# Accelerating Extra Dimensional Page Walks for Confidential Computing

Dong Du
Institute of Parallel and Distributed Systems, Shanghai Jiao Tong University
Engineering Research Center for Domain-specific Operating Systems (MoE)
Shanghai, China
dd_nirvana@sjtu.edu.cn

Bicheng Yang
Institute of Parallel and Distributed Systems, Shanghai Jiao Tong University
Engineering Research Center for Domain-specific Operating Systems (MoE)
Shanghai, China
bichengyang@sjtu.edu.cn

Yubin Xia
Institute of Parallel and Distributed Systems, Shanghai Jiao Tong University
Shanghai AI Laboratory
Engineering Research Center for Domain-specific Operating Systems (MoE)
Shanghai, China
xiayubin@sjtu.edu.cn

Haibo Chen
Institute of Parallel and Distributed Systems, Shanghai Jiao Tong University
Engineering Research Center for Domain-specific Operating Systems (MoE)
Shanghai, China
haibochen@sjtu.edu.cn

## ABSTRACT

To support highly scalable and fine-grained computing paradigms such as microservices and serverless computing better, modern hardware-assisted confidential computing systems, such as Intel TDX and ARM CCA, introduce *permission table* to achieve fine-grained and scalable memory isolation among different domains. However, it also adds an extra dimension to page walks besides page tables, leading to significantly more memory references (e.g., 4→12 for RISC-V Sv39)[1]. We observe that most costs (about 75%) caused by the extra dimension of page walks are used to validate page table pages. Based on this observation, this paper proposes **HPMP** (Hybrid Physical Memory Protection), a hardware-software co-design (on RISC-V) that protects page table pages using segment registers and normal pages using permission tables to balance scalability and performance. We have implemented HPMP and Penglai-HPMP (a TEE system based on HPMP) on FPGA with two RISC-V cores (both in-order and out-of-order). Evaluation results show that HPMP can reduce costs by 23.1%–73.1% on BOOM and significantly improve performance on real-world applications, including serverless computing (FunctionBench) and Redis.

[1]All memory reference numbers presented in this paper adhere to the RISC-V ISA specification [105] and do not take into account PWC or other micro-architecture optimizations that could potentially bypass page table pages.


*MICRO '23, October 28-November 1, 2023, Toronto, ON, Canada*

ACM ISBN 979-8-4007-0329-4/23/10…$15.00
https://doi.org/10.1145/3613424.3614293

**ACM Reference Format:**
Dong Du, Bicheng Yang, Yubin Xia, and Haibo Chen. 2023. Accelerating Extra Dimensional Page Walks for Confidential Computing. In *56th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '23), October 28-November 1, 2023, Toronto, ON, Canada.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3613424.3614293

## 1 INTRODUCTION

There are two trends in cloud computing. First, there has been a surge of interest in using confidential computing [1, 2, 9, 53, 61, 70, 80] to host security-sensitive applications in the cloud without trusting the cloud providers [20, 27, 34, 41, 43, 48, 62, 88, 93, 98]. Second, microservices [40] and serverless computing [60, 67, 104] have emerged as new paradigms in the cloud. These paradigms use single-purpose services or functions as the basic computation unit and can achieve more than 100 instances per node [50, 51].

However, traditional confidential computing solutions like Intel SGX [80] and ARM Trustzone [61] are not well-suited to support such highly scalable and fine-grained computing paradigms because of their segment-based physical memory isolation [47, 53, 70, 105]. Segment-based isolation [47, 53, 70, 105] uses registers in the CPU to manage memory regions and their permissions. Although the checking is efficient, this approach has scalability and granularity issues as the CPU can only support a limited number of segment registers due to constrained hardware resources. For example, Intel SGX relies on Processor Reserved Memory (PRM)[47] (one region) to isolate enclave's data and code from untrusted software. However, it leads to orders of magnitude slowdown when the fixed-size PRM is insufficient[93]. RISC-V TEE (Trusted Execution Environment) systems like Keystone [70] use RISC-V PMP [105] for physical memory isolation, which can only support <16 domains.

To overcome these limitations, hardware vendors nowadays tend to use a dedicated table, called *permission table* in this paper, to manage the permissions of physical memory in a fine-grained manner, usually at the page level. A permission table is similar

to a page table. However, instead of translating a virtual address to a physical address, the table receives a physical address and returns a permission (e.g., read/write/execute) to assist the CPU in determining whether an access is valid. For example, ARM proposes CCA [9] as its next-generation confidential computing system. The key difference between Trustzone and CCA is the *Granule Protection Table (GPT)*[7], a permission table for fine-grained and configurable granularity. Similarly, Intel proposes TDX that utilizes a Physical Address Metadata Table (PAMT)[13] to record permissions of all secure memory. AMD SEV-SNP [1] leverages a Reverse Map Table (RMP) [8] to record secure page metadata. By using permission tables, hardware vendors can achieve fine-grained and scalable memory isolation, which is suitable for highly scalable and fine-grained computing paradigms such as serverless computing.

