Evaluation of MindPalace for Chip Design Tradeoffs on Function-as-a-Service

Kaifeng Xu
Princeton University
Princeton, USA
kaifengx@princeton.edu

Georgios Tziantzioulis [1]
Princeton University
Princeton, USA
georgios.tziantzioulis@pm.me

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

***Abstract*—Function-as-a-Service (FaaS) continues to be a popular and growing way to access abundant cloud resources. Both the research community and industry have begun investigating designing new or modifying existing processors for FaaS. While there exists many traditional computer architecture simulators, the complex, full-system and full-stack nature of FaaS workloads has opened the door to needing more purpose-built FaaS-specific simulators along with easy to use, simulation-friendly, characteristic workloads. In order to fully study FaaS workloads, the simulator needs to run a full OS running many concurrent containers with a full-stack serverless platform like OpenWhisk.**

This paper characterizes and describes MindPalace which is a simulator and framework designed to investigate micro-architecture innovations for FaaS workloads. MindPalace utilizes both QEMU and ChampSim. With QEMU, MindPalace is able to run complex, full-system and full-stack workloads including running the Apache OpenWhisk platform. By leveraging ChampSim, MindPalace can conduct detailed computer architecture and micro-architecture research and has access to advanced micro-architecture models for branch predictors and prefetchers. In addition, MindPalace uses a VM with Apache OpenWhisk running.

In this work, we compare MindPalace to other simulators and use MindPalace to investigate FaaS-specific architecture ideas including a design which reloads branch predictor state between function invocations. We also characterize MindPalace and study the accuracy of simulation versus real hardware in terms of branch prediction rates, cache miss rates, and IPC.

***Index Terms*—Computer Architecture, Simulation, Function-as-a-Service, Tracing**

I. Introduction

Serverless computing and Function-as-a-Service (FaaS) have been and continue to be growing portions of most cloud providers' workloads. FaaS provides a nice abstraction to the cloud programmer because it enables the programmer to only worry about the function that they are writing and outsource the OS, runtime, libraries, and autoscaling, to the cloud provider. This makes it easy to deploy functions and provides an easy pay-per-function execution abstraction with the cloud provider providing more resources as demand increases. Amazon Lambda [2], Microsoft Azure Functions [3], and Google Cloud Functions [4] are all examples of the three major cloud vendors embracing FaaS. When using FaaS, the cloud user simply writes a function which ultimately lives in a container and that function can respond to outside inputs all while being charged on a per-function-execution basis. One of the key benefits of FaaS is that the programmer does not need to worry about concerns such as managing VMs or starting new VMs to deal with increased load, with these concerns being left to the FaaS provider.

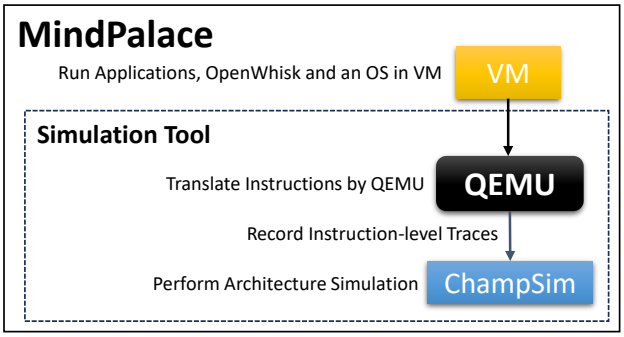


Fig. 1: MindPalace Framework Overview. The MindPalace simulation tool utilizes a modified QEMU which generates detailed traces that are fed into ChampSim for micro-architecture simulation. A VM running functions under OpenWhisk is our example workload. This is a simplified version of Figure 1 in [1].

While the FaaS programming and payment model has many advantages, the short-lived and fine-grain interleaved nature of running many potentially millisecond [5]–[7] functions from different customers/tenants is in conflict with many of the optimizations that have traditionally been exploited for performance in computer micro-architecture. In particular, typical computer architecture optimizations utilize temporal and spatial locality but the short-lived nature of FaaS functions can make it difficult for the micro-architecture to exploit locality of this exciting workload class.

Making matters even worse for the micro-architecture is that FaaS functions run inside a complex framework where the functions execute inside of containers which are themselves scheduled by complex FaaS frameworks like OpenWhisk [8] or the built-in schedulers of commercial FaaS platforms. The use of these complex frameworks together with the short-lived nature of the functions typically causes the OS and background

[1]The author was with the Department of Electrical and Computer Engineering, Princeton University, Princeton, NJ, USA at the time this work was conducted. He is now with Advanced Micro Devices (AMD).

2766-0486/25/$31.00 ©2025 IEEE
DOI 10.1109/ISPASS64960.2025.00027

tasks for the FaaS framework/scheduler taking a considerable fraction of the CPU resources and runtime.

There has been prior work that has studied the challenges of running FaaS workloads on current processors [9]–[11], but there is relatively little work studying new micro-architectures for FaaS and Serverless. Recent work has studied prefetching processor front-end state [12] and instructions [13] for warm container instances. But both of these works simplify the complex FaaS software stack. In particular, they focus only on containers and Remote Procedure Calls (RPCs), but do not simulate a full FaaS platform such as OpenWhisk therefore they can potentially miss full-system effects. In addition, other researchers have studied partitioning resources for serverless [14].

The emergence of FaaS workloads along with the architecture problems that they cause motivates the creation of new simulators. Many current simulators focus only on running user-space applications [15]–[18] which is insufficient to characterize the complex nature of a complete FaaS system. Full system simulators such as gem5 [19], [20] may execute too slow or have never demonstrated the ability to run full-stack FaaS environments. While recent simulators such as vSwarm [13] have been used to study [12] FaaS workloads, they have done so without running a full FaaS environment like OpenWhisk.

MindPalace[1] was recently introduced in an extended poster abstract [1]. In this work, we build upon that prior work by describing and evaluating MindPalace in more detail. While [1] only described MindPalace at a high level, in this paper, we dive into the details including how the framework works internally, discuss how MindPalace has been calibrated against real hardware, qualitatively evaluate MindPalace against other simulators including discussion of why MindPalace is needed for full-stack FaaS workloads along with a table comparing MindPalace to prior simulators, provide an example architecture use case, and provide a detailed related work section.

MindPalace is a full-system architectural framework which contains both an open-source simulator that builds upon QEMU [21] and ChampSim [22] as well as a virtual machine (VM) with OpenWhisk. In Figure 1 the high-level design of MindPalace can be seen. MindPalace uses and extends QEMU to record detailed memory and instruction traces for the OS and application. Due to the high compatibility of QEMU, it is able to handle large complex workloads such as full system FaaS workloads. The traces feed into ChampSim which provides detailed micro-architectural modeling and enables MindPalace to be used for detailed performance simulation.

We demonstrate MindPalace's ability to be used for researching new architecture and micro-architecture ideas by running serverless functions on full-stack OpenWhisk. We study function invocation traces generated MindPalace and show how such analysis can be used to foster future FaaS-specific architecture innovation. One example we study is the reloading of branch predictor state for TAGE [23] style branch predictors. By reloading branch predictor state, warmup of branch predictors can be sped up and we see MindPalace being the simulator of choice for future research into FaaS-specific architecture designs.

[1]MindPalace can be found at: github.com/PrincetonUniversity/MindPalace.

