channels, supports simultaneous accesses to the same memory region, and enables both sequential and random access patterns. HBM Connect [7] implements both the traffic generators and the interconnect using HLS, achieving a drastically lower bandwidth, and [19] also focuses on HLS-based accelerators deployed on FPGA platforms with HBM. DRAM Bender [23] employs a custom-ISA core to execute fine-grained memory tests, enabling precise control for low-level characterization analyses of HBM2 such as Rowhammer [22] vulnerability. While effective for standalone testing and isolated DRAM analysis, it overlooks performance characterization in broader system contexts.

6 Conclusions

This paper introduced Toki, the first HW/SW framework designed to profile HBM performance in FPGA-based systems with RISC-V soft cores and PCIe traffic incoming from the host via DMA. Toki's design is intentionally modular and extensible. The compute subsystem can accommodate diverse processor designs beyond Snitch and CVA6, including many-core overlay processors and soft GPGPUs. Similarly, the framework is portable across FPGA platforms with HBM support, with current validation on the AMD Alveo U55C and planned support for future generations and alternative devices.

Looking forward, Toki provides a foundation for systematic evaluation of emerging FPGA architectures, memory technologies, and processor designs. By releasing the whole framework as open source, we invite the community to extend and adapt Toki to their specific research needs and contribute new insights to the field.

Acknowledgments

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or HaDEA. Neither the European Union nor the granting authority can be held responsible for them. The authors thank the AMD University Program for providing the AMD Alveo U55C card used in this work.

References

[1] Riadh Ben Abdelhamid, Vladislav Valek, and Dirk Koch. 2024. SPARKLE: 400 RISC-V GIPS with 1,024 Barrel Processors on a single Datacenter FPGA Card. In *2024 IEEE 17th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*. 524–531. doi:10.1109/MCSoC64144.2024.00091

[2] Advanced Micro Devices, Inc. 2023. Alveo U55C Data Center Accelerator Cards Data Sheet (DS978). https://docs.amd.com/r/en-US/ds978-u55c

[3] Advanced Micro Devices, Inc. 2024. AXI High Bandwidth Memory Controller LogiCORE IP Product Guide (PG276). https://docs.amd.com/r/en-US/pg276-axi-hbm

[4] Advanced Micro Devices, Inc. 2024. DMA/Bridge Subsystem for PCI Express Product Guide (PG195). https://docs.amd.com/r/en-US/pg195-pcie-dma

[5] Chihyo Ahn, Ruobing Han, Udit Subramanya, Jisheng Zhao, Blaise Tine, and Hyesoon Kim. 2025. SoftCUDA: Running CUDA on Softcore GPU. In *2025 IEEE 33rd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 138–142. doi:10.1109/FCCM62733.2025.00049

[6] Christophe Bobda, Joel Mandebi Mbongue, Paul Chow, Mohammad Ewais, Naif Tarafdar, Juan Camilo Vega, Ken Eguro, Dirk Koch, Suranga Handagala, Miriam Leeser, Martin Herbordt, Hafsah Shahzad, Peter Hofste, Burkhard Ringlein, Jakub Szefer, Ahmed Sanaullah, and Russell Tessier. 2022. The Future of FPGA Acceleration in Datacenters and the Cloud. *ACM Trans. Reconfigurable Technol. Syst.* 15, 3, Article 34 (feb 2022), 42 pages. doi:10.1145/3506713

[7] Young-kyu Choi, Yuze Chi, Weikang Qiao, Nikola Samardzic, and Jason Cong. 2021. HBM Connect: High-Performance HLS Interconnect for FPGA HBM. In *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (Virtual Event, USA) *(FPGA '21)*. Association for Computing Machinery, New York, NY, USA, 116–126. doi:10.1145/3431920.3439301

[8] Mahmoud Ahmed Elshimy, Veronia Iskandar, Diana Goehringer, and Mohamed A. Abd El Ghany. 2023. A Near-Memory Dynamically Programmable Many-Core Overlay. In *2023 IEEE 16th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*. 268–275. doi:10.1109/MCSoC60832.2023.00047

[9] Andrea Galimberti, Gabriele Montanaro, Andrea Motta, Federico Proverbio, and Davide Zoni. 2025. A Benchmarking Platform for DDR4 Memory Performance in Data-Center-Class FPGAs. In *2025 IEEE International Symposium on Circuits and Systems (ISCAS)*. 1–5. doi:10.1109/ISCAS56072.2025.11043686

[10] Dimitrios Gourounas, Austin G. James, Bagus Hanindhito, Arash Fathi, Lizy K. John, and Andreas Gerstlauer. 2025. HighWave: Large-Scale High-Bandwidth Wave Simulations on FPGAs. In *2025 IEEE 33rd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 66–74. doi:10.1109/FCCM62733.2025.00041

[11] Jan Gray. 2016. GRVI Phalanx: A Massively Parallel RISC-V FPGA Accelerator Accelerator. In *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 17–20. doi:10.1109/FCCM.2016.12

[12] Jan Gray. 2019. 2GRVI Phalanx: A 1332-Core RISC-V RV64I Processor Cluster Array with an HBM2 High Bandwidth Memory System, and an OpenCL-like Programming Model, in a Xilinx VU37P FPGA [WIP Report]. In *2019 IEEE/ACM International Workshop on Heterogeneous High-performance Reconfigurable Computing (H2RC)*. https://h2rc.cse.sc.edu/2019/papers/lightning_2_1_Gray.pdf

[13] Yuwei Hu, Yixiao Du, Ecenur Ustun, and Zhiru Zhang. 2021. GraphLily: Accelerating Graph Linear Algebra on HBM-Equipped FPGAs. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. 1–9. doi:10.1109/ICCAD51958.2021.9643582

[14] Hongjing Huang, Zeke Wang, Jie Zhang, Zhenhao He, Chao Wu, Jun Xiao, and Gustavo Alonso. 2022. Shuhai: A Tool for Benchmarking High Bandwidth Memory on FPGAs. *IEEE Trans. Comput.* 71, 5 (2022), 1133–1144. doi:10.1109/TC.2021.3075765

[15] Veronia Iskandar, Mohamed A. Abd El Ghany, and Diana Göhringer. 2022. Near-memory Computing on FPGAs with 3D-stacked Memories: Applications, Architectures, and Optimizations. *ACM Trans. Reconfigurable Technol. Syst.* 16, 1, Article 16 (Dec. 2022), 32 pages. doi:10.1145/3547658

[16] Abhishek Kumar Jain, Chirag Ravishankar, Hossein Omidian, Sharan Kumar, Maithilee Kulkarni, Aashish Tripathi, and Dinesh Gaitonde. 2023. Modular and Lean Architecture with Elasticity for Sparse Matrix Vector Multiplication on FPGAs. In *2023 IEEE 31st Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 133–143. doi:10.1109/FCCM57271.2023.00023

[17] Chulwoo Kim, Hyun-Woo Lee, and Junyoung Song. 2016. Memory Interfaces: Past, Present, and Future. *IEEE Solid-State Circuits Magazine* 8, 2 (2016), 23–34. doi:10.1109/MSSC.2016.2546659

[18] Kexin Li, Shaoxian Xu, Zhiyuan Shao, Ran Zheng, Xiaofei Liao, and Hai Jin. 2024. ScalaBFS2: A High-performance BFS Accelerator on an HBM-enhanced FPGA Chip. *ACM Trans. Reconfigurable Technol. Syst.* 17, 2, Article 32 (April 2024), 39 pages. doi:10.1145/3650037

[19] Alec Lu, Zhenman Fang, Weihua Liu, and Lesley Shannon. 2021. Demystifying the Memory System of Modern Datacenter FPGAs for Software Programmers through Microbenchmarking. In *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (Virtual Event, USA) *(FPGA '21)*. Association for Computing Machinery, New York, NY, USA, 105–115. doi:10.1145/3431920.3439284

[20] Stefan Mach, Fabian Schuiki, Florian Zaruba, and Luca Benini. 2021. FPnew: An Open-Source Multiformat Floating-Point Unit Architecture for Energy-Proportional Transprecision Computing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 29, 4 (2021), 774–787. doi:10.1109/TVLSI.2020.3044752

[21] Gabriele Montanaro, Andrea Galimberti, and Davide Zoni. 2026. Omega: A Hardware-Software Framework for Complete Design Space Exploration of FPGA-Based Heterogeneous Multi-Core SoCs. *IEEE Trans. Comput.* 75, 2 (2026), 435–447. doi:10.1109/TC.2025.3624704

[22] Onur Mutlu and Jeremie S. Kim. 2020. RowHammer: A Retrospective. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 8 (2020), 1555–1571. doi:10.1109/TCAD.2019.2915318

[23] Ataberk Olgun, Hasan Hassan, A. Giray Yağlıkçı, Yahya Can Tuğrul, Lois Orosa, Haocong Luo, Minesh Patel, Oğuz Ergin, and Onur Mutlu. 2023. DRAM Bender: An Extensible and Versatile FPGA-Based Infrastructure to Easily Test State-of-the-Art DRAM Chips. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 42, 12 (2023), 5098–5112. doi:10.1109/TCAD.2023.3282172

[24] Elias Perdomo, Xavier Martorell, Teresa Cervero, and Behzad Salami. 2024. Memory Sandbox: A Versatile Tool for Analyzing and Optimizing HBM Performance in FPGA. In *2024 IEEE 36th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*. 206–217. doi:10.1109/SBAC-PAD63648.2024.00026

[25] Manoj B. Rajashekar, Xingyu Tian, and Zhenman Fang. 2024. HiSpMV: Hybrid Row Distribution and Vector Buffering for Imbalanced SpMV Acceleration on FPGAs. In *Proceedings of the 2024 ACM/SIGDA International Symposium on Field Programmable Gate Arrays* (Monterey, CA, USA) *(FPGA '24)*. Association for Computing Machinery, New York, NY, USA, 154–164. doi:10.1145/3626202.3637557

[26] Canberk Sönmez, Mohamed Shahawy, and Paolo Ienne. 2025. HBMex: An Attachment for Nonbursting Accelerators to Enhance HBM Performance. In *2025 IEEE 33rd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 29–37. doi:10.1109/FCCM62733.2025.00037

[27] Enhao Tang, Shun Li, Ruiqi Chen, Hao Zhou, Yuhanxiao Ma, Haoyang Zhang, Jun Yu, and Kun Wang. 2024. Graph-OPU: A Highly Flexible FPGA-Based Overlay Processor for Graph Neural Networks. *ACM Trans. Reconfigurable Technol. Syst.* 17, 4, Article 58 (Nov. 2024), 33 pages. doi:10.1145/3691636

[28] Abdul Rehman Tareen, Marius Meyer, Christian Plessl, and Tobias Kenter. 2024. HiHiSpMV: Sparse Matrix Vector Multiplication with Hierarchical Row Reductions on FPGAs with High Bandwidth Memory. In *2024 IEEE 32nd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 32–42. doi:10.1109/FCCM60383.2024.00014

[29] Zhihan Xu, Tian Ye, Rajgopal Kannan, and Viktor K. Prasanna. 2025. FAST: FPGA Acceleration of Fully Homomorphic Encryption with Efficient Bootstrapping. In *Proceedings of the 2025 ACM/SIGDA International Symposium on Field Programmable Gate Arrays* (Monterey, CA, USA) *(FPGA '25)*. Association for Computing Machinery, New York, NY, USA, 115–126. doi:10.1145/3706628.3708879

[30] Yang Yang, Rajgopal Kannan, and Viktor K. Prasanna. 2025. High Throughput Matrix Transposition on HBM-Enabled FPGAs. In *2025 IEEE 33rd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 38–46. doi:10.1109/FCCM62733.2025.00038

[31] Enxin Yi, Yiru Duan, Yinuo Bai, Kang Zhao, Zhou Jin, and Weifeng Liu. 2024. Cuper: Customized Dataflow and Perceptual Decoding for Sparse Matrix-Vector Multiplication on HBM-Equipped FPGAs. In *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 1–6. doi:10.23919/DATE58400.2024.10546672

[32] Tomofumi Yuki. 2014. Understanding PolyBench/C 3.2 Kernels. In *Proceedings of the 4th International Workshop on Polyhedral Compilation Techniques*. Vienna, Austria, 1–5. https://acohen.gitlabpages.inria.fr/impact/impact2014/papers/impact2014-yuki.pdf

[33] Florian Zaruba and Luca Benini. 2019. The Cost of Application-Class Processing: Energy and Performance Analysis of a Linux-Ready 1.7-GHz 64-Bit RISC-V Core in 22-nm FDSOI Technology. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 27, 11 (2019), 2629–2640. doi:10.1109/TVLSI.2019.2926114

[34] Florian Zaruba, Fabian Schuiki, Torsten Hoefler, and Luca Benini. 2021. Snitch: A Tiny Pseudo Dual-Issue Processor for Area and Energy Efficient Execution of Floating-Point Intensive Workloads. *IEEE Trans. Comput.* 70, 11 (2021), 1845–1860. doi:10.1109/TC.2020.3027900

[35] Shulin Zeng, Jun Liu, Guohao Dai, Xinhao Yang, Tianyu Fu, Hongyi Wang, Wenheng Ma, Hanbo Sun, Shiyao Li, Zixiao Huang, Yadong Dai, Jintao Li, Zehao Wang, Ruoyu Zhang, Kairui Wen, Xuefei Ning, and Yu Wang. 2024. FlightLLM: Efficient Large Language Model Inference with a Complete Mapping Flow on FPGAs. In *Proceedings of the 2024 ACM/SIGDA International Symposium on Field Programmable Gate Arrays* (Monterey, CA, USA) *(FPGA '24)*. Association for Computing Machinery, New York, NY, USA, 223–234. doi:10.1145/3626202.3637562

Received 18 November 2025; accepted 23 February 2026