However, the use of permission tables can add an additional dimension of page walks (hardware or software), besides page table and nested page table, which can negatively impact memory access performance. For example, a 2-level permission table leads to eight more memory references (total: 12 references) for RISC-V Sv39 (3-level page table). This issue is even more serious for 4-level or 5-level page table architectures, such as RISC-V Sv48 and Sv57 [105], Intel 5-level paging and EPT [12]. This trend, *from segment-based to table-based isolation*, raises an important question for architecture and system designers: *How can we achieve fine-grained physical memory isolation without sacrificing performance?*

The costs caused by permission tables come from two sources: (1) validating the physical address of page table (PT) pages during page table walks, and (2) validating the physical address of data/instructions. However, we observe that most memory references and costs are from the first source. For example, 75% (6 out of 8) of introduced memory references are for checking the permissions of three page table pages in RISC-V Sv39. Based on this observation, this paper proposes **HPMP** (Hybrid Physical Memory Protection), a hardware-software co-design (on RISC-V) that harmonizes the benefits of segment and table-based isolation. Specifically, HPMP protects page table pages (managed by the OS kernel) using segments for fast checking and data pages using permission tables to support highly scalable and fine-grained computing paradigms like serverless computing.

To achieve this goal, HPMP includes two key technical contributions. First, we propose a new ISA extension called *PMP Table*. PMP Table is a new RISC-V physical memory isolation design that harmonizes the benefits of both segment and permission tables. Unlike prior systems that use a hybrid design of segment and table for address translation, such as direct segment [33, 55, 59, 68] and RMM [64], PMP Table is designed to support both segment and table-based isolation *without* duplicating hardware structures. To this end, we borrow the idea of *huge page* and think of a segment as the huge page of a permission table. Specifically, PMP Table can use a permission table's registers to save the whole region's permission, which is the same as segment registers. This enables significant flexibility for both hardware and software. In hardware, our prototype introduces zero new registers and instructions and simply reuses the reserved bits in existing RISC-V PMP registers. In software, HPMP can dynamically switch between segment and table-based isolation or even use them simultaneously, achieving high flexibility.

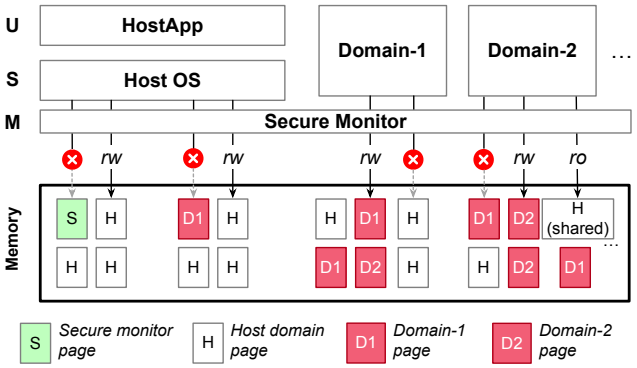



**Figure 1: Confidential systems with memory isolation.** ***Most RISC-V TEEs [11, 52, 53, 70] utilize a secure monitor running on machine mode (denoted as M) to isolate domains using a physical memory isolation mechanism (e.g., PMP).***

Second, we propose Penglai-HPMP, a new confidential system based on Penglai Enclave [15, 53], which includes a secure monitor (software TCB in TEE) and OS kernel. A key design in Penglai-HPMP is the general memory segment (GMS) abstraction, which represents a continuous memory region with the same permission and a label ("fast" or "slow"). The OS kernel is extended to organize PT pages in GMSs with "fast" labels. Penglai-HPMP then isolates "fast" GMSs using segments while using permission tables for others. As a result, Penglai-HPMP can reduce the memory references from 12 to 6 for RISC-V Sv39. This reduction in memory references can significantly improve the performance of highly scalable and fine-grained computing paradigms.

We have implemented HPMP and Penglai-HPMP on an FPGA using two open-source RISC-V cores, BOOM (out-of-order core) [114] and RocketCore (in-order core) [28]. Our evaluation results show that HPMP can effectively mitigate the costs of extra-dimensional walks, reducing costs by 23.1%–73.1% on BOOM. For real-world applications, Penglai with permission table incurs a performance slowdown of 5.5%–20.3% and 10.8%–31.8% for serverless applications (FunctionBench) and Redis, respectively, compared with PMP-based isolation, while Penglai-HPMP can reduce the costs to 3.5% and 4.5% on average on BOOM. We also explore a similar idea in a virtualized environment and analyze how HPMP works in a fragmented environment and how optimizations like PWC [36] can further improve performance.

## 2 MOTIVATION

### 2.1 Confidential Systems

Figure 1 shows the architecture of confidential systems on RISC-V, which is also representative of other ISAs. Typically, the only software trusted computing base (TCB) in confidential systems is the *secure monitor*, a small software component running in the most privileged mode (e.g., machine mode in RISC-V) [53, 70]. The secure monitor is responsible for isolating physical memory among different domains by utilizing specific hardware mechanisms.