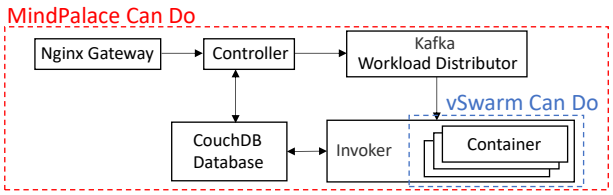


Fig. 2: OpenWhisk Architecture [8]. vSwarm-u [13] can only run container instances while MindPalace is capable of running full-stack OpenWhisk.

Utilizing the FaaS benchmark suite FunctionBench [24] and utilizing benchmarks from SPEC 2017 [25] we characterize the simulation results of MindPalace and compare simulated performance and other metrics (utilizing performance counters) against real hardware. We calibrate the underlying ChampSim versus real hardware by developing a methodology of error correction factors. We study what parameters influence the calibration of simulated performance of ChampSim and hence MindPalace versus real hardware. Utilizing performance counters on real hardware we are able to study how well ChampSim matches real hardware in terms of cache miss rate and branch miss rate.

The following contributions are made by this paper:

- This work provides a detailed description of the inner workings of MindPalace
- This work compares MindPalace to vSwarm-u and discusses what portions of a FaaS computation are not characterized on vSwarm-u when compared to a simulator that runs a full FaaS framework like MindPalace
- We demonstrate how MindPalace can be useful in designing new architectures for FaaS workloads, by studying the reloading of TAGE branch predictors
- We provide validation of MindPalace and calibration of ChampSim against a real machine utilizing performance counters
- Detailed related work comparing MindPalace to prior work

II. Limitations of Current Simulators

A full-system FaaS architectural simulation requires: an execution environment to run the full-system workload (normally a running virtual machine), a functional simulator that emulates the system behavior, and a performance simulator that simulates micro-architectural details.

Functional simulators are designed to emulate the behavior of instruction set architectures (ISA) without the implementation details of an underlying micro-architecture. This allows them to have faster emulation speed while capturing high-level information, such as memory accesses. Performance simulators aim to model the micro-architecture and provide detailed performance numbers for the target workload. Despite

TABLE I: Comparisons of Architecture Simulators

Simulator or Platform	Func. Model	Perf. Model	User-level	Full-system	Instr-level Info	Deploy Full FaaS Framework	Design New Architecture	Open Source	Speed
Real Processor	Yes	Yes	Yes	Yes	No	Yes	No	Unlikely	Fast
Pin [15]	Yes	No	Yes	No	Yes	No	No	No	Fast
Simics [26]	Yes	No	Yes	Yes	Yes	Unknown	No	No	Medium
SAE [27]	Yes	No	Yes	Yes	Yes	Unknown	Yes	No	Unknown
ChampSim [22]	No	Yes	Yes	Yes	Yes	No	Yes	Yes	Medium
Sniper [16]	Yes	Yes	Yes	No	Yes	No	Yes	Yes	Fast
gem5 [19], [20]	Yes	Yes	Yes	Yes	Yes	Unknown	Yes	Yes	Slow
vSwarm-u [13]	Yes	Yes	Yes	Yes	Yes	Unknown	Yes	Yes	Slow
QEMU [21]	Yes	No	Yes	Yes	Yes	Yes	No	Yes	Fast
QTrace [28]	Yes	No	Yes	Yes	Yes	Unknown	No	Yes	Medium
Tejas [29]	Yes	Yes	Yes	Yes	Yes	Unknown	Yes	Partial	Unknown
MindPalace	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Medium

the slower simulation compared to functional simulators, performance simulators enable the study of the effects of micro-architecture techniques and are essential for evaluating new ideas. Many simulators combine a functional simulator with a performance simulator to gain the benefits of both.

Table I shows a comparison of popular simulators that computer architects use to perform architecture studies. Some simulators, like Pin [15] and Sniper [16] aim to understand user-level behavior of applications, but cannot measure kernel-level overhead. Many performance simulators that focus on detailed micro-architecture simulations rely on other tools to generate input traces, like ChampSim [22] and other cache simulators, like PriME [18] and Moola [30]. Some functional simulators can simulate or emulate the entire system including the OS, like SAE [27], Simics [26], QEMU [21], and other tools based on QEMU such as QTtrace [28], but do not have a performance model necessary to design novel architectures. gem5 [19], [20], at first glance seems to be a good fit for the purpose, but after putting in significant engineering effort, the complex networking and time-out issues make the OpenWhisk controller freeze and never proceed. vSwarm-u [13] runs containers in gem5, but to the best of our knowledge no one has demonstrated gem5 running a full-stack FaaS platform. A detailed comparison of vSwarm-u and MindPalace will be shown in later evaluation section. Tejas [29] does not have a publicly available version for full-system mode, thus it is unknown whether it can run full serverless framework.

In addition, the complex middleware in the FaaS platforms has made architectural simulation a difficult task. Take OpenWhisk [8] as an example, which is an open-source serverless platform that can deploy and run serverless functions in Docker containers. Fig. 2 shows the structure of OpenWhisk. The Ngnix gateway serves as the interface for users to interact with the platform, such as create, invoke, or delete functions. The Controller handles the requests from the gateway and picks one Invoker to schedule the function. Then, Kafka, a workload distributor, sends requests to the Invoker. Finally, the Invoker handles function execution in Docker containers. The Controller and the Invoker also maintain software caches, and if a cache miss happens, they will fetch data from the CouchDB database. All these components have challenged a simulator's execution environment, such as requiring support for containerization, handling networking, and virtual timing.

Designing new architectures for future cloud computing

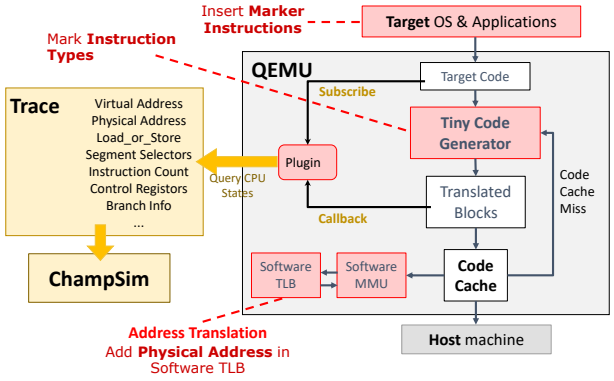


Fig. 3: MindPalace Workflow and QEMU modifications (modified parts in red). This provides a more detailed description of MindPalace than [1] based on a figure in [1]

frameworks requires a simulator that provides all the features mentioned in the Table I. Full-system simulation is necessary to understand the computing framework and the impact the kernel has on workloads. Moreover, it must use a performance model to evaluate new architectural designs. That's why MindPalace is developed to meet those requirements.

III. Simulator Design

There are two simulator models used in MindPalace. It contains a functional-level model which is based on QEMU, and a performance model which is based on ChampSim. In this manner, MindPalace can utilize both the ability of QEMU to perform full-system emulation of extensive **production scale** infrastructure and workloads (e.g., FaaS) and the ability of ChampSim to perform architecture simulation. MindPalace is constructed by modifying QEMU to generate instruction and memory access traces, then subsequently feeding the traces into ChampSim to perform the architectural simulation.

As shown in Fig. 3, the most important parts of MindPalace are QEMU components. Their responsibilities are to run a target VM and execute target applications. MindPalace gathers information for each instruction including virtual and physical addresses, segment selector state, and control register state. Utilizing this state, MindPalace outputs a compressed tracefile containing instruction and memory information.

ChampSim uses the tracefile generated by the modified QEMU as input. Because the tracefile only needs to be generated once per benchmark execution, the same tracefile can be used by multiple architecture design space explorations (multiple ChampSim runs with different parameters)

ChampSim serves as the performance model of MindPalace. ChampSim was used since it is a trace-based simulator and is widely used to study micro-architecture designs such as cache replacement policies [31]–[33], branch predictors [34]–[36], and prefetchers [37]–[39]. With ChampSim, MindPalace can enable architects to create new architectures that fit the system workload best.

A. Functional Model

1) **QEMU:** QEMU is a fast machine emulator which can emulate different CPUs (x86, ARM, RISC-V, etc.), running on different hosts (x86, PowerPC, Sparc, etc.) [21]. In addition, QEMU supports full-system emulation. A complete, unmodified operating system and applications, together called the Target, can run in the virtual machine.

The Tiny Code Generator (TCG) is the central module of QEMU, which functions as a dynamic translator. TCG translates the target code to the host code. It will fetch the target code, translate it into translated blocks (TB), store these blocks in the code cache, and send them to the host machine for execution. QEMU also has a software MMU, which handles the software TLB and performs the address translation during memory accesses.

2) **QEMU plugin:** MindPalace uses QEMU plugins to add more instrumentation during code translation. As shown in Fig. 3, for each target instruction, the QEMU plugin subscribes to the instruction and then inserts a callback function into the translated blocks. When the translated blocks are executed, the callback function will be called to perform some extra work such as counting the number of instructions. In this manner, instruction-level tracing for a full system can be realized.

However, this plugin function is not enough for micro-architecture studies. Although the plugin can gather more information about each instruction, it does not support complete memory tracing such as the type of memory access (load/store) or the corresponding physical addresses. Without the capability to obtain accessed physical addresses, the simulation is not able to model an actual cache and TLB. Similarly, the branch information is needed if we perform a branch predictor study, and also register values, like segment selectors, are required to understand whether the instruction/data access is in ring 0 (kernel space) or ring 3 (user space).

Figure 3 shows the modified components in QEMU that are there to overcome these limitations. The following subsections describe these modifications in detail.

3) **Software TLB:** QEMU only supports a software TLB that translates the virtual address (VA) of target OS and applications into the VA of the QEMU process on the host. It does not record the physical address (PA) on the target machine because QEMU will not use the PA of the target machine to access any memory, so it is not recorded in the TLB.

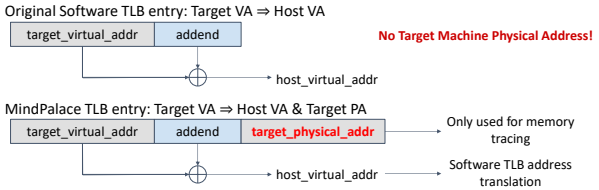


Fig. 4: Modified Software TLB

Fig. 4 shows the structure of the default software TLB entry in QEMU. The entry only contains the target VA as the tag, and an addend, which is the difference between the host virtual address and the target virtual address. When QEMU queries the software TLB, it looks up the software TLB table, finds the matched target VA, and adds on the target VA to the addend to calculate the QEMU VA.

However, MindPalace adds the target PA in the TLB entry as an extra field. When an entry is allocated for a target page, MindPalace adds a target PA field into the software TLB. Fig. 4 displays the modified TLB. The PA field is only used for memory tracing, while the original addend is used for normal QEMU address translation. In this manner, whenever there is a memory access, QEMU will query the software TLB as usual, but with a proper QEMU plugin function, it can also record the PA of the target memory accesses.

4) **TCG:** In order to obtain more functional information about an instruction such as the instruction type, checking the TLB is not sufficient. After translation by the TCG, all functional information will be hidden, and QEMU can see only the translated code. Thus, TCG needs to be modified to record instruction type information. The plugin should also add the interface for the metadata of each instruction.

5) **CPU States:** MindPalace can record whether the processor is in user space code or system (OS) code. This information is useful for analyzing how much time is being spent in the OS kernel and for differentiating user-space functions from OS code. In order to know whether the current state is in kernel or user-space, MindPalace uses segment selector and control register information.

With segment selector values, MindPalace is able to understand segment information, especially the ring level of the current instruction. There are three fields in a segment selector: bits[15:3] Index of GDT or LDT, bit[2] Table Indicator (TI) flag, and bits[1:0] Requested Privilege Level (RPL). If the values of the RPLs in the CS and DS are 0, then the system is in kernel mode. Otherwise, the values of RPLs should be 3, which is in user mode.

MindPalace captures five control registers: CR0, CR1, CR2, CR3, and CR4. One important control register value is the page table base pointer, which is stored in the CR3 register. This value is used to perform the page table walk when there is a TLB miss. MindPalace can use this value to distinguish between different processes. With CR3 value in the trace, the architecture simulation is able to clearly see the interleaving

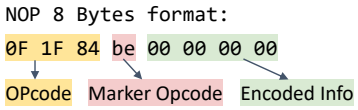


Fig. 5: Marker Instruction (Use Marker Opcode to distinguish with NOPs)

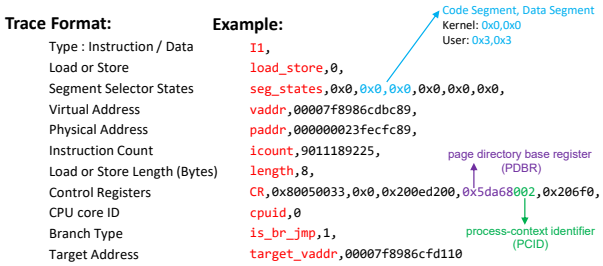


Fig. 6: Trace Format for Each Line

of different processes and help create architecture designs that are optimized for this behavior.

6) **Marker Instructions:** MindPalace adds `Marker` instructions that can be used to mark that a user-defined point has been reached. These `Marker` instructions are saved to the trace and enable finding specific functions or applications.

`Marker` instructions exploit a special NOP in the x86. MindPalace utilizes a 8-byte NOP (see Fig. 5). The first 3 bytes form the opcode of NOP, the 4th byte is the opcode of the `Marker` instruction, and the last 4 bytes are user-defined information. MindPalace supports three types of `Markers`:

- `Marker` Opcode 0xbe: Start Tracing
- `Marker` Opcode 0xed: End Tracing
- `Marker` Opcode 0xac: User-encoded Information

B. Trace

The trace is the layer separating the functional model and performance model. It is generated by the modified QEMU for each instruction executed on the VM. The trace is then modified to make the trace work with ChampSim. ChampSim then simulates the architecture being researched.

Figure 6 shows an example of the trace format. There are two types of trace lines: one is the instruction trace line that shows the VA and PA of an instruction, instruction type, the branch type, branch target address, and other register values. The other is the data access trace line which records memory access type, VA and PA of the accessed data, length of the data, as well as other CPU states.

C. Performance Model

MindPalace leverages ChampSim [22] which can model a multi-core, out-of-order CPU including existing processors and newly proposed designs. It can customize cache configures, branch predictor, and prefetchers prior to simulation.

MindPalace makes no modifications to the usage of ChampSim. It only adds a parsing step to transform the trace format from QEMU output to the format ChampSim can decode. MindPalace simplifies the trace format, reducing the extra register values that ChampSim will not utilize. Those values remain in the QEMU trace in case there is a need for them.

TABLE II: MindPalace Simulation Setup

Reference CPU	Intel(R) Core(TM) i9-12900K
OS	Ubuntu 22.04.3 LTS
QEMU version	5.0.50
VM speed	1GHz
ChampSim version	commit 29f568c
Core Number	1
ROB Size	352 entries
Scheduler size	160 entries
Dispatch Width	4 execution, 2 load. 2 store ports
Load Queue Size	128 entries
Store Queue Size	72 entries
I-TLB	128 entries 8-way, latency 1 cycles
D-TLB	64 entries 4-way, latency 1 cycles
Second-TLB	2048 entries 16-way, latency 6 cycles
L1I Cache	32KB 8-way, latency 4 cycles
L1D Cache	48KB 12-way, latency 5 cycles
L2 Cache	1.25MB 10-way, latency 13 cycles
L3 Cache	30MB 12-way, latency 33 cycles
Cache line size	64 Bytes
Replacement Policy	LRU
Prefetcher	None
Branch Predictor	256Kbits TAGE [23]
Branch Mispredict Penalty	18 cycles
Main Memory	DDR5-4800, 32GB, 2 channels

IV. Evaluation

This section presents use cases of MindPalace, demonstrating an analysis of FaaS function invocation, and showing MindPalace's ability for FaaS design evaluation.

A. Experiment Setup

The simulated CPU is kept close to a real processor. As Table II shows, we set up the buffer sizes, cache configurations, main memory latency, and other micro-architecture parameters, using the Intel i9-12900K CPU as a reference. The OS in the virtual machine runs Ubuntu 22.04.3 LTS, and OpenWhisk is deployed in the virtual machine. MindPalace uses a single core model which aligns with the default AWS Lambda instance which allocates a single vCPU for FaaS functions.

Because the QEMU virtual machine is emulation-based, which by default passes all the time stamp instructions like `rdtsc` to the host machine, the virtual time in the VM is irrelevant to the VM speed and is overestimated if the VM runs slow. This has a huge influence on timing-related container or FaaS middleware events. Thus, the experiments in MindPalace are all set to be running at the fixed frequency, 1 billion instructions per second (trying to model a 1GHz IPC=1 machine). The way to realize this is by using the 'icount' mode in QEMU to set a fixed virtual execution time for each instruction.

We use OpenWhisk actions to invoke different functions in FunctionBench [24]. A user-defined Marker instruction is inserted into functions to mark the beginning of the function. With this Marker instruction, MindPalace can decode the trace line and mark when the function gets invoked and starts.

B. Comparison of vSwarm-u and MindPalace

vSwarm-u [13] is another tool to study micro-architecture for serverless invocations. In order to compare the difference

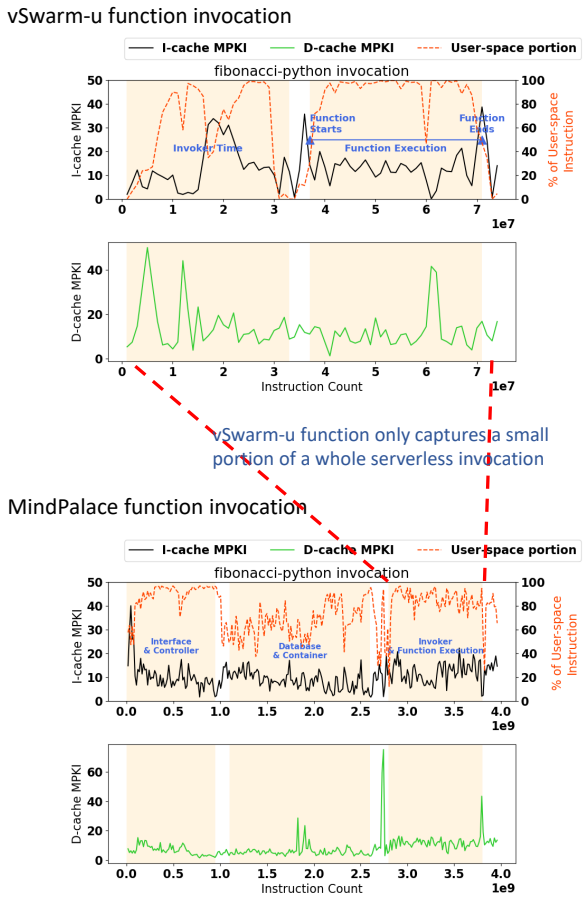


Fig. 7: Comparison of fibonacci-python function invocation in vSwarm-u and MindPalace. The vSwarm-u function only captures a small portion of a whole serverless invocation. [1]

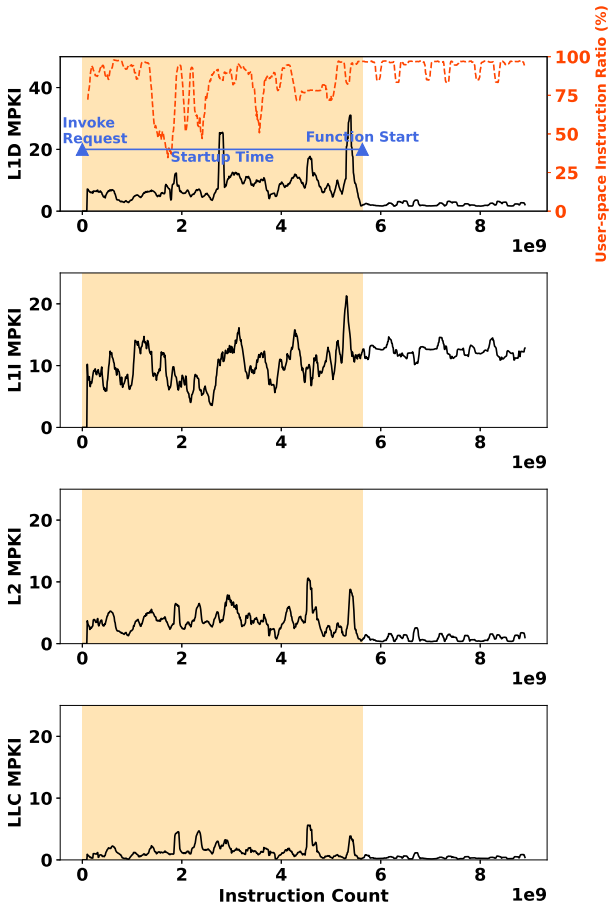


Fig. 8: 'Chameleon' Function Invocation Analysis. MindPalace successfully captures system behaviors in full FaaS framework as shown in Figure 2. Each subplot shows the cache behavior for the instructions executed. The red line in the first plot shows the percentage of user-space (ring 3) instructions. The first 5.6 billion instructions are startup time for the function.

of vSwarm-u and MindPalace we take the VM image from vSwarm-u and run the function invocation under MindPalace. Figure 7 shows the results of the same vSwarm function fibonacci-python under invocation of both simulators.

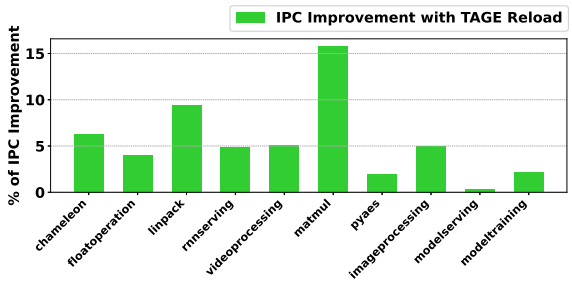
The biggest observation is that vSwarm-u invocation is much shorter than MindPalace invocation. The vSwarm-u function is 74 million instructions compared to 4 billion instructions of a full function invocation for MindPalace. There are multiple portions of the system that vSwarm-u does not model. In vSwarm-u, the invocations are only after containers have been launched, the simulator only captures RPC requests and the function runtime. While MindPalace is able to capture a whole serverless invocation including network interface, controller, database and container operations, as shown in Figure 7. In addition, the number of instructions in the function execution of MindPalace's invocation is larger than the execution section of a vSwarm-u invocation. This is because there are more processes running at the same time in a full OpenWhisk invocation, for instance the controller and OS processes. This also causes us to find a lower percentage of user-space instructions during the runtime of the function.

As a result, compared to vSwarm-u function, MindPalace is able to capture the full system components in a FaaS invocation. Although vSwarm-u is able to capture function invocations in warm containers, MindPalace is more useful in studying more complicated FaaS invocations including system interactions, which can be useful in future research.

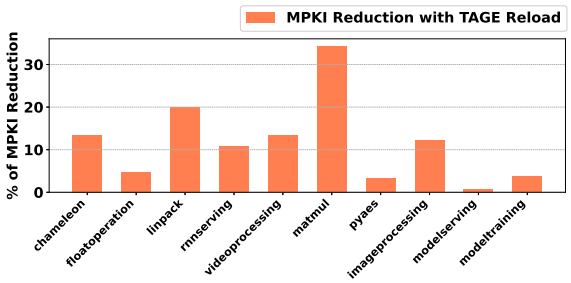
C. Function Invocation Analysis

With MindPalace, we are able to analyze a function invocation in detail. Fig. 8 presents a fine-grained study of a single 'Chameleon' function invocation. The x-axis is instruction count and is sampled every 10M instructions. The y-axis

[1]This plot is generated under the configuration of single DRAM channel.



(a) IPC Improvement. Average 5.5%.



(b) Branch Misprediction per Kilo-Instructions (MPKI) Reduction. Average 11.5%

Fig. 9: Performance improvement for first 10M instructions by reloading TAGE table states (for fast warmup).

contains cache misses and user-space instruction ratio. There are two marker events displayed in the figure to identify incoming request and the actual function start time.

As shown in Figure 8, MindPalace detects a higher usage of kernel-space instructions before the start of the function. During the startup time, OpenWhisk makes many system calls to handle the request from Nginx, to distribute serverless actions made by the controller and the distributor, to load code and data from the database, and to start Docker containers for function execution.

From that experiment, we also see that all caches have slightly higher miss rates before the function starts. This is because, first, the function needs to load instructions and data due to cold start, and also the function working set size is usually larger than L1 cache, causing a higher L1 cache miss (The first 5.6 billion instructions).

These findings confirm that cold function invocations for serverless functions have poor locality for cache state.

D. Branch Predictor Exploration

A key feature of MindPalace is to enable the study and evaluation of architectures that target FaaS systems. This section showcases a branch predictor design for FaaS systems.

Serverless functions face cold micro-architecture states when they is frequent switching. Thus there is a need to quickly warm branch predictors and other micro-architectural state. The branch predictor in particular needs time to warm-up [10], [12].

Restoring the context for micro-architectural states is a potential solution to this challenge. It can be achieved by recording branch predictor history tables and reloading them at the start of subsequent invocations. With the MindPalace framework, we are able to demonstrate the benefits of such a design.

In the first invocation of a function, we store all branch predictor tables (all TAGE banks) after 10M instructions. At this point, TAGE has been filled and is able to handle many cold misses. Then, before the same function is invoked, the tables are reloaded, thus functions start to execute under a warmed branch predictor.

MindPalace shows the performance gain for the first 10M instructions, demonstrating a faster warmup, by simulating the whole design and the results are displayed in Figure 9. This technique of reloading TAGE tables can improve IPC by an average of 5.5% across those 10M instructions with a peak improvement of 15.8%. The branch outcome misprediction can be reduced by an average of 11.5% with a peak improvement of 33%. These results can motivate further studies of better branch predictor designs.

V. Validation

This section extends previous analysis of MindPalace's accuracy [1] and presents a more detailed verification of both functional and performance models. The accuracy of ChampSim has not be deeply analyzed. To address this, we compare real hardware against ChampSim configured in the same manner as the hardware.

A. Functional Model Verification

MindPalace makes multiple modifications to QEMU. In this section we discuss how we verify the changes, especially the software TLB modification for target physical addresses, the most significant functional change in QEMU.

To verify the target physical address translation, MindPalace checks the kernel `pagemap` and compares the mappings with the generated trace. In the guest Linux kernel, `/proc/PID/pagemap` holds the VA to PA mappings for all the processes. MindPalace takes this as ground truth to verify the address translation. Every trace line, instruction trace, or data access, will record the VA and the corresponding PA. MindPalace takes this VA, queries the mapping in the guest kernel `pagemap`, and compares the physical page number in `pagemap` with the PA value in the trace. Using this methodology we verified MindPalace's address translation generation using three different processes.

Other verification tests are also conducted to check the correctness of MindPalace additions. The target addresses in direct-jump/direct-call instructions are verified by checking the following executed instruction address. The target address should be identical to the following instruction address except for a corner case that an interrupt happens on that direct-jump/direct-call instruction. This rarely happens and can be verified by checking whether the next instruction is an entry point of an interrupt handler.

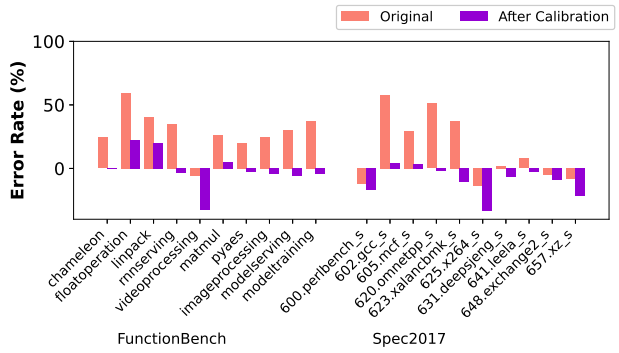


Fig. 10: IPC Error Rate before Calibration (left bars in each benchmark) and after Calibration (right bars in each benchmark). A positive error rate means the simulated IPC result is higher than real hardware IPC, while a negative error rate means the simulated IPC result is lower than real hardware IPC. 26.4% is the average of the absolute value of error rates without calibration and 10.6% is the average with calibration.

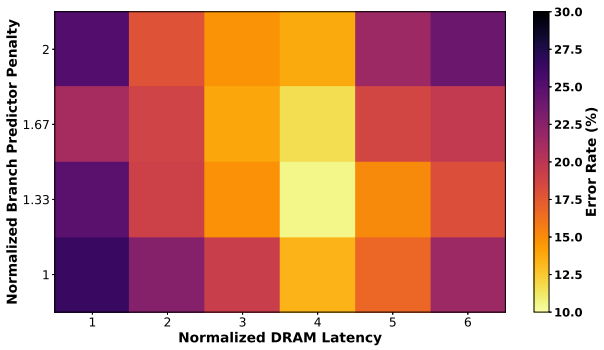


Fig. 11: IPC Average Absolute Value of Error Rates using Different Error Correction Factors.

B. Performance Model Verification

To verify the performance model, we conduct cache miss, branch miss, and IPC analysis on FunctionBench [24] and the Integer benchmarks from SPEC 2017 [25]. We compare the simulator results with that of a real machine.

In order to match simulation with real hardware, we use a real Intel i9-12900K processor running Ubuntu 22.04.5. The active core number in the real machine is set to one, and hyper-threading and hardware prefetching are turned off. Turbo boosting is also turned off to keep a fixed 1GHz CPU frequency to reduce the discrepancy of time stamps between MindPalace and the real machine. Each benchmark is executed three times on real hardware, and the hardware performance is recorded using hardware performance counters and the Perf tool [40]. The results are the average of the three tests. Instruction count is used to align the starting point of real machine experiments with MindPalace simulations and lengths are cut to be the same. The SPEC benchmark runs are specifically for the purpose of error analysis of MindPalace, and they are not compliant with standard SPEC reporting rules.

During the QEMU emulation, MindPalace skips the first 9 billion instructions to ensure the benchmark has started and records hot regions. For each benchmark, we then record one billion instructions. MindPalace configures ChampSim close to a real machine (Table II).

A previous study [41] has shown full-system simulations can have a significant error rate compared to performance counters on real hardware. Even cycle-accurate simulators like gem5 can experience over 100% mismatch in L1D cache misses for some benchmarks and over 1000% mismatch in LLC misses for some benchmarks [41]. Also, performance counters may not be deterministic and include variances [42]. Although validating a full-system simulator with performance counters is not easy, we present our attempts to compare the results, analyze the errors, and derive calibration of MindPalace to real hardware.

*1) **IPC Analysis and Error Correction**:* As far as we know, no prior ChampSim studies have provided detailed validation of ChampSim against real hardware. Thus in order to achieve better fidelity, this work explores in detail error calibration factors for ChampSim parameters to calibrate simulator-hardware differences.

We investigated what parameters most affect ChampSim, matching real hardware in terms of performance/IPC. We found that branch misprediction penalty and DRAM latency have the greatest impact on matching real hardware. Figure 10 shows the IPC error rate when compared to real hardware and how much MindPalace can successfully be corrected. If without any calibration, original ChampSim configurations have a significant error in performance estimation. The average of the absolute value of error rates across all benchmarks is 26.4%, as depicted by the orange bars (left bars) of each group in Figure 10.

The mismatches are majorly contributed to by the benchmarks that exhibit high cache misses per kilo-instructions (MPKI) in the LLC (see Fig. 12), which means higher DRAM throughput. The latency of DRAM in the simulation was set using the DDR5's datasheets, setting tRP = tRCD = tCAS = 40ns. This is likely an underestimate of DRAM latency without full interconnect modeling.

The implementation of ChampSim's frontend is not exactly the same as a real processor. ChampSim stalls the frontend pipeline to pay misprediction penalty rather than fetching the wrong path instructions and recovering afterwards. Because the wrong path fetching may pollute cache or other micro-architecture structures, this makes ChampSim underestimate branch misprediction penalty.

Fig. 11 shows the heatmap of error rate under different combinations of branch misprediction penalty and DRAM latency. Branch misprediction penalty and DRAM latency are scaled to different factors to see which one provides the best IPC match. Using a branch predictor factor 1.33 with DRAM latency factor of 4 permits the best average IPC match, which is 10.6%. This is a better performance match, and the error

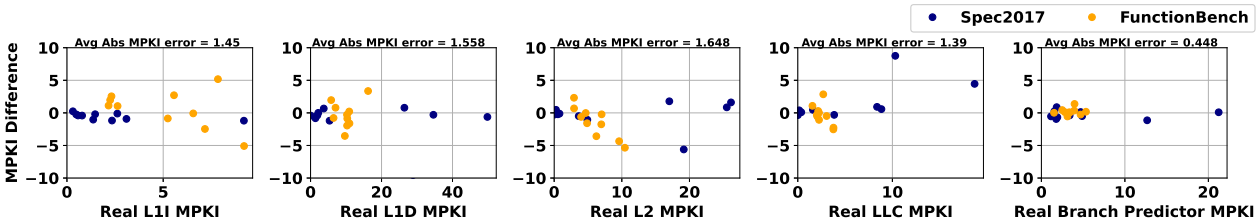


Fig. 12: Cache Misses Per Kilo-Instruction (MPKI) and Branch Predictor Mispredictions Per Kilo-Instruction (MPKI) Validation against Real Machine. Each figure presents a scatter plot of the MPKI difference of FunctionBench and Integer benchmarks from SPEC 2017, comparing MindPalace results to those obtained from real hardware. The x-axis represents the MPKI values obtained from real hardware, while the y-axis displays the MPKI difference.

correction factors' calibration of parameters comply with the above analysis. Those error correction factors are applied for the rest of the validation and previous evaluations.

Although different processors may require different calibration factors, the same methods can be used to calibrate MindPalace to them.

*2) **Cache Miss Analysis**:* Fig. 12 shows the cache MPKI difference for different levels of the cache hierarchy using the benchmarks and methodology described above. We define the MPKI difference as $MPKI_{difference} = MPKI_{MindPalace} - MPKI_{Real}$

The absolute average MPKI error for each level of cache are 1.45 for the L1I cache, 1.558 for the L1D cache, 1.648 for the unified L2 cache, and 1.39 for the LLC. We use instruction count to align the starting point of the simulation and real machine. Due to different scheduling and interrupts, the alignment between simulator and real hardware may not be exactly the same impacting accuracy.

*3) **Branch Miss Analysis**:* Using the same benchmarks, we verify MindPalace with a real branch predictor. The right-most plot in Figure 12 compares the branch predictor mispredictions per kilo-instruction (MPKI) of the real hardware versus a 256Kbits TAGE simulation. The average absolute MPKI difference is 0.448 across all of the benchmarks.

One challenge for validation of branch predictors is that the real Intel hardware likely has a different branch predictor design than stock TAGE with some extra, case specific predictions. Our results show that the 256Kbits TAGE in MindPalace is close to real Intel hardware.

VI. Related Work

Studies of Micro-architectural Limitations in Cloud Computing Frameworks. Multiple prior works focus on the architectural behavior of emerging cloud workloads and identify the shortcomings of server processor architectures. CloudSuite was introduced to study the inefficiencies in scale-out workloads and identified as critical issues the high miss rate in the instruction cache, the large data working sets, the low parallelism, and the limited processor bandwidth [43]. Jia *et al.* [44] present a study of data analysis workloads in the data center and show implications for L1I cache, instruction TLB, and branch predictor designs. Chen *et al.* [45] characterize event-driven applications and study how the number of cores and LLC capacity influences tail-latency. The study of warehouse-scale (WSC) workloads [46] reveals the diverse characterizations of WSC and highlights common issues in processor front-end stalls, and data cache misses. Recently we have seen the rise of research in microservices, the long front-end stalls were discovered by [47], and benchmarks were created to study these microservices [9], [48]. Function-as-a-Service (FaaS) has also become increasingly popular. Shahrad *et al.* [10] identify the decrease in IPC, accuracy degradation in branch predictor, as well as higher bandwidth requirement in different FaaS workloads. Traces [6] and benchmarks [49] [50] have been provided to study the system behavior. There are some works on architecture designs for FaaS systems such as instruction prefetching [13], branch target buffer (BTB) and bimodal predictor prefetching [12], and micro-architecture partitioning [14].

Tools for Cloud Architecture Research. Researchers from academia and industry have created many tools to understand workloads and current architectures. One typical method is to obtain system-level action traces from cloud vendors and analyze the bottlenecks and performance in existing commercial servers, such as Facebook [51], Google [52] and Microsoft [6]. Although this method can evaluate current system performance, it hides much of the micro-architecture details, making it difficult to understand the architectural characteristics. Some tools utilize hardware performance counters, such as the Linux Perf tool [40] and Intel vTune [53]. These tools only reveal architectural implications and limitations on existing processors in aggregate and cannot be used to design new architectures. While tools like Pin [15] can capture instruction-level traces without kernel instructions and cannot be used to study full systems. Other dynamic translation tools such as QEMU [21] and Valgrind [54] can emulate a full system without modifications and potentially be used to do full-system studies. However, they miss the essential performance model to simulate a micro-architecture.

Performance simulators can be split into two categories: execution-driven and trace-driven. Execution-driven simulators include ZSim [17] and Graphite [55], where target benchmarks can be executed without modifications. The second category, trace-driven simulators, includes frameworks such as McSimA+ [56], Tejas [29] and ChampSim [22] that take input instruction traces and can model the entire system, while

frameworks like Moola [30] take memory access traces and focus on a subset of the computer system's operation.

Many simulators manage to combine a functional model with a performance model. gem5 [19], [20] combines both a functional and cycle-accurate performance model. Even though gem5 can perform full-system simulation, it has networking and timing difficulties when deploying full FaaS platforms. Schall *et al.* [13] present an attempt to run containers on gem5 but are still simply invoking containers with RPC, not a full FaaS software stack, which sets it apart from this work which runs the OpenWhisk [8] FaaS framework. Flexus [57] takes Simics [26] as the functional model, and does statistical sampling for architecture simulations. However, Flexus has not been updated since 2012 and is not used in emerging micro-architecture study. The same issue happened for MARSSx86 [58] and QTrace [28], which are no longer significantly maintained.

VII. Conclusion

This paper expands on an extended poster abstract [1] on MindPalace by detailing the inner working of MindPalace. MindPalace is an open-source, full-stack, simulation framework targeting computer architecture research for FaaS workloads.

We compared MindPalace qualitatively to prior tools including vSwarm-u, we showed how MindPalace can be used to explore function startup, and how it can be used to explore complex system behaviors when running full-stack FaaS workloads. This work demonstrated MindPalace being used for an example architecture research through a study of reloading branch predictor state. Finally, this work investigates validating and calibrating the performance of MindPalace.

In conclusion, we see an exciting future for FaaS and computer architectures for FaaS. We see MindPalace and this paper as laying the groundwork for future FaaS-targeted computer architectures.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Award No. CCF-1822949. This material is based on research sponsored by Air Force Research Laboratory (AFRL) and Defense Advanced Research Projects Agency (DARPA) under agreement No. 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] K. Xu, G. Tziantzioulis, and D. Wentzlaff, "MindPalace: A framework for studying microarchitecture design of function-as-a-service," in *2024 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2024, pp. 313–315.

[2] Amazon, "Aws lambda," https://aws.amazon.com/lambda/.

[3] Microsoft, "Azure functions," https://azure.microsoft.com/en-us/services/functions.

[4] Google, "Google cloud functions," https://cloud.google.com/functions.

[5] Z. Jia and E. Witchel, "Nightcore: efficient and scalable serverless computing for latency-sensitive, interactive microservices," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021, pp. 152–166.

[6] M. Shahrad, R. Fonseca, I. Goiri, G. Chaudhry, P. Batum, J. Cooke, E. Laureano, C. Tresness, M. Russinovich, and R. Bianchini, "Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider," in *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, Jul. 2020, pp. 205–218. [Online]. Available: https://www.usenix.org/conference/atc20/presentation/shahrad

[7] A. Sriraman and T. F. Wenisch, "µTune: Auto-Tuned threading for OLDI microservices," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. Carlsbad, CA: USENIX Association, Oct. 2018, pp. 177–194. [Online]. Available: http://www.usenix.org/conference/osdi18/presentation/sriraman

[8] Apache Software Foundation, "Apache openwhisk," https://openwhisk.apache.org., Accessed: 2022-07-15.

[9] Y. Gan, Y. Zhang, D. Cheng, A. Shetty, P. Rathi, N. Katarki, A. Bruno, J. Hu, B. Ritchken, B. Jackson, K. Hu, M. Pancholi, Y. He, B. Clancy, C. Colen, F. Wen, C. Leung, S. Wang, L. Zaruvinsky, M. Espinosa, R. Lin, Z. Liu, J. Padilla, and C. Delimitrou, "An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud and Edge Systems," in *Proceedings of the Twenty Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, April 2019.

[10] M. Shahrad, J. Balkind, and D. Wentzlaff, "Architectural implications of function-as-a-service computing," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '52. New York, NY, USA: Association for Computing Machinery, 2019, p. 1063–1075. [Online]. Available: https://doi.org/10.1145/3352460.3358296

[11] Y. Zhu, D. Richins, M. Halpern, and V. J. Reddi, "Microarchitectural implications of event-driven server-side web applications," in *Proceedings of the 48th International Symposium on Microarchitecture*, 2015, pp. 762–774.

[12] D. Schall, A. Sandberg, and B. Grot, "Warming up a cold front-end with ignite," in *56th IEEE/ACM International Symposium on Microarchitecture*, 2023.

[13] D. Schall, A. Margaritov, D. Ustiugov, A. Sandberg, and B. Grot, "Lukewarm serverless functions: Characterization and optimization," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ser. ISCA '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 757–770.

[14] J. Stojkovic, E. Choukse, E. Saurez, Í. Goiri, and J. Torrellas, "Mosaic: Harnessing the micro-architectural resources of servers in serverless environments," in *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2024, pp. 1397–1412.

[15] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, "Pin: building customized program analysis tools with dynamic instrumentation," in *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '05. Association for Computing Machinery, 2005, p. 190–200.

[16] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, 2011, pp. 1–12.

[17] D. Sanchez and C. Kozyrakis, "Zsim: Fast and accurate microarchitectural simulation of thousand-core systems," *ACM SIGARCH Computer architecture news*, vol. 41, no. 3, pp. 475–486, 2013.

[18] Y. Fu and D. Wentzlaff, "Prime: A parallel and distributed simulator for thousand-core chips," in *2014 IEEE International Symposium on*

Performance Analysis of Systems and Software (ISPASS), 2014, pp. 116–125.

[19] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, p. 1–7, aug 2011. [Online]. Available: https://doi.org/10.1145/2024716.2024718

[20] J. Lowe-Power, A. M. Ahmad, A. Akram, M. Alian, R. Amslinger, M. Andreozzi, A. Armejach, N. Asmussen, B. Beckmann, S. Bharadwaj, G. Black, G. Bloom, B. R. Bruce, D. R. Carvalho, J. Castrillon, L. Chen, N. Derumigny, S. Diestelhorst, W. Elsasser, C. Escuin, M. Fariborz, A. Farmahini-Farahani, P. Fotouhi, R. Gambord, J. Gandhi, D. Gope, T. Grass, A. Gutierrez, B. Hanindhito, A. Hansson, S. Haria, A. Harris, T. Hayes, A. Herrera, M. Horsnell, S. A. R. Jafri, R. Jagtap, H. Jang, R. Jeyapaul, T. M. Jones, M. Jung, S. Kannoth, H. Khaleghzadeh, Y. Kodama, T. Krishna, T. Marinelli, C. Menard, A. Mondelli, M. Moreto, T. Mück, O. Naji, K. Nathella, H. Nguyen, N. Nikoleris, L. E. Olson, M. Orr, B. Pham, P. Prieto, T. Reddy, A. Roelke, M. Samani, A. Sandberg, J. Setoain, B. Shingarov, M. D. Sinclair, T. Ta, R. Thakur, G. Travaglini, M. Upton, N. Vaish, I. Vougioukas, W. Wang, Z. Wang, N. Wehn, C. Weis, D. A. Wood, H. Yoon, and Éder F. Zulian, "The gem5 simulator: Version 20.0+," 2020. [Online]. Available: https://arxiv.org/abs/2007.03152

[21] F. Bellard, "QEMU, a fast and portable dynamic translator." in *USENIX annual technical conference, FREENIX Track*, vol. 41, no. 46. California, USA, 2005, pp. 10–5555.

[22] N. Gober, G. Chacon, L. Wang, P. V. Gratz, D. A. Jimenez, E. Teran, S. Pugsley, and J. Kim, "The championship simulator: Architectural simulation for education and competition," 2022. [Online]. Available: https://arxiv.org/abs/2210.14324

[23] A. Seznec and P. Michaud, "A case for (partially) tagged geometric history length branch prediction," *The Journal of Instruction-Level Parallelism*, vol. 8, p. 23, 2006.

[24] J. Kim and K. Lee, "Functionbench: A suite of workloads for serverless cloud function service," in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. IEEE, 2019, pp. 502–504.

[25] Standard Performance Evaluation Corporation, "SPEC CPU® 2017 Benchmark Suite," https://www.spec.org/cpu2017, 2017.

[26] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A full system simulation platform," *Computer*, vol. 35, no. 2, pp. 50–58, 2002.

[27] N. Chachmon, D. Richins, R. Cohn, M. Christensson, W. Cui, and V. J. Reddi, "Simulation and analysis engine for scale-out workloads," in *Proceedings of the 2016 International Conference on Supercomputing*, ser. ICS '16. New York, NY, USA: ACM, 2016, pp. 22:1–22:13. [Online]. Available: http://doi.acm.org/10.1145/2925426.2926293

[28] X. Tong and A. Moshovos, "Qtrace: a framework for customizable full system instrumentation," in *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2015, pp. 245–255.

[29] S. R. Sarangi, R. Kalayappan, P. Kallurkar, S. Goel, and E. Peter, "Tejas: A java based versatile micro-architectural simulator," in *2015 25th International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*. IEEE, 2015, pp. 47–54.

[30] C. F. Shelor and K. M. Kavi, "Moola: Multicore cache simulator," in *30th International Conference on Computers and Their Applications CATA-2015*, 2015.

[31] A. Jain and C. Lin, "Hawkeye: Leveraging belady's algorithm for improved cache replacement," in *The 2nd Cache Replacement Championship (CRC-2 Workshop in ISCA 2017)*, 2017.

[32] V. Young, C.-C. Chou, A. Jaleel, and M. Qureshi, "Ship++: Enhancing signature-based hit predictor for improved cache performance," in *The 2nd Cache Replacement Championship (CRC-2 Workshop in ISCA 2017)*, 2017.

[33] P. Faldu and B. Grot, "Reuse-aware management for last-level caches," in *International Workshop on Cache Replacement Championship (CRC), co-located with ISCA*, 2017.

[34] A. Seznec, "A 256 kbits l-tage branch predictor," *Journal of Instruction-Level Parallelism (JILP) Special Issue: The Second Championship Branch Prediction Competition (CBP-2)*, vol. 9, pp. 1–6, 2007.

[35] Y. Ishii, "Fused two-level branch prediction with ahead calculation," *Journal of Instruction-Level Parallelism*, vol. 9, pp. 1–19, 2007.

[36] H. Gao and H. Zhou, "Pmpm: Prediction by combining multiple partial matches," *Journal of Instruction-Level Parallelism*, vol. 9, pp. 1–18, 2007.

[37] S. Pakalapati and B. Panda, "Bouquet of instruction pointers: Instruction pointer classifier-based spatial hardware prefetching," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 118–131.

[38] A. Ros and A. Jimborean, "A cost-effective entangling prefetcher for instructions," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 99–111.

[39] G. Vavouliotis, L. Alvarez, V. Karakostas, K. Nikas, N. Koziris, D. A. Jiménez, and M. Casas, "Exploiting page table locality for agile tlb prefetching," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 85–98.

[40] "Perf," https://perf.wiki.kernel.org/index.php/Main_Page.

[41] A. Akram and L. Sawalha, "A survey of computer architecture simulation techniques and tools," *IEEE Access*, vol. 7, pp. 78 120–78 145, 2019.

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

[43] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi, "Clearing the clouds: a study of emerging scale-out workloads on modern hardware," in *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS XVII. Association for Computing Machinery, 2012, p. 37–48.

[44] Z. Jia, L. Wang, J. Zhan, L. Zhang, and C. Luo, "Characterizing data analysis workloads in data centers," in *2013 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2013, pp. 66–76.

[45] S. Chen, S. GalOn, C. Delimitrou, S. Manne, and J. F. Martinez, "Workload characterization of interactive cloud services on big and small server platforms," in *2017 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2017, pp. 125–134.

[46] S. Kanev, J. P. Darago, K. Hazelwood, P. Ranganathan, T. Moseley, G.-Y. Wei, and D. Brooks, "Profiling a warehouse-scale computer," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, 2015, pp. 158–169.

[47] Y. Gan and C. Delimitrou, "The architectural implications of cloud microservices," *IEEE Computer Architecture Letters*, vol. 17, no. 2, pp. 155–158, 2018.

[48] A. Sriraman and T. F. Wenisch, "μ suite: a benchmark suite for microservices," in *2018 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2018, pp. 1–12.

[49] M. Copik, G. Kwasniewski, M. Besta, M. Podstawski, and T. Hoefler, "Sebs: A serverless benchmark suite for function-as-a-service computing," in *Proceedings of the 22nd International Middleware Conference*, 2021, pp. 64–78.

[50] T. Yu, Q. Liu, D. Du, Y. Xia, B. Zang, Z. Lu, P. Yang, C. Qin, and H. Chen, "Characterizing serverless platforms with serverlessbench," in *Proceedings of the 11th ACM Symposium on Cloud Computing*, 2020, pp. 30–44.

[51] M. Chow, D. Meisner, J. Flinn, D. Peek, and T. F. Wenisch, "The mystery machine: End-to-end performance analysis of large-scale internet services," in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, 2014, pp. 217–231.

[52] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Heterogeneity and dynamicity of clouds at scale: Google trace analysis," in *Proceedings of the third ACM symposium on cloud computing*, 2012, pp. 1–13.

[53] J. Reinders, *VTune performance analyzer essentials*. Intel Press Santa Clara, 2005.

[54] N. Nethercote and J. Seward, "Valgrind: a framework for heavyweight dynamic binary instrumentation," in *Proceedings of the 28th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '07. Association for Computing Machinery, 2007, p. 89–100.

[55] J. E. Miller, H. Kasture, G. Kurian, C. Gruenwald, N. Beckmann, C. Celio, J. Eastep, and A. Agarwal, "Graphite: A distributed parallel simulator for multicores," in *HPCA-16 2010 The Sixteenth International*

Symposium on High-Performance Computer Architecture. IEEE, 2010, pp. 1–12.

[56] J. H. Ahn, S. Li, S. O, and N. P. Jouppi, "Mcsima+: A manycore simulator with application-level+ simulation and detailed microarchitecture modeling," in *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2013, pp. 74–85.

[57] T. F. Wenisch, R. E. Wunderlich, M. Ferdman, A. Ailamaki, B. Falsafi, and J. C. Hoe, "Simflex: statistical sampling of computer system simulation," *IEEE Micro*, vol. 26, no. 4, pp. 18–31, 2006.

[58] A. Patel, F. Afram, S. Chen, and K. Ghose, "Marss: A full system simulator for multicore x86 cpus," in *2011 48th ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2011, pp. 1050–1055.