SMAPPIC: Scalable Multi-FPGA Architecture Prototype Platform in the Cloud

Grigory Chirkov
Princeton University
Princeton, USA
gchirkov@princeton.edu

David Wentzlaff
Princeton University
Princeton, USA
wentzlaf@princeton.edu

ABSTRACT

Traditionally, architecture prototypes are built on top of FPGA infrastructure, with two associated problems. First, very large FPGAs are prohibitively expensive for most people and institutions. Second, the burden of FPGA development adds to an already uneasy life of researchers, especially those who focus on software. Large designs that do not fit into a single FPGA exacerbate these issues even more. This work presents SMAPPIC – the first open-source prototype platform for shared memory multi-die architectures on cloud FPGAs. SMAPPIC leverages the OpenPiton/BYOC infrastructure and AWS F1 instances to make FPGA-based prototypes of System-on-Chips, processor cores, accelerators, cache subsystems, etc., cheap, scalable, and straightforward. SMAPPIC enables many use cases that are not possible or significantly more complicated in existing software and FPGA tools. This work has the potential to accelerate the rate of innovation in computer engineering fields in the nearest future.

CCS CONCEPTS

• **Hardware** → **Simulation and emulation**; **Reconfigurable logic and FPGAs**; • **Computer systems organization** → *Multicore architectures*; *Heterogeneous (hybrid) systems*.

KEYWORDS

Modeling, FPGA, cloud, multi-die, interconnect, multicore, heterogeneity

ACM Reference Format:
Grigory Chirkov and David Wentzlaff. 2023. SMAPPIC: Scalable Multi-FPGA Architecture Prototype Platform in the Cloud. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '23), March 25–29, 2023, Vancouver, BC, Canada.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3575693.3575753

1 INTRODUCTION

The approaching demise of Moore's Law [25] puts the task of optimizing computation on the shoulders of system-level researchers: computer architects, compiler engineers, operating system and runtime developers, etc. Correspondingly, the number of papers in the top venues in these fields has increased dramatically over the last 20 years [1–3]. One can imagine a future where all the advances in computation come purely from clever new techniques and features designed by these researchers. However, most of such work demands working on futuristic, currently nonexistent hardware. This highlights the importance of a cheap, fast, scalable, and easy-to-use prototyping infrastructure.

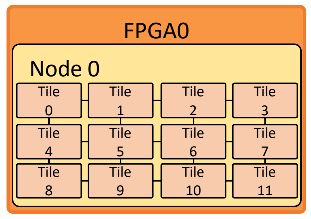
Traditionally, architecture prototypes are built on top of Field-Programmable Gate Array (FPGA) infrastructure. This approach has two significant associated problems. First, FPGA development is a complicated task that requires deep knowledge of not only Register Transfer Level (RTL) but also the details and physical constraints of the chosen FPGA platform. Many researchers (especially those who focus more on software aspects) would rather spend their time and effort on something closer to their interests. Second, FPGAs needed for large systems' prototypes must be bought by researchers for a significant price (starting from $5000 or $20000 for a set of four) [62]. Therefore, independent researchers cannot afford to use FPGAs at all, and large entities cannot afford to scale out FPGA infrastructure, i.e., evaluate multiple systems in parallel or use multiple FPGAs for a single prototype.

Moreover, the end of Moore's Law gradually slows down the transistor density scaling and forces designers to implement increasingly larger designs each year [33]. Correspondingly, prototypes are also becoming larger and no longer fit into a single FPGA. Partitioning prototypes across multiple FPGAs further exacerbates both price and complexity issues. Emulation systems like Synopsys Zebu [55] solve the complexity problem but only at the cost of even more expensive hardware.

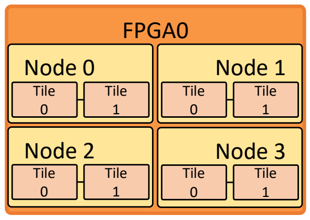
The current COVID-19 pandemic makes FPGAs even less accessible. Working with an FPGA often requires physical access to the hardware, which can be unavailable due to pandemic restrictions like lockdowns and Work-From-Home orders. Moreover, due to the pandemic-induced chip shortage, purchasing FPGAs is challenging even with the necessary funds. For example, lead times for large FPGAs from Xilinx now exceed half a year [62].

However, FPGAs have debuted as offerings from multiple public cloud providers in the past five years, including Amazon Web Services (AWS) [19, 32, 43]. This development presents a unique opportunity for researchers to embrace FPGA prototyping or migrate their flow from existing on-premise solutions to cloud FPGAs, enabling them to run thousands of instances in parallel or to distribute large designs across multiple FPGAs for a modest price. Such a prototype also presents the opportunity to use large datasets available only in the cloud or have an FPGA prototype interacting with cloud-only services like complex Function-as-a-Service (FaaS) pipelines. Computer architecture, operating systems, and compilers

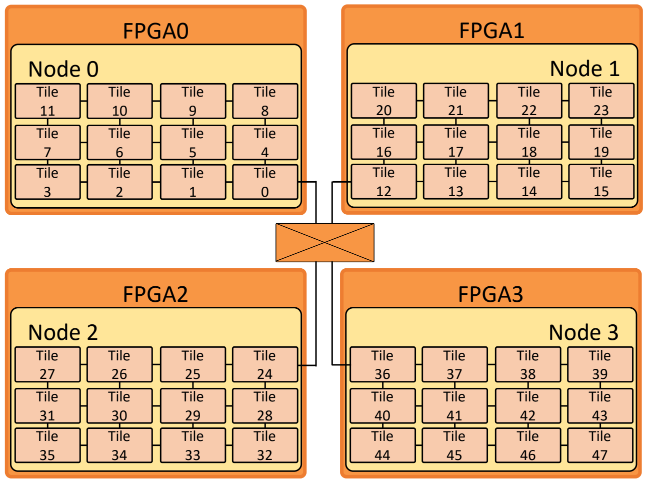
ASPLOS '23, March 25–29, 2023, Vancouver, BC, Canada
© 2023 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9916-6/23/03.
https://doi.org/10.1145/3575693.3575753



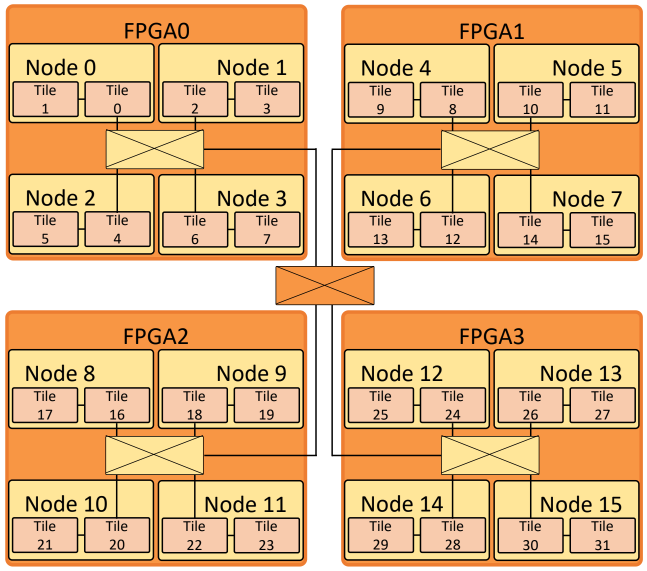
(a) Classic 1x1x12 configuration for single-node targets.



(b) Cost-efficient 1x4x2 configuration for small targets.



(c) 4x1x12 configuration for multi-node targets with large nodes.



(d) 4x4x2 configuration for multi-node targets with small nodes.

Figure 1: SMAPPIC generates optimized prototypes for various targets. Configurations are denoted in AxBxC format, where A is the number of FPGAs, B is the number of nodes per FPGA, and C is the number of tiles per node.

classes can also use cloud FPGA platforms' on-demand scale-out nature for educational purposes beyond what a single academic institution can purchase.

This paper presents Scalable Multi-FPGA Architecture Prototype Platform in the Cloud (SMAPPIC) – the first open-source[1] prototype platform for shared memory multi-die architectures on cloud FPGAs. SMAPPIC is designed with three main goals: ease of use, cost-effectiveness, and scalability. They all are achieved through a modular, hierarchical, and parametrizable structure based on a cloud FPGA backend.

SMAPPIC is a ready-to-use solution that makes architectural FPGA prototyping easy and accessible to researchers from various computer science and computer engineering fields. Researchers do not need to be experts in FPGA development to implement their prototypes, and the modular design of SMAPPIC allows them to make RTL modifications with non-invasive changes. Moreover, many software studies can be performed by tuning built-in platform parameters without the need to change RTL code altogether. SMAPPIC's cloud FPGA backend makes it accessible and cost-effective, allowing users to pay for infrastructure per hour.

Each prototype in SMAPPIC contains one or more separate nodes, where each node represents a single chip or die of the target system. Nodes can be distributed across multiple target FPGAs; they can be independent and represent separate systems or be merged with newly designed inter-node interconnect into one large system. SMAPPIC configurations are described in AxBxC notation, where A is the number of FPGAs in the prototype, B is the number of nodes per FPGA, and C is the number of tiles per node. Fig. 1 shows the example SMAPPIC hierarchies:

(1) Fig. 1a shows a classic 1x1x12 configuration with a single prototype that implements one node in one FPGA.
(2) Fig. 1b shows a cost-efficient 1x4x2 configuration. Such configuration allows modeling multiple systems with different parameters in one FPGA to optimize utilization and cost-efficiency.
(3) Fig. 1c shows a 4x1x12 configuration. It represents a cache-coherent multi-node system with 4 large 12-core nodes.
(4) Fig. 1d shows a 4x4x2 configuration. It represents a cache-coherent multi-node system with 16 small 2-core nodes.

We chose the BYOC/OpenPiton [10, 11] framework as SMAPPIC's node foundation and AWS EC2 F1 instances as a target FPGA infrastructure. Together, they provide all the necessary features needed to achieve our design goals:

(1) The BYOC framework provides a modular and configurable RTL model of a heterogeneous computing platform with Network-On-Chip (NoC) interconnect and scalable coherent cache subsystem inside each node. Various research groups have already integrated Ariane [65], OpenSPARC T1 [37], ao486 [39], PicoRV32 [60], AnyCore [17], BlackParrot [41] cores as well as NVDLA [34] and MIAOW GPU [8] accelerators into BYOC. These computational cores can be used out of the box without any modifications. Moreover, the Transaction Response Interface (TRI) allows for fast and easy integration of other new computation cores.

[1]Source code: https://parallel.princeton.edu

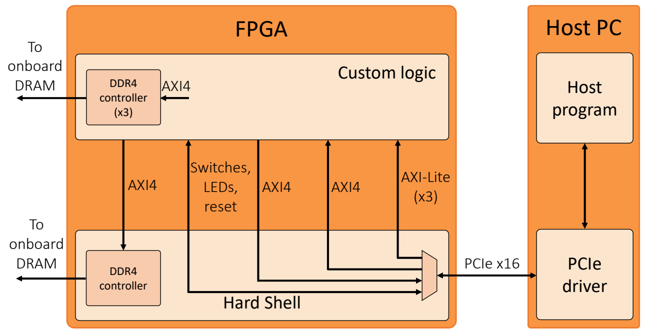


Figure 2: F1 instances organization. F1 FPGA contains two partitions: Custom Logic (CL) and Hard Shell (HS). Users can fully customize CL; HS is fixed and provided by AWS. All communication with the host happens through HS, where requests and responses are converted from the host-FPGA PCIe connection to AXI4/AXI-Lite interfaces and back.

(2) AWS EC2 F1 instances provide infrastructure for cheap, scalable, on-demand FPGA prototyping. Four independent DRAM interfaces in F1 make it possible to pack up to four nodes into one FPGA. At the same time, the F1 instance's ability to perform direct FPGA-to-FPGA communication through the PCIe interface makes SMAPPIC's coherent inter-node interconnect possible.

There are many unique use cases that are impossible or significantly harder without SMAPPIC. We discuss some of them in this paper, including large-scale multi-node architecture modeling, accelerator evaluation and verification, hardware/software co-development, *in situ* studies of a custom architecture interaction with a cloud infrastructure, cost-efficient architecture modeling, remote work, education, and more.

Our contributions include:

- The first open-source prototype platform for shared memory multi-node architectures on cloud FPGAs
- The first open-source RISC-V multi-node system with unified memory
- The first open-source 48-core 64-bit RISC-V system with coherent memory
- The first demonstration of full-stack Linux running on a 48-core RISC-V system in NUMA mode
- The first comparison of architecture modeling methods in the cloud with respect to cost-efficiency
- Demonstration of custom prototype interacting with public cloud services

2 BACKGROUND

2.1 AWS EC2 F1 Infrastructure

AWS debuted EC2 F1 instances as an option in its public cloud in November 2016 as a way to offer its customers customizable acceleration capabilities. AWS currently offers three F1 instances: f1.2xlarge, f1.4xlarge, and f1.16xlarge; their parameters and prices are listed in Table 1. Overall, users can get up to 8 Xilinx VU9P [64] FPGAs inside one instance. Each hour of a single FPGA usage costs $1.65.

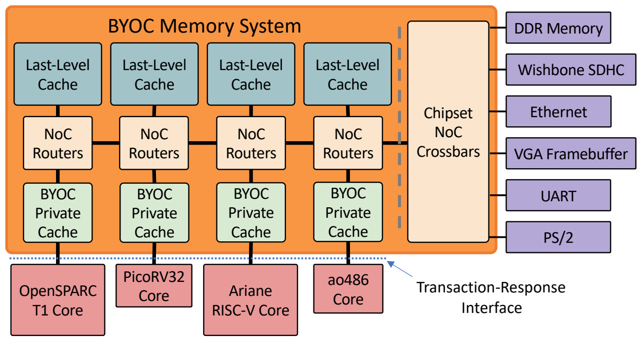


Figure 3: BYOC principal diagram (reproduced from [10]). BYOC has a modular open-source design, configurable cache subsystem, and a wide range of integrated cores and ISAs.

Fig. 2 shows a logical representation of F1 instances [42]. FPGAs in F1 are divided into two parts: 1) A Hard Shell (HS) provided by AWS that must be included in the final FPGA image, and 2) Custom Logic (CL), where developers have complete control over instantiated logic. Developers can access four DDR4 memory controllers and three AXI-Lite interfaces for logic configuration and management per FPGA. Data movement to/from the instance is performed through two AXI4 interfaces: one inbound and one outbound. The HS converts these AXI4 signals to/from PCIe Gen3 x16 connections. Depending on the target address, the outbound AXI4 request is routed to one of the FPGAs connected to the host or to the host itself.

The described setup is easy to recreate locally, given sufficient funds. Similar boards can be bought directly from Xilinx [62] or other vendors [12]. HS uses Xilinx XDMA IP under the hood, which is provided with each copy of Xilinx Vivado tools [63]. However, we estimate that the upfront cost of a similar system with a single FPGA (including server, FPGA, and FPGA memory) is about $8000, which can be too large of an investment for many researchers, especially if sporadically used.

2.2 BYOC and OpenPiton

BYOC [10] is a hardware framework for heterogeneous-ISA research. It supports the connection of cores with different ISAs and microarchitectures as well as accelerator integration. The high-level design of BYOC is shown in Fig. 3.

BYOC is built on top of OpenPiton [9] framework and Piton processor [31] and provides a modern, tiled manycore design with a cache coherence protocol and a Network-on-Chip (NoC) interconnect scalable to up to 500 million cores. BYOC's cache subsystem is configurable and allows choosing different cache sizes and associativities. Users can select one of the ten provided core models. Notably, the framework already integrates Ariane [65], AnyCore [17], BlackParrot [41], OpenSparcT1 [37], PicoRV32 [60], and ao486 [39] cores. If this choice is not enough for the users, they can choose to connect their design using the Transaction Response Interface (TRI).

TRI is the gateway between a core and the central part of BYOC: its memory subsystem (see Fig. 3). This interface and BYOC Private Cache (BPC) were designed to isolate cores from details of

Table 1: Available AWS EC2 F1 instances [44].

Instance	f1.2xl	f1.4xl	f1.16xl
#vCPUs	8	16	64
Host Memory	122 GB	244 GB	976 GB
Storage	470 GB	940 GB	3760 GB
#FPGAs	1	2	8
FPGA Memory	64 GB	128 GB	512 GB
Pricing	\$1.65/hr	\$3.30/hr	\$13.20/hr
Hardware price	≈ \$8000	≈ \$16000	≈ \$64000

the coherence protocol in the underlying cache subsystem, thereby simplifying the integration of new compute units (cores) into BYOC. BPC can work as either a private L1 or L2 cache. BPCs are interconnected with each other and slices of distributed shared last-level cache (LLC) through three NoCs. The LLC also uses the NoCs to interact with the memory controller and peripherals, all located inside the chipset.

The BYOC framework has an open-source codebase and is written in Verilog and SystemVerilog. This modular and configurable design makes it an excellent platform for experimenting and developing new hardware/software features; SMAPPIC leverages it to provide the same features in its nodes. At the same time, BYOC provides only limited FPGA prototyping capabilities without multi-node support.

3 ARCHITECTURE

A SMAPPIC prototype consists of one or several nodes; each node represents a single chip or die of the target system. Nodes can be independent of each other and implement separate instances of the prototyped design, or they can be connected with SMAPPIC's specially designed inter-node interconnect to build a prototype of a single large shared memory multi-node system. We measured the inter-FPGA round-trip latency on the PCIe bus in an F1 instance to be about 1250 nanoseconds. At a typical FPGA frequency of about 100 MHz, this latency equals 125 cycles, which is the same order of magnitude as the inter-socket latency in a typical multi-socket system. Therefore, a multi-node system can be modeled either by putting all nodes into one FPGA or by distributing nodes across multiple FPGAs.

The nodes in our implementation are separate BYOC instances. However, the inter-node interconnect does not rely on any fundamental properties of BYOC, and BYOC can be replaced with any node model with multi-node capabilities. We chose BYOC because it strengthens SMAPPIC's advantages: modularity, configurability, and ease of use.

3.1 Inter-node Interconnect

SMAPPIC's inter-node interconnect is designed to make large-scale multi-node prototypes possible. It binds together nodes located in the same FPGA and different FPGAs. To achieve this, the interconnect encapsulates traffic between nodes into AXI4 write requests. This solution allows connecting nodes on the same FPGA using the AXI4 crossbar and nodes on different FPGAs using the AXI4-PCIe transducer provided in Hard Shell. The encapsulation does not

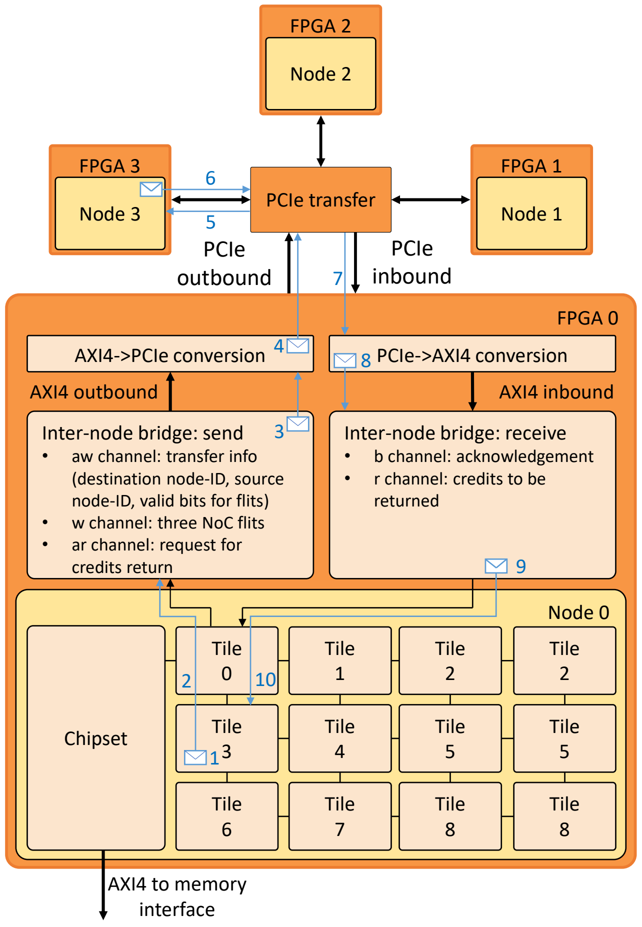


Figure 4: SMAPPIC in 4x1x12 configuration. Inter-node NoC packets are first routed towards Tile 0, then pushed out in the northbound direction into the inter-node bridge, where they are encapsulated into AXI4 requests and tunneled in the PCIe bus to other nodes in other FPGAs.

change the traffic and does not significantly rely on packet structure, and the BYOC can be replaced with any other node model with inter-node capabilities fairly easily.

Fig. 4 shows the high-level design of SMAPPIC in a 4x1x12 configuration. Each node is located in its own FPGA and communicates with other nodes through inbound and outbound AXI4 interfaces. We will go through an example of what happens during a BPC cache miss to show the functionality of SMAPPIC. Blue numbers and lines in Fig. 4 schematically show the sequence of events and NoC communication.

- Stage 1: a core on node 0 in tile 3 does a memory read that causes L1 and BPC cache misses. BPC issues a read request to the cache line's home LLC slice in this case. The homing mechanism in BPC decides where the cache line's home is. The original BYOC design supports multi-chip configuration through a special hardware/software mechanism called Coherence Domain Restriction [22]. Unlike BYOC, SMAPPIC's

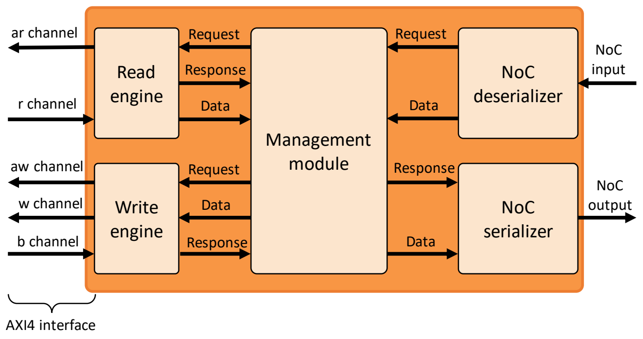


Figure 5: NoC-AXI4 bridge transduces requests from BYOC NoC protocol to AXI4 protocol. The bridge buffers requests for non-blocking operation and aligns requests to a 64-byte boundary to conform to AXI4 protocol requirements.

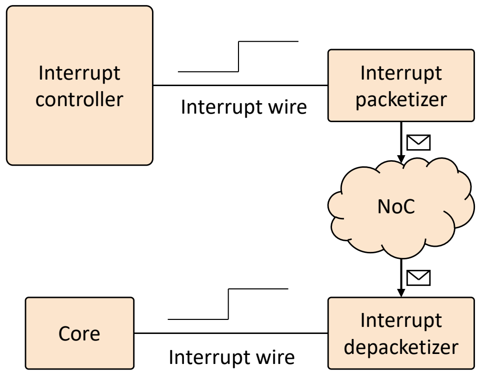


Figure 6: Interrupt packetizer and depacketizer enable sending interrupts across node boundaries.

homing mechanism was changed to distribute cache lines across all nodes in the system and work out of the box without software support. The read is local if the cache line's home slice is located in the same node. However, in our example, the core is in node 0, and the home slice is in node 3, making this request inter-node.

- Stage 2: NoC routers are programmed to route inter-node packets into tile 0, then in the northbound direction, where they leave the node and enter the inter-node bridge specifically designed to encode NoC packets into outgoing AXI4 requests and vice versa.
- Stage 3: A NoC packet is encapsulated into AXI4 write requests. The request's address encodes the destination node ID, source node ID, and valid bits for flits encoded in write data. To guarantee the absence of deadlocks in the system, the NoCs must have credit-based flow controls. To return credits, the sending side periodically issues an AXI4 read request to the receiving side and gets the number of returned credits in response.
- Stages 4-5: The request is converted into PCIe transfer inside HS and sent to the other FPGA. This PCIe traffic goes directly from FPGA to FPGA and does not use the host CPU.
- Stages 6-7: The response from the home slice is sent back to node 0 in the form of another PCIe transfer.
- Stage 8: The PCIe transfer is converted into an AXI4 request inside HS.
- Stage 9: An AXI4 request is decoded inside the inter-node bridge. The valid bits are taken from the request address, and NoC flits are taken from the request data. The request's address also contains the source node-ID, which is used for credit accounting. If the inter-node bridge gets an AXI4 read request, it returns the credits in response.
- Stage 10: Response is routed back to the original core in tile 3 on node 0.

3.2 Memory Interfaces

The F1 infrastructure provides developers access to up to four DRAM interfaces. Developers are expected to use AXI4 interfaces for read and write requests, and BYOC's original memory controller is incompatible with this protocol.

To manipulate the memory interfaces, we designed a NoC-AXI4 memory controller that transduces requests from the native NoC protocol to the AXI4 interface and sends responses back via the NoC to the requesting tile. Its high-level design is shown in Fig. 5. Incoming requests from the NoC are first deserialized and then forwarded to the management module. There, requests are buffered to enable non-blocking functionality and higher throughput. Requests are then steered into the corresponding engine: memory reads into the read engine and memory writes into the write engine. The engines assign an AXI4-ID to each request and save the Miss Status Handling Register (MSHR), the ID-MSHR mapping, the origin of the request in the system, and other miscellaneous information. The request's address and data (in the case of a write) are then aligned to a 64-byte boundary to fulfill the AXI4 protocol requirements. Upon response arrival, ID-MSHR mapping is used to restore the initial request's MSHR. If it is a read response and the original request is smaller than 64 bytes, the necessary portion of bytes is selected. The response and data (in case of a read) are then pushed into the NoC serializer, after which they leave the memory controller.

3.3 RISC-V Interrupt Controller

Another essential part of SMAPPIC is its RISC-V interrupt controller. We want to make the transition to our platform as seamless for researchers as possible. With the growing popularity of RISC-V in the community, it was important for us to have this functionality available out of the box.

In the RISC-V specification [5], cores are notified about pending interrupts by asserting a wire going straight from the interrupt controller into the core. This specification lacks scalability because a) in the case of a manycore system, a wire goes across the whole node and uses significant routing resources and affects the timing of the design, and b) in the case of a multi-node system, it is difficult or impossible to route a wire across node boundary.

To deal with this problem, we designed an interrupt packetizer and depacketizer. Fig. 6 shows their functionality. The interrupt packetizer scans the interrupt controller's outputs, and when they

Table 2: Prototyped System Parameters

Instruction set	RISC-V 64-bit
Operating system	Linux v5.12
Frequency	100 MHz
Core	Ariane
Core pipeline	In-order, 6 stages
Branch history table entries	128
ITLB entries	16
DTLB entries	16
L1D cache	8 KB, 4 ways
L1I cache	16 KB, 4 ways
BPC cache	8 KB, 4 ways
LLC cache slice	64 KB, 4 ways
DRAM latency	80 cycles
Inter-node round-trip latency	125

change, it notifies the appropriate core about the change by sending a NoC packet. On the core side, the interrupt depacketizer sniffs the network traffic, and when an interrupt packet arrives, the interrupt wires are asserted or deasserted depending on the packet contents.

3.4 I/O Interfaces

3.4.1 UART and Serial Interfaces. UART is an essential interface for SMAPPIC: it is used for all the external interactions like loading tests into memory and console I/O operations. However, the F1 infrastructure does not provide a UART interface to the developer.

To tackle this issue, we encapsulate UART into the AXI-lite protocol and use one of the available AXI-Lite interfaces in F1 to further tunnel data from the FPGA to the host. The UART to AXI-Lite conversion is done with a Xilinx UART16550 IP [61] included with Xilinx Vivado. Further inside the host system, we developed a new program that creates a virtual serial device and tunnels data from the PCIe driver into this virtual serial device.

We instantiate two UART devices inside each BYOC instance: one with the standard baud rate of 115200 bits/second for console interactions/instance management and one with an "overclocked" transmission rate of around 1 Mbit/second for data transmission. We used the overclocked device to connect the prototype to the Internet using the standard modem connection utility *pppd*.

3.4.2 Virtual SD Card. Like many datacenter FPGAs, the F1 FPGA does not have an SD card slot. However, BYOC requires an SD card controller to be able to provide a filesystem and run Linux. To solve this problem, we introduce the notion of a "Virtual device" into SMAPPIC. We call a device virtual if it does not physically exist in the system; all requests to this device are instead forwarded into the SMAPPIC system's main memory. Virtual devices provide only the functionality of the original device and do not model performance properly.

Since F1 lacks an SD card slot, we make the device virtual and map it into the top half of the FPGA's DRAM; the remaining bottom half is used for the prototype's main memory. For SD card initialization, we wrote a specialized Linux driver that is run on the host system. It performs the writes to the addresses inside the FPGA's PCIe address space, and these writes eventually appear on the FPGA's inbound AXI4 bus. This way, we can inject NoC flits inside the system. In particular, we can inject NoC packets that target the prototype's memory controller and perform writes in the SD card region of memory.

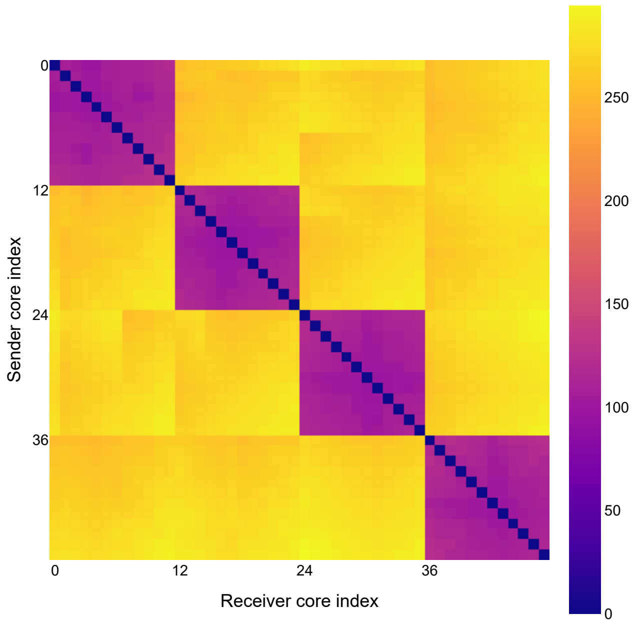


Figure 7: Inter-core latencies in multi-node prototype in cycles. There are four clearly visible NUMA domains in the system. Round-trip latency inside one domain is about 100 cycles, and across domains – 250 cycles, 2.5 times higher.

3.5 Modeling Off-node Interfaces

The node's interaction with the outside world cannot be mapped into FPGA gates. This includes inter-node communication, memory operations, input/output (I/O) operations, network operations, etc. To combat this modeling challenge, we include a traffic shaper with configurable bandwidth and latency in the inter-node bridge and memory controller. This solution provides performance models of interconnect and memory interfaces in addition to the functional models described above. I/O operations are modeled only functionally in SMAPPIC.

4 SMAPPIC USE CASES

This section describes our vision of how SMAPPIC can be employed for various problems.

4.1 Large-Scale Multi-node Architecture Modeling

In our first example, we build an FPGA prototype of a 64-bit cache-coherent RISC-V multi-node system with a total of 48 cores. Using this prototype, we run a full-stack Linux operating system and evaluate its' implementation of NUMA mode in the RISC-V architecture.

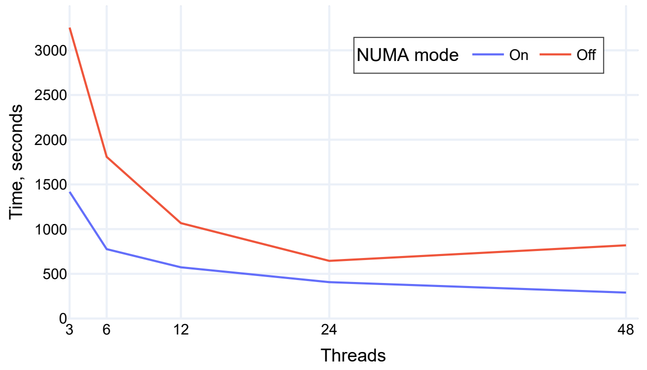


Figure 8: The difference in performance between NUMA-aware and non-NUMA-aware Linux kernel running multi-threaded integer sorting benchmark. NUMA mode reduces runtimes by 1.6-2.8 times. The effect is especially strong with a high thread count because of increased inter-node communication in non-NUMA mode.

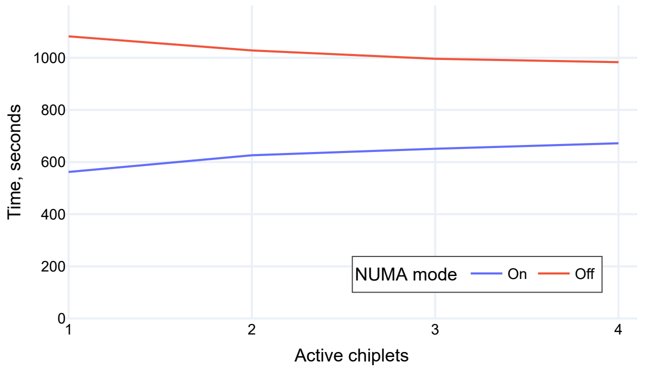


Figure 9: Thread allocation effect on integer sorting benchmark runtime. The number of threads is fixed to 12; they are distributed between either 1, 2, 3, or 4 active nodes. In NUMA mode, more active nodes mean increased memory access latency and slightly higher runtime. With NUMA mode disabled, more active nodes mean a higher chance of thread and data collocation on the same node and slightly better performance.

We use the integrated Ariane core in this example. SMAPPIC makes synthesizing the FPGA prototype straightforward: the user simply specifies the preferred core type, the number of tiles per node, the number of ndoes per FPGA, and the number of FPGAs. As an example, we create a prototype with a 4x1x12 configuration: four FPGAs, one node per FPGA, and 12 cores per node for a total of 48 Ariane cores. Fig. 2 shows the parameters of the prototyped system. The FPGA image generation takes about 2 hours on a desktop machine equipped with an Intel Core-i9 9900K CPU and requires about 32 GB of memory. Final postprocessing is done by AWS in their datacenter and takes another 2 hours. Loading the bitstream into FPGA takes about 10 seconds.

As a part of this case study, we run the full-stack Linux kernel in the generated prototype. The Linux kernel works out of the box in SMAPPIC. Non-uniform memory access (NUMA) mode has been available for RISC-V platforms since release 5.12, and since our setup has NUMA properties, we enabled it in the kernel configuration. The software reads NUMA parameters from the device tree during the boot process.

The first metric of interest for us is the communication latency between cores. Fig. 7 shows the heatmap of round-trip latencies between different cores in the system in cycles. The locations of the four NUMA nodes are clearly visible in this chart: cores on the same node have round-trip latency of about 100 cycles, and round-trip latency for cores on different nodes is about 250 cycles, 2.5 times higher than intra-node. This number is roughly the same for multi-socket Intel server platforms [21], which validates our idea of modeling multi-node systems with multi-FPGA setups. The inter-node link latency can be adjusted to represent systems with a slower interconnect, e.g., Ampere Altra [4].

We then evaluate the real-world performance of our prototype with an integer sorting benchmark from the NAS parallel benchmark (NPB) suite [7]. The benchmark uses a parallel bucket sorting algorithm to sort a 134-million-long array of integers (NPB's class C input size). Fig. 8 shows the performance scaling with the number of cores with Linux NUMA mode on versus off. NUMA mode reduces runtime by 1.6-2.8 times depending on the number of threads. The difference is especially large with a high thread count; memory allocation becomes even more critical in this situation because of the large volume of inter-node communication and associated congestion. We also compared the scaling characteristics of the system with a similar multi-socket server setup from Intel. We saw a similar dynamic going from low thread count to full system usage.

Fig. 9 shows the same benchmark run in the same system, but this time we fix the number of threads to 12 and pin them to either 1, 2, 3, or 4 nodes using the *taskset* utility to study how much sub-optimal thread allocation hurts performance. Distributing threads among more nodes in NUMA mode leads to higher average memory access latency and slightly higher runtime. Interestingly, with NUMA mode disabled, the situation is reversed, and the performance slightly increases when cores are spaced further apart. We think that this happens because, with more nodes involved, there is a higher chance that at least some of the threads and application data are collocated on the same node.

The studies shown in this section are not possible on any simulator or emulator currently available to researchers. Only the SMAPPIC prototype can model benchmark runs in a large NUMA system running full-stack Linux at speeds around 100 MHz.

4.2 Accelerator Verification and Evaluation

SMAPPIC is a valuable framework for accelerator development. It includes tools for easy accelerator integration into the node's architecture. Users can employ provided coherence tools to enable fine-grained interaction between the accelerator and the rest of the system. Superior speeds of FPGA emulation allow evaluating and verifying the accelerator with much larger input datasets than possible with purely software tools.

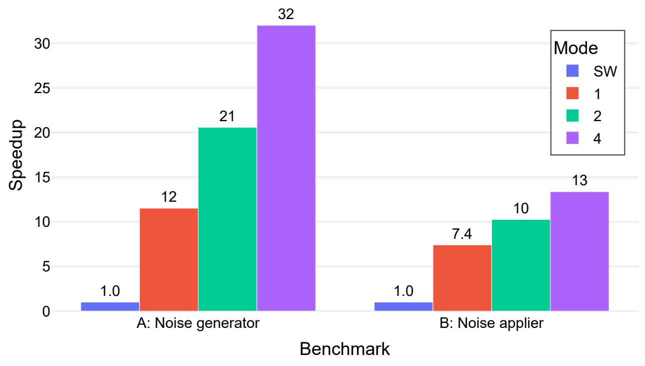


Figure 10: Performance evaluation of the GNG accelerator in SMAPPIC. The speedup is calculated relative to the software implementation. Four execution modes are compared: software (SW), 1 number per fetch (1), 2 numbers per fetch (2), and 4 numbers per fetch (4). The hardware implementation works much faster, especially with combined fetches.

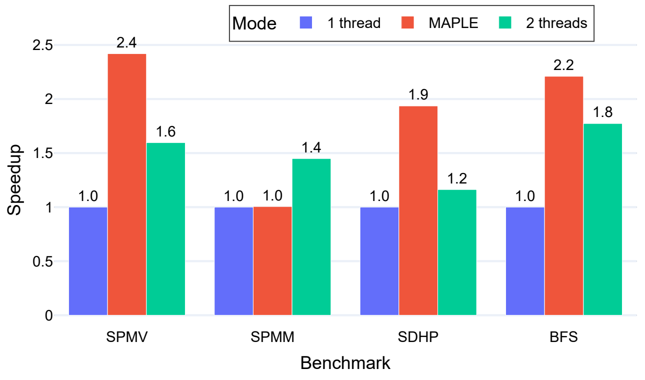


Figure 11: Performance evaluation of MAPLE engine in SMAPPIC. Speedup is calculated relative to single-thread execution. Three execution modes are shown: single-thread, single-thread with MAPLE engine, and 2-thread.

As an example, we built a prototype of a CPU with an integrated Gaussian Noise Generator (GNG) accelerator [28] from the OpenCores project [36]. Gaussian noise is widely used in communication channel testing, molecular dynamics simulation, and financial modeling [27]. The accelerator generates noise using the random numbers from the Tausworthe Generator [59]. It took around 1.5 hours of a graduate student's work to do the initial integration and debugging. We use a SMAPPIC 1x1x2 configuration for simplicity, but users can quickly scale this prototype by only changing command line options passed to build scripts. Tile 0 contains the Ariane core, and tile 1 contains the GNG.

To fetch a new number from the GNG, Ariane issues a non-cacheable load to the accelerator's memory address. We develop and study two integration schemes: base and optimized. In the base scheme, each non-cacheable load returns a single 16-bit number. In the optimized scheme, the load returns two or four 16-bit numbers packed into one 32-bit or 64-bit integer respectively. The optimization replaces multiple fetch operations with only one and reduces the number of needed loads.

After the prototype is built, users can easily evaluate and verify the accelerator with SMAPPIC's built-in machinery. For illustrative purposes, we compare the accelerator's performance with the software implementation executed in Ariane. We execute two benchmarks on top of Linux. Benchmark A ("Noise generator") generates 64 MB of noise to compare software and hardware GNG implementations. Benchmark B ("Noise applier") converts generated noise into 8-bit integers and applies it to a 32 MB long sequence to compare two implementations in a real-life scenario.

Fig. 10 shows performance results. The accelerator significantly decreases benchmarks' execution times, especially when it combines multiple number fetches into one request. The performance difference is smaller in benchmark B because less of the execution time is accelerated here. The whole process of accelerator evaluation took about one graduate student's workday from start to end, including accelerator integration, debugging, optimization, synthesizing FPGA images, developing accelerator software, and running benchmarks.

4.3 Hardware/Software Co-development

Many works on programming languages and operating systems propose ideas based on slight architectural changes to existing hardware [22, 48, 66]. Researchers from these fields want to be able to run full-stack operating systems and be able to interact with the hardware in real time. Due to its scalability, speed, cost, configurability, and full-stack SMP Linux support, SMAPPIC is a perfect tool for such work.

To demonstrate this, we reevaluate the MAPLE [38] engine in SMAPPIC and compare it with parallel 2-thread software execution. MAPLE is an accelerator for Decoupled Access Execute programs [49]. The Execute part runs in the general-purpose core, while the Access part is offloaded to MAPLE. MAPLE is programmed before the execution starts to asynchronously fetch the data from memory and supply it to the Execute core right when needed. This approach grants substantial performance gains in applications with irregular memory access patterns through fine-grained software/hardware interaction.

We use SMAPPIC's 1x1x6 configuration with Ariane cores in tiles 0, 1, 4, 5, and MAPLE engines in tiles 2, 3. The benchmarks and datasets are the same as in the original work. MAPLE's code and integration with OpenPiton are open-sourced, so the engine works in SMAPPIC out of the box. We would like to note that integrating even such delicate mechanisms in SMAPPIC is easy and takes only about a hundred lines of Verilog code [38].

Fig. 11 shows MAPLE's performance results. The evaluation shows that MAPLE is more efficient than the second thread in latency-bound applications. Unsurprisingly, SMAPPIC produces charts identical to the ones described in the original MAPLE work because the modeled hardware is very similar. However, this case study is interesting in another aspect.

The test execution would often hang the whole system during our initial testing. After a couple of hours of debugging, we noticed

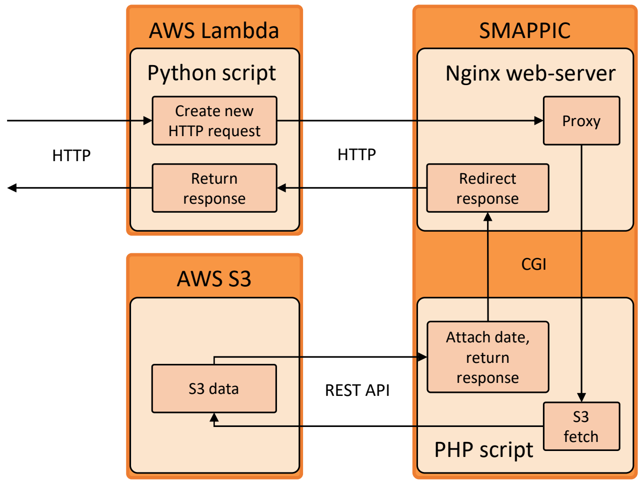


Figure 12: SMAPPIC in an experimental cloud pipeline. SMAPPIC's cloud nature allows for *in situ* cloud studies of custom architecture's interaction with other datacenter services.

that the problem disappeared when each program's thread was pinned to some core. Eventually, we found a piece of Verilog code in MAPLE's design that memorizes the core ID at the execution start and uses this information later for some operations. This design is problematic for running programs on top of the operating system because of the thread migration.

Interestingly, MAPLE's designers had not seen this problem before we contacted them. Even though they used FPGA for testing and executing tests on top of Linux as well, their FPGA was small and could fit only two Ariane cores inside. This case provides yet another reason why SMAPPIC helps so much with design verification. The ability to execute tests in **large** prototypes on top of the operating system at high frequencies is currently provided in only our open-source tool.

4.4 *In Situ* Studies of Custom Architecture Interaction with Cloud Infrastructure

While there are many ways to model new architectures, modeling their interactions with existing cloud infrastructure can be challenging. Researchers that study custom cloud architectures would greatly benefit from the ability to integrate the new hardware directly into a datacenter setting. SMAPPIC makes this task a lot easier because the prototype is located in the cloud, runs full-stack SMP Linux, and supports network connections via its serial interfaces. This aspect allows users to transform their custom architectures into first-class citizens inside the AWS infrastructure.

To prove this, we build a cloud pipeline that includes a SMAPPIC 1x1x4 prototype and multiple AWS services. The prototype executes the Nginx web server and PHP backend script, as well as interacts with AWS Lambda and S3 services. Fig. 12 shows the built pipeline. The incoming HTTP request goes through the following series of events:

Table 3: Requirements for host machines and cheapest suitable AWS EC2 instances [44].

	Sniper	gem5	Verilator	SMAPPIC & FireSim
#vCPUs	2	1	1	1
Memory	8 GB	64 GB	8 GB	8 GB
FPGAs	0	0	0	1
Instance	t3.m	r5.2xl	t3.m	f1.2xl
Price/hr	$0.04	$0.45	$0.04	$1.65

(1) The HTTP request enters the AWS Lambda function. The function acts as a gateway from the Internet to the private network. It redirects requests to the Nginx web server running on the SMAPPIC prototype.
(2) The web server guides the request through the Common Gateway Interface (CGI) into the PHP script executed in the prototype alongside the web server.
(3) The script fetches the data from the AWS S3 service.
(4) The script attaches the current time to the data and returns it as a response to the web server.
(5) The web server returns the response back to the Lambda function.
(6) The function returns the response back to the original client.

This workflow presents a unique opportunity for researchers to integrate their custom design into a massive warehouse-scale datacenter and test its behavior and performance while interacting with complex cloud services. It enables the researchers to run RISC-V in the cloud and Internet with AWS's vast APIs and observe the execution details directly inside the machine.

4.5 Cost-Efficient Architecture Modeling

In Sec. 4.1 we demonstrate that SMAPPIC is a valuable tool for large-scale architecture modeling. However, SMAPPIC also shows superior cost-efficiency among other modeling tools in the cloud. We compare the cost of modeling a similar 64-bit RISC-V system with different tools to demonstrate this. We use Sniper [14] as an example of a parallel simulator, gem5 [30] as an example of a cycle-level simulator, and Verilator [51] as an example of an RTL simulator. We also compare our work against another cloud FPGA tool, FireSim [24]. We use two FireSim configurations: 1) without network simulation and with one quad-core RocketChip instance, and 2) supernode configuration with network simulation and with four single-core RocketChip instances on one FPGA. We want to compare with the most cost-efficient FireSim configuration as measured by running benchmarks.

We execute benchmarks on a real RISC-V chip SiFive Freedom U740 SoC [46], part of the HiFive Unmatched [47] development platform, to get baseline results when running benchmarks in real RISC-V silicon.

We use a SMAPPIC prototype in a 1x4x2 configuration *without* an inter-node interconnect. This configuration allows us to model four independent prototypes inside one FPGA and significantly improves the cost-efficiency of SMAPPIC. The studied system parameters are listed in Table 2. All other tools were configured to model systems

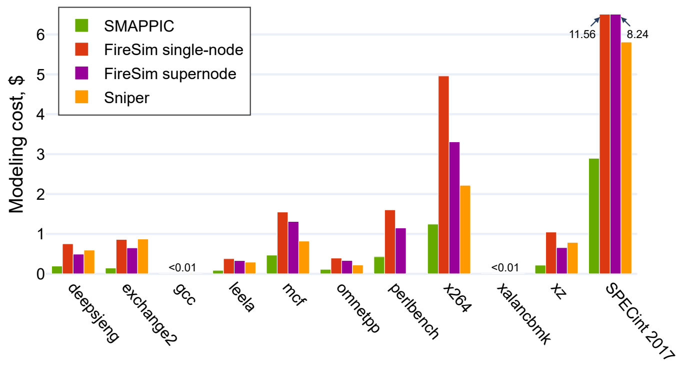


Figure 13: Modeling costs in dollars. Results for gem5 are not shown because its costs are 4-5 orders of magnitude higher than in all other benchmarks. SMAPPIC delivers the best cost efficiency in a cloud setting out of all methods.

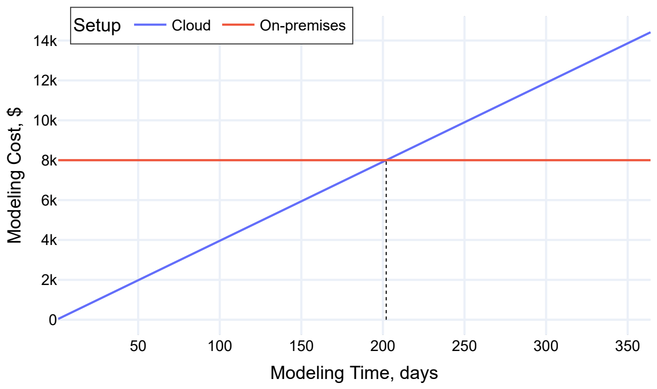


Figure 14: Cost of FPGA modeling in the cloud vs. locally on-premises. The cloud setup is more cost-efficient for up to 200 days of continuous experiments.

as similar to ours as possible. It is worth noting that all of the tools below are configured to use RISC-V workloads, except Sniper. Sniper has had initial support for RISC-V since version 7.3 [50], but it could not run RISC-V benchmarks out of the box or after we attempted to find and fix the problem. Therefore, we run Sniper with the X86-64 version of benchmarks. We do not expect this to affect our results significantly.

Our target benchmark suite is SPECint 2017 with the "test" input size for simplicity. We could not run the benchmark *perlbench* on Sniper because it does not support forks inside executing workloads.

For each of the studied tools, we find the smallest and cheapest suitable instance in AWS EC2 offerings and then use its price per hour to calculate the cost of benchmark execution. Our minimal instance selections are described in Table 3.

The main limiting factor of software tools is the amount of memory the host instance has: Sniper needs at least 8 GB for proper functioning, and most of the gem5 runs could fit into a 64 GB allocation. However, two of the benchmarks required even more memory. For example, the gem5 run with the *mcf* benchmark completes only on a host with 350 GB of memory.

Modeling costs are shown in Fig. 13. As we expected, SMAPPIC shows the best cost-efficiency among other approaches. The main reason is its ability to allocate multiple prototype instances in a single FPGA and the high frequency provided by direct RTL-to-gate mapping. gem5 shows the worst results with 4-5 orders of magnitude higher cost than other tools because of its large memory requirements and long execution time. For this reason, we did not include gem5 results in the chart.

Compared to a single-node FireSim configuration, SMAPPIC shows about four times better cost-efficiency. Single-node FireSim and SMAPPIC have similar frequencies. However, SMAPPIC models four separate instances in parallel on one FPGA, whereas FireSim models only one. Similar to SMAPPIC, supernode FireSim puts four systems into one FPGA but still shows worse results because of its lower frequency.

For the sake of completeness, we also compare Verilator with SMAPPIC. In our small "Hello World" example, the Verilator simulation takes 65 seconds, and SMAPPIC finishes in 4ms. Based on data from Table 3 we can derive that SMAPPIC is about 1600 times more cost-efficient. Such a significant advantage makes a big difference in design verification.

As mentioned in Section 2.1, researchers can avoid using cloud infrastructure for FPGA prototypes and purchase similar hardware for on-premise usage instead. Our estimates on the price of similar servers with similar FPGA boards are shown in Table 1.

Based on these estimates and F1 instance pricing, we can compare the costs of using cloud and on-premises FPGA prototypes. Results are shown in Fig. 14.

Given the same prototype configuration running in both cases, it takes more than 200 days of continuous modeling to justify buying a private FPGA setup and using it on-premises. We consider this scenario possible only for huge research groups where such hardware will be shared between many users. However, even in such cases, the burden of proper hardware management and lack of ability to scale modeling makes the cloud option more preferable.

4.6 Remote Work

Using a cloud setup might be the only option during unexpected workflow disruptions like the current COVID-19 pandemic. In the last year and a half, together with our collaborators from other institutions, we finished two large chip tapeouts. These chips are based on heterogeneous architectures with multiple accelerators and complex interactions through a coherent cache subsystem. Such chip parameters require extensive design testing with real, large workloads and driver-controlled accelerators running under a full-stack operating system.

In such conditions, SMAPPIC was the only realistic option to perform design verification. RTL simulators do not have high enough performance to make the modeling of complete real system (with OS) workloads possible. At the same time, physical FPGA access was not available due to the Work-From-Home order.

After we finished the initial integration of accelerators, the prototype and all of its functionality worked out of the box. The cloud nature of SMAPPIC allowed us to launch multiple instances and work in parallel while being physically distributed in many different time zones. We found two critical bugs inside the accelerators themselves, one bug inside the RISC-V interrupt controller, and one problem with the interfacing of an accelerator and TRI interface.

4.7 Using SMAPPIC for Education

The value of using FPGAs in teaching architecture classes has been noted by multiple educators [35, 52]. However, before the era of cloud FPGAs, educational institutions could afford only limited use of FPGAs in classes. SMAPPIC is a potent tool for overcoming this problem. With a proper management tool, educators can use SMAPPIC to launch hundreds of instances on-demand and pay only for the time students actually use FPGAs for their assignments or projects. SMAPPIC is one of the tools used for final projects by students taking the computer architecture class at our institution this year.

4.8 Platform Limitations

Even though SMAPPIC allows for many unique use cases that are not achievable with other tools, there are a couple of factors to consider before choosing it as a modeling platform.

First, while SMAPPIC provides a parametrized model of a typical tiled multicore architecture with NoC and directory coherence, there are only a couple of provided **fixed** core models. For example, all case studies in this section used the Ariane core: an in-order core with a single-issue pipeline which may not be representative of large-core out-of-order designs. Users have two options on how to approach this limitation.

(1) Provide the RTL design of their own custom core, use the functionality provided by the BYOC framework and TRI interface, and integrate it into the SMAPPIC model. In this case, the modeling results will be more accurate and represent the real system. However, this option requires some effort from the user.
(2) Use one of the core models provided by SMAPPIC out of the box. This option requires less effort, but the modeling results will not represent the final target system exactly if its core differs from the core chosen in SMAPPIC. Nevertheless, we think this option is valuable for researchers who are more focused on obtaining *qualitative* results or care more about the interactions in the system as a whole. For example, this option is useful for studying the interaction between the cores and the integrated accelerators.

Another factor to consider when choosing SMAPPIC for research is the physical constraints of the FPGA infrastructure:

(1) There is a finite number of gates per FPGA; this restricts the amount of logic that can be allocated per prototype and its frequency. For example, F1 FPGAs can fit at most 12 Ariane tiles at 75 MHz frequency. Table 4 shows various possible SMAPPIC setups. All configurations use core parameters from Table 2.
(2) Each F1 FPGA contains only four memory slots, and each node in SMAPPIC has a separate memory controller. This requirement puts a cap of at most four nodes per FPGA in SMAPPIC.
(3) Only four FPGAs in the F1 instance are connected with low-latency PCIe links. Therefore, one SMAPPIC prototype can include at most four FPGAs.

Table 4: SMAPPIC configurations with frequencies and LUT utilizations. Configurations are shown in BxC format, where B is the number of nodes per FPGA, and C is the number of tiles per node. Tile parameters are listed in Table 2.

Configuration	Frequency	LUT Utilization
1x12	75 MHz	97%
1x10	100 MHz	83%
2x4	100 MHz	73%
2x5	75 MHz	88%
4x2	100 MHz	87%

(4) The PCIe connection has a round-trip latency of 1250 ns. This latency puts a lower limit on modeled inter-node link latency. For example, at 100 MHz, the round-trip latency is equal to 125 cycles.

Most of these limits are not fundamental and are only dictated by the AWS infrastructure: the number of tiles per FPGA is limited by the FPGA size, the number of nodes per FPGA is limited by the number of DRAM slots per FPGA, and the number of FPGAs in a prototype is limited by the number of FPGAs in one F1 instance. The limits can be raised in the future if AWS updates its cloud FPGA infrastructure or other cloud providers introduce a larger selection of FPGAs.

We also want to note that SMAPPIC is not a total substitution for classic software simulators. Researchers often value flexibility over execution time, cost, or accuracy. FPGA emulation can be a poor fit in such situations because each system parameter's change leads to complete prototype regeneration which can take hours. However, the use cases discussed above demonstrate situations where SMAPPIC is the users' only real option due to its scale, speed, accuracy, and cost-efficiency.

5 RELATED WORK

5.1 FPGAs for Architecture Modeling

There are a few commercial tools for FPGA-accelerated design modeling, such as Cadence Palladium [13], Mentor Veloce [45], and Synopsys Zebu [55]. These systems are proprietary, have incredibly high prices (millions of dollars), and are not accessible to most researchers. SMAPPIC is open-source and uses a publicly available cloud infrastructure instead.

ProtoFlex [18] and FAST [15] use hardware to accelerate only the performance layer of the model. Some researchers propose using an FPGA to model only subparts of the whole system, e.g., memory subsystem [53] or NoC [26, 29]. In contrast, SMAPPIC models each node entirely inside the FPGA, making performance-accuracy tradeoffs more visible.

RAMP Gold [57], DIABLO [56], and HASim [40] put both performance and functional models into the FPGA. However, these frameworks use various techniques to pack more target logic inside a single FPGA at the cost of lower frequency. SMAPPIC uses direct gate-to-RTL mapping and keeps frequency as high as possible instead. SMAPPIC models large systems by partitioning across multiple FPGAs. Such a scaling strategy has good cost-efficiency in a cloud setting.

5.2 Multi-node System Modeling

PriME [23] supports the modeling of independent chips without any connectivity. MGPUSim [54] was designed specifically for multi-die modeling but only in the GPU domain. Sniper [14] supports modeling multi-die systems with shared memory but at much lower speeds than FPGAs. Unlike all these tools, SMAPPIC supports fast modeling of heterogeneous shared memory multi-node systems.

Enzian [20] and QuickIA [16] are similar to SMAPPIC in that they provide the foundation for building heterogeneous multi-chip prototypes. However, both frameworks use CPU + FPGA design where only the logic inside the FPGA can be modified. In contrast, SMAPPIC is fully customizable and can be changed by configuration options or modifying source code.

BYOC [10] and OpenPiton [9, 11] provide separate tools for multi-chip modeling and FPGA prototyping. However, they do not allow making FPGA prototypes of multi-chip architectures. Moreover, they support multi-chip capabilities only through a special hardware/software mechanism called Coherence Domain Restriction [22]. In contrast, SMAPPIC changes the original BYOC framework to provide multi-node capabilities out of the box without additional software modifications.

5.3 Multi-FPGA Architecture Modeling

FireSim [24] and DIABLO [56] support the modeling of multi-chip systems distributed over multiple FPGAs. However, they do it only without a unified memory, with different chips connected over Ethernet links. Unlike these tools, SMAPPIC provides coherent inter-node interconnect that tunnels existing inter-node interconnect through FPGA's PCIe connection. This mechanism allows the modeling of multi-node systems with unified memory.

The Twinstar platform [6] can model the full 16-core Bluegene/Q compute chip but uses a local, sophisticated, expensive, and proprietary setup consisting of 60 FPGAs and can reach only speeds of 4 MHz. Moreover, the logic partition between the FPGAs in Twinstar was done manually by the engineers. SMAPPIC divides the design between FPGAs based on the user's input and leverages the cloud infrastructure to free researchers from these complications.

5.4 Architecture Modeling in the Cloud and Its Cost Optimization

The FAME work [58] estimates the cost of software architecture simulation in the cloud. It concludes that running FPGA simulations on-premises is more cost-effective than running **software** simulations in the cloud. Our work makes much broader estimations and adds price calculations for cloud FPGA simulations and different types of software simulations.

The FireSim paper [24] compares the cost of cloud and on-premises FPGA simulation and concludes that large-scale simulations are financially sustainable only when using cloud infrastructure. Unlike SMAPPIC, FireSim is not compared with software simulators in terms of pricing.

5.5 Architecture Modeling in Cloud FPGAs

The main previous work on utilizing cloud FPGAs for architecture modeling is FireSim [24]. We consider SMAPPIC and FireSim two significantly different frameworks because they have different goals influencing their designs.

FireSim is designed from the ground up to be a datacenter simulator, and there is a big emphasis on simulating the network between separate nodes. This factor puts an additional restriction on the simulation speed and the size of separate computational nodes.

Unlike FireSim, SMAPPIC is designed to make the modeling of a separate system scalable. This goal allows SMAPPIC to scale prototypes up to large, 48-core, 4-node systems and be cost-effective when modeling small systems simultaneously. This result is achievable because we do not have the additional requirement of simulating IP network interactions.

6 CONCLUSION

This paper presents SMAPPIC – the first open-source prototype platform for shared memory multi-die architectures on cloud FPGAs. SMAPPIC achieves ease of use, cost-effectiveness, and multi-node scalability through modular, hierarchical, and parametrizable structure based on a cloud FPGA backend. By using BYOC's infrastructure and AWS EC2 F1 instances, SMAPPIC provides a configurable underlying design that allows researchers to quickly and easily start using cloud-enabled FPGA prototypes.

SMAPPIC makes many unique use cases possible or significantly more accessible, in particular: large-scale multi-node architecture modeling, accelerator evaluation and verification, hardware/software co-development, *in situ* studies of custom architecture interaction with a cloud infrastructure, cost-efficient modeling, remote work, and education. We use SMAPPIC to build the first open-source multi-node 48-core 64-bit Linux-capable RISC-V system with unified memory and to make the first comparison of the architecture modeling methods' costs in a cloud. SMAPPIC has the potential to enable even more exceptional studies in the future.

ACKNOWLEDGMENTS

We would like to thank Alexey Lavrov, Jonathan Balkind, and the whole OpenPiton/BYOC team for sharing their expertise on two frameworks, Marcelo Orenes-Vera for his help in reimplementing MAPLE in SMAPPIC, Georgios Tziantzioulis for his aid with GNG accelerator integration, and Ang Li, Zujun Tan, Yebin Chon, Rohan Prabhakar, August Ning, and anonymous reviewers for their valuable feedback. We also thank the AWS Cloud Credit for Research program. This material is based on research sponsored by the National Science Foundation under Grant No. CNS-1823222, Air Force Research Laboratory (AFRL) and Defense Advanced Research Projects Agency (DARPA) under agreements No. FA8650-18-2-7852 and FA8650-18-2-7862. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Air Force Research Laboratory (AFRL) and Defense Advanced Research Projects Agency (DARPA) or the U.S. Government. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] ACM. 2020. ISCA '20: Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture. https://dl.acm.org/doi/proceedings/10.5555/3426241

[2] ACM. 2021. ASPLOS 2021: Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems. https://dl.acm.org/doi/proceedings/10.1145/3445814

[3] ACM. 2021. PLDI 2021: Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation. https://dl.acm.org/doi/proceedings/10.1145/3453483

[4] Dr. Ian Cutress Andrei Frumusanu. 2021. AMD 3rd Gen EPYC Milan Review: A Peak vs Per Core Performance Balance. https://www.anandtech.com/show/16529/amd-epyc-milan-review/4

[5] John Hauser Andrew Waterman, Krste Asanović. 2021. RISC-V Specification. https://riscv.org/technical/specifications/

[6] Sameh Asaad, Ralph Bellofatto, Bernard Brezzo, Chuck Haymes, Mohit Kapur, Benjamin Parker, Thomas Roewer, Proshanta Saha, Todd Takken, and José Tierno. 2012. A Cycle-Accurate, Cycle-Reproducible Multi-FPGA System for Accelerating Multi-Core Processor Simulation. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays* (Monterey, California, USA) *(FPGA '12)*. Association for Computing Machinery, New York, NY, USA, 153–162. https://doi.org/10.1145/2145694.2145720

[7] David H. Bailey. 2011. *NAS Parallel Benchmarks*. Springer US, Boston, MA, 1254–1259. https://doi.org/10.1007/978-0-387-09766-4_133

[8] Raghuraman Balasubramanian, Vinay Gangadhar, Ziliang Guo, Chen-Han Ho, Cherin Joseph, Jaikrishnan Menon, Mario Paulo Drumond, Robin Paul, Sharath Prasad, Pradip Valathol, and Karthikeyan Sankaralingam. 2015. MIAOW - An open source RTL implementation of a GPGPU. In *2015 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS XVIII)*. 1–3. https://doi.org/10.1109/CoolChips.2015.7158663

[9] Jonathan Balkind, Ting-Jung Chang, Paul J. Jackson, Georgios Tziantzioulis, Ang Li, Fei Gao, Alexey Lavrov, Grigory Chirkov, Jinzheng Tu, Mohammad Shahrad, and David Wentzlaff. 2020. OpenPiton at 5: A Nexus for Open and Agile Hardware Design. *IEEE Micro* 40, 4 (2020), 22–31. https://doi.org/10.1109/MM.2020.2997706

[10] Jonathan Balkind, Katie Lim, Michael Schaffner, Fei Gao, Grigory Chirkov, Ang Li, Alexey Lavrov, Tri M. Nguyen, Yaosheng Fu, Florian Zaruba, Kunal Gulati, Luca Benini, and David Wentzlaff. 2020. BYOC: A "Bring Your Own Core" Framework for Heterogeneous-ISA Research. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems* (Lausanne, Switzerland) *(ASPLOS '20)*. Association for Computing Machinery, New York, NY, USA, 699–714. https://doi.org/10.1145/3373376.3378479

[11] Jonathan Balkind, Michael McKeown, Yaosheng Fu, Tri Nguyen, Yanqi Zhou, Alexey Lavrov, Mohammad Shahrad, Adi Fuchs, Samuel Payne, Xiaohua Liang, Matthew Matl, and David Wentzlaff. 2016. OpenPiton: An Open Source Manycore Research Framework. *SIGARCH Comput. Archit. News* 44, 2 (mar 2016), 217–232. https://doi.org/10.1145/2980024.2872414

[12] Bittware. [n. d.]. BittWare XUP-P3R. https://www.bittware.com/fpga/xup-p3r/

[13] Cadence. [n. d.]. High performance hardware and software verification and debug of complex SoCs and Systems. https://www.cadence.com/en_US/home/tools/system-design-and-verification/emulation-and-prototyping/palladium.html

[14] Trevor E. Carlson, Wim Heirman, and Lieven Eeckhout. 2011. Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation. In *SC '11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–12. https://doi.org/10.1145/2063384.2063454

[15] Derek Chiou, Dam Sunwoo, Joonsoo Kim, Nikhil A. Patil, William Reinhart, Darrel Eric Johnson, Jebediah Keefe, and Hari Angepat. 2007. FPGA-Accelerated Simulation Technologies (FAST): Fast, Full-System, Cycle-Accurate Simulators. In *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*. 249–261. https://doi.org/10.1109/MICRO.2007.36

[16] Nagabhushan Chitlur, Ganapati Srinivasa, Scott Hahn, P K Gupta, Dheeraj Reddy, David Koufaty, Paul Brett, Abirami Prabhakaran, Li Zhao, Nelson Ijih, Suchit Subhaschandra, Sabina Grover, Xiaowei Jiang, and Ravi Iyer. 2012. QuickIA: Exploring heterogeneous architectures on real prototypes. In *IEEE International Symposium on High-Performance Comp Architecture*. 1–8. https://doi.org/10.1109/HPCA.2012.6169046

[17] Rangeen Basu Roy Chowdhury, Anil K. Kannepalli, and Eric Rotenberg. 2016. AnyCore-1: A comprehensively adaptive 4-way superscalar processor. In *2016 IEEE Hot Chips 28 Symposium (HCS)*. 1–1. https://doi.org/10.1109/HOTCHIPS.2016.7936237

[18] Eric S. Chung, Michael K. Papamichael, Eriko Nurvitadhi, James C. Hoe, Ken Mai, and Babak Falsafi. 2009. ProtoFlex: Towards Scalable, Full-System Multiprocessor Simulations Using FPGAs. *ACM Trans. Reconfigurable Technol. Syst.* 2, 2, Article 15 (jun 2009), 32 pages. https://doi.org/10.1145/1534916.1534925

[19] Alibaba Cloud. [n. d.]. Alibaba Cloud Computing Services. https://www.alibabacloud.com/product/computing?spm=a3c0i.239195.6791778070.157.7af912bdlh7EC5#J_9413186770

[20] David Cock, Abishek Ramdas, Daniel Schwyn, Michael Giardino, Adam Turowski, Zhenhao He, Nora Hossle, Dario Korolija, Melissa Licciardello, Kristina Martsenko, Reto Achermann, Gustavo Alonso, and Timothy Roscoe. 2022. Enzian: An Open, General, CPU/FPGA Platform for Systems Software Research. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Lausanne, Switzerland) *(ASPLOS 2022)*. Association for Computing Machinery, New York, NY, USA, 434–451. https://doi.org/10.1145/3503222.3507742

[21] Andrei Frumusanu. 2021. Intel 3rd Gen Xeon Scalable (Ice Lake SP) Review: Generationally Big, Competitively Small. https://www.anandtech.com/show/16594/intel-3rd-gen-xeon-scalable-review/4

[22] Yaosheng Fu, Tri M. Nguyen, and David Wentzlaff. 2015. Coherence domain restriction on large scale systems. In *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 686–698. https://doi.org/10.1145/2830772.2830832

[23] Yaosheng Fu and David Wentzlaff. 2014. PriME: A parallel and distributed simulator for thousand-core chips. In *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 116–125. https://doi.org/10.1109/ISPASS.2014.6844467

[24] Sagar Karandikar, Howard Mao, Donggyu Kim, David Biancolin, Alon Amid, Dayeol Lee, Nathan Pemberton, Emmanuel Amaro, Colin Schmidt, Aditya Chopra, Qijing Huang, Kyle Kovacs, Borivoje Nikolic, Randy Katz, Jonathan Bachrach, and Krste Asanović. 2018. FireSim: FPGA-accelerated Cycle-exact Scale-out System Simulation in the Public Cloud. In *Proceedings of the 45th Annual International Symposium on Computer Architecture* (Los Angeles, California) *(ISCA '18)*. IEEE Press, Piscataway, NJ, USA, 29–42. https://doi.org/10.1109/ISCA.2018.00014

[25] Hassan Khan, David Hounshell, and Erica Fuchs. 2018. Science and research policy at the end of Moore's law. *Nature Electronics* 1 (01 2018). https://doi.org/10.1038/s41928-017-0005-9

[26] Kouadri-Mostefaoui, Abdellah-Medjadji, Benaoumeur Senouci, and Frederic Petrot. 2008. Large Scale On-Chip Networks : An Accurate Multi-FPGA Emulation Platform. In *2008 11th EUROMICRO Conference on Digital System Design Architectures, Methods and Tools*. 3–9. https://doi.org/10.1109/DSD.2008.130

[27] D.-U. Lee, J.D. Villasenor, W. Luk, and P.H.W. Leong. 2006. A hardware Gaussian noise generator using the Box-Muller method and its error analysis. *IEEE Trans. Comput.* 55, 6 (2006), 659–671. https://doi.org/10.1109/TC.2006.81

[28] Guangxi Liu. [n. d.]. OpenCores Gaussian Noise Generator. https://opencores.org/projects/gng

[29] Yangfan Liu, Peng Liu, Yingtao Jiang, Mei Yang, Kejun Wu, Weidong Wang, and Qingdong Yao. 2010. Building a multi-FPGA-based emulation framework to support networks-on-chip design and verification. *International Journal of Electronics - INT J ELECTRON* 97 (10 2010), 1241–1262. https://doi.org/10.1080/00207217.2010.512017

[30] Jason Lowe-Power, Abdul Mutaal Ahmad, Ayaz Akram, Mohammad Alian, Rico Amslinger, Matteo Andreozzi, Adrià Armejach, Nils Asmussen, Brad Beckmann, Srikant Bharadwaj, Gabe Black, Gedare Bloom, Bobby R. Bruce, Daniel Rodrigues Carvalho, Jeronimo Castrillon, Lizhong Chen, Nicolas Derumigny, Stephan Diestelhorst, Wendy Elsasser, Carlos Escuin, Marjan Fariborz, Amin Farmahini-Farahani, Pouya Fotouhi, Ryan Gambord, Jayneel Gandhi, Dibakar Gope, Thomas Grass, Anthony Gutierrez, Bagus Hanindhito, Andreas Hansson, Swapnil Haria, Austin Harris, Timothy Hayes, Adrian Herrera, Matthew Horsnell, Syed Ali Raza Jafri, Radhika Jagtap, Hanhwi Jang, Reiley Jeyapaul, Timothy M. Jones, Matthias Jung, Subash Kannoth, Hamidreza Khaleghzadeh, Yuetsu Kodama, Tushar Krishna, Tommaso Marinelli, Christian Menard, Andrea Mondelli, Miquel Moreto, Tiago Mück, Omar Naji, Krishnendra Nathella, Hoa Nguyen, Nikos Nikoleris, Lena E. Olson, Marc Orr, Binh Pham, Pablo Prieto, Trivikram Reddy, Alec Roelke, Mahyar Samani, Andreas Sandberg, Javier Setoain, Boris Shingarov, Matthew D. Sinclair, Tuan Ta, Rahul Thakur, Giacomo Travaglini, Michael Upton, Nilay Vaish, Ilias Vougioukas, William Wang, Zhengrong Wang, Norbert Wehn, Christian Weis, David A. Wood, Hongil Yoon, and Éder F. Zulian. 2020. The gem5 Simulator: Version 20.0+. https://doi.org/10.48550/ARXIV.2007.03152

[31] Michael McKeown, Alexey Lavrov, Mohammad Shahrad, Paul J. Jackson, Yaosheng Fu, Jonathan Balkind, Tri M. Nguyen, Katie Lim, Yanqi Zhou, and David Wentzlaff. 2018. Power and Energy Characterization of an Open Source 25-Core Manycore Processor. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 762–775. https://doi.org/10.1109/HPCA.2018.00070

[32] Microsoft. [n. d.]. NP-series. https://docs.microsoft.com/en-us/azure/virtual-machines/np-series

[33] Samuel Naffziger, Noah Beck, Thomas Burd, Kevin Lepak, Gabriel H. Loh, Mahesh Subramony, and Sean White. 2021. Pioneering Chiplet Technology and Design for the AMD EPYC™ and Ryzen™ Processor Families : Industrial Product. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. 57–70. https://doi.org/10.1109/ISCA52012.2021.00014

[34] NVIDIA. [n. d.]. NVIDIA Deep Learning Accelerator. http://nvdla.org/

[35] Joaquín Olivares, José Manuel Palomares, José Manuel Soto, and Juan Carlos Gámez. 2010. Teaching microprocessors design using FPGAs. In *IEEE EDUCON 2010 Conference*. 1189–1193. https://doi.org/10.1109/EDUCON.2010.5492390

[36] OpenCores. 2022. OpenCores project. https://opencores.org/
[37] Oracle. [n. d.]. OpenSPARC T1. https://www.oracle.com/servers/technologies/opensparc-t1-page.html
[38] Marcelo Orenes-Vera, Aninda Manocha, Jonathan Balkind, Fei Gao, Juan L. Aragón, David Wentzlaff, and Margaret Martonosi. 2022. Tiny but Mighty: Designing and Realizing Scalable Latency Tolerance for Manycore SoCs. In *Proceedings of the 49th Annual International Symposium on Computer Architecture* (New York, New York) *(ISCA '22)*. Association for Computing Machinery, New York, NY, USA, 817–830. https://doi.org/10.1145/3470496.3527400
[39] Aleksander Osman. [n. d.]. ao486. https://github.com/alfikpl/ao486
[40] Michael Pellauer, Michael Adler, Michel Kinsy, Angshuman Parashar, and Joel Emer. 2011. HAsim: FPGA-based high-detail multicore simulation using time-division multiplexing. In *2011 IEEE 17th International Symposium on High Performance Computer Architecture*. 406–417. https://doi.org/10.1109/HPCA.2011.5749747
[41] Daniel Petrisko, Farzam Gilani, Mark Wyse, Dai Cheol Jung, Scott Davidson, Paul Gao, Chun Zhao, Zahra Azad, Sadullah Canakci, Bandhav Veluri, Tavio Guarino, Ajay Joshi, Mark Oskin, and Michael Bedford Taylor. 2020. BlackParrot: An Agile Open-Source RISC-V Multicore for Accelerator SoCs. *IEEE Micro* 40, 4 (2020), 93–102. https://doi.org/10.1109/MM.2020.2996145
[42] Amazon Web Services. 2021. AWS Shell Interface Specification. https://github.com/aws/aws-fpga/blob/master/hdk/docs/AWS_Shell_Interface_Specification.md
[43] Amazon Web Services. 2022. Amazon EC2 F1 Instances. https://aws.amazon.com/ec2/instance-types/f1/
[44] Amazon Web Services. 2022. Amazon EC2 On-Demand Pricing. https://aws.amazon.com/ec2/pricing/on-demand/
[45] Siemens. [n. d.]. Veloce HW-Assisted Verification System. https://eda.sw.siemens.com/en-US/ic/veloce/
[46] SiFive. 2021. SiFive FU740-C000 Manual. https://sifive.cdn.prismic.io/sifive/de1491e5-077c-461d-9605-e8a0ce57337d_fu740-c000-manual-v1p3.pdf
[47] SiFive. 2022. HiFive Unmatched. https://www.sifive.com/boards/hifive-unmatched
[48] Dimitrios Skarlatos, Apostolos Kokolis, Tianyin Xu, and Josep Torrellas. 2020. Elastic Cuckoo Page Tables: Rethinking Virtual Memory Translation for Parallelism. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems* (Lausanne, Switzerland) *(ASPLOS '20)*. Association for Computing Machinery, New York, NY, USA, 1093–1108. https://doi.org/10.1145/3373376.3378493
[49] James E. Smith. 1982. Decoupled Access/Execute Computer Architectures. *SIGARCH Comput. Archit. News* 10, 3 (apr 1982), 112–119. https://doi.org/10.1145/1067649.801719
[50] Sniper. 2021. Sniper Releases. http://snipersim.org/w/Releases
[51] Wilson Snyder. 2013. Verilator: Open simulation-growing up. *DVClub Bristol* (2013).
[52] Andrew Strelzoff. 2007. Teaching computer architecture with fpga soft processors. In *ASEE Southeast Section Conference*.
[53] Bharat Sukhwani, Thomas Roewer, Charles L. Haymes, Kyu-Hyoun Kim, Adam J. McPadden, Daniel M. Dreps, Dean Sanner, Jan Van Lunteren, and Sameh Asaad. 2017. ConTutto – A Novel FPGA-based Prototyping Platform Enabling Innovation in the Memory Subsystem of a Server Class Processor. In *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 15–26. https://doi.org/10.1145/3123939.3124535
[54] Yifan Sun, Trinayan Baruah, Saiful A. Mojumder, Shi Dong, Xiang Gong, Shane Treadway, Yuhui Bao, Spencer Hance, Carter McCardwell, Vincent Zhao, Harrison Barclay, Amir Kavyan Ziabari, Zhongliang Chen, Rafael Ubal, José L. Abellán, John Kim, Ajay Joshi, and David Kaeli. 2019. MGPUSim: Enabling Multi-GPU Performance Modeling and Optimization. In *Proceedings of the 46th International Symposium on Computer Architecture* (Phoenix, Arizona) *(ISCA '19)*. Association for Computing Machinery, New York, NY, USA, 197–209. https://doi.org/10.1145/3307650.3322230
[55] Synopsys. [n. d.]. ZeBu Server 4. https://www.synopsys.com/verification/emulation/zebu-server.html
[56] Zhangxi Tan, Zhenghao Qian, Xi Chen, Krste Asanovic, and David Patterson. 2015. DIABLO: A Warehouse-Scale Computer Network Simulator Using FPGAs. *SIGPLAN Not.* 50, 4 (mar 2015), 207–221. https://doi.org/10.1145/2775054.2694362
[57] Zhangxi Tan, Andrew Waterman, Rimas Avizienis, Yunsup Lee, Henry Cook, David Patterson, and Krste Asanović. 2010. RAMP gold: an FPGA-based architecture simulator for multiprocessors. In *Proceedings of the 47th Design Automation Conference*. 463–468. https://doi.org/10.1145/1837274.1837390
[58] Zhangxi Tan, Andrew Waterman, Henry Cook, Sarah Bird, Krste Asanović, and David Patterson. 2010. A Case for FAME: FPGA Architecture Model Execution. In *Proceedings of the 37th Annual International Symposium on Computer Architecture* (Saint-Malo, France) *(ISCA '10)*. Association for Computing Machinery, New York, NY, USA, 290–301. https://doi.org/10.1145/1815961.1815999
[59] Robert Tausworthe. 1965. Random Numbers Generated by Linear Recurrence Modulo Two. *Mathematics of Computation - Math. Comput.* 19 (05 1965), 201–201. https://doi.org/10.2307/2003345
[60] Clifford Wolf. [n. d.]. PicoRV32. https://github.com/cliffordwolf/picorv32
[61] Xilinx. 2016. Xilinx AXI UART 16550 v2.0. https://www.xilinx.com/support/documentation/ip_documentation/axi_uart16550/v2_0/pg143-axi-uart16550.pdf
[62] Xilinx. 2022. Alveo U250 Data Center Accelerator Card. https://www.xilinx.com/products/boards-and-kits/alveo/u250.html
[63] Xilinx. 2022. DMA/Bridge Subsystem for Express v4.1. https://www.xilinx.com/content/dam/xilinx/support/documentation/ip_documentation/xdma/v4_1/pg195-pcie-dma.pdf
[64] Xilinx. 2022. Xilinx Virtex UltraScale+. https://www.xilinx.com/products/silicon-devices/fpga/virtex-ultrascale-plus.html#productAdvantages
[65] 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 (Nov 2019), 2629–2640. https://doi.org/10.1109/tvlsi.2019.2926114
[66] Hansen Zhang, Soumyadeep Ghosh, Jordan Fix, Sotiris Apostolakis, Stephen R. Beard, Nayana P. Nagendra, Taewook Oh, and David I. August. 2019. Architectural Support for Containment-Based Security. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems* (Providence, RI, USA) *(ASPLOS '19)*. Association for Computing Machinery, New York, NY, USA, 361–377. https://doi.org/10.1145/3297858.3304020

Received 2022-07-07; accepted 2022-09-22