The secure monitor can partition the system into isolated *domains*. Each domain has its private physical memory regions and

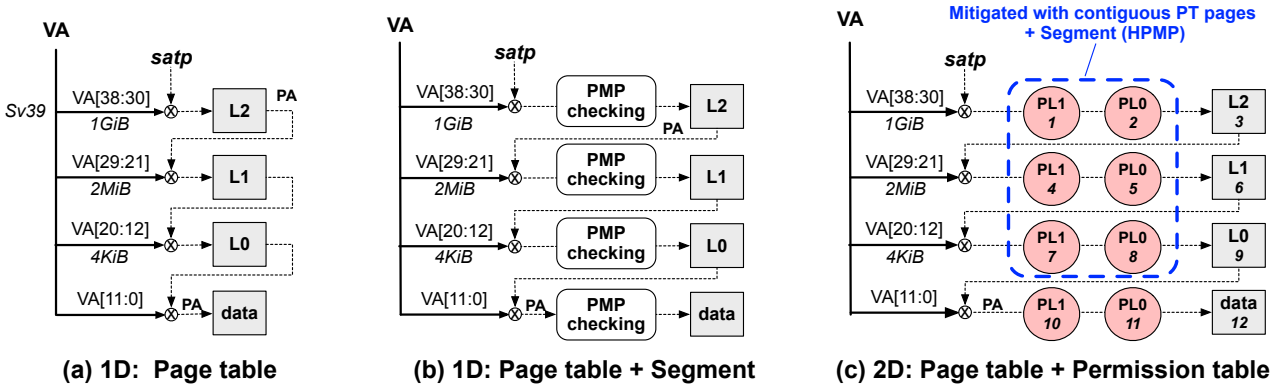



**Figure 2: Physical memory isolation schemes based on RISC-V.** ***(a) Memory access in a traditional way. (b) Memory access with segment-based isolation. (c) Memory access with table-based isolation. We use RISC-V PMP as the segment-based isolation, and a generic permission table (similar to ARM GPT) as the table-based isolation. Squares and circles represent memory references. L2, L1, L0 represent page tables from root to leaf, and PL1, PL0 represent permission tables from root to leaf. RISC-V Sv39 (3-level page table) here. All memory reference numbers presented in this paper adhere to the RISC-V ISA specification [105] and do not take into account PWC or other micro-architecture optimizations that could potentially bypass page table pages.***

can have its own OS kernel and applications. For example, Domain-1 and Domain-2 in the figure have their own data pages labeled with *D1* and *D2*. A domain's private memory cannot be accessed by other domains, guaranteed by physical memory isolation hardware. The *Host* is the default domain when the system is booted. The secure monitor usually provides interfaces for the *Host* to manage the life-cycle of other domains, such as creation and destruction. The *Host* cannot access others' private memory either.

This model can represent confidential systems on different architectures. For example, ARM CCA [9] utilizes a monitor running on EL3 to manipulate the hardware GPT for memory isolation and allows the *Host* to create new realms (similar to *domains*) with private memory. Intel TDX [13] introduces a TDX module that operates in Secure-Arbitration Mode to manage TD-ownership tags and Physical-Address-Metadata Table (PAMT). It offers domain abstraction known as Trust Domains (TD). RISC-V Keystone [70] and Penglai [53] utilize a secure monitor running on M-mode to manage enclaves and rely on RISC-V PMP and other hardware features [22] for isolation.

## 2.2 Physical Memory Isolation and Challenges

Physical memory isolation is an important technique to achieve isolation among domains. We classify the approaches used by existing confidential systems for physical memory isolation into two categories.

***Segment-based isolation.*** Segment-based isolation is commonly realized through segment registers, which record the *range* and *access permissions* for a fixed number of memory regions. For example, RISC-V PMP [105] is a segment-based design. We show how RISC-V PMP changes the behavior of a memory access in Figure 2. As shown in Figure 2-a, a hardware without confidential computing support (or not enabled) needs to perform four memory references for a memory access (3-level page table, TLB miss), including three references for page table pages and one reference for the data page. As shown in Figure 2-b, with RISC-V PMP, the hardware will further check the four memory references using PMP registers. For each memory reference, the hardware will find a matching PMP entry, check the permission, and decide whether the access is allowed or denied. Segment-based isolation is efficient because all the permissions are saved in registers, and the checking is performed in the CPU.

However, a drawback of segment-based isolation is its scalability and coarse-grained granularity. For example, RISC-V TEEs based on PMP [70] can only support a limited number of domains. Although they can utilize either enclave runtime or hardware extensions [19] to mitigate the limitation, the issue remains and becomes more serious for cloud scenarios.

***Permission table-based isolation.*** To achieve fine-grained isolation (e.g., 4KB page), hardware vendors introduce yet another table [8, 9, 13], the *permission table*, to maintain the permissions of each physical page. It has a similar structure as a page table and will map a physical address to a permission. The permission table can support (almost) unlimited memory regions using page granularity. However, a drawback is that the table introduces more memory references. As shown in Figure 2-c, a 2-level permission table can incur eight more memory references for RISC-V Sv39 (out of 12 total), forming a two-dimensional (2D) page walk even for a non-virtualized environment. Even so, due to the importance of scalable and fine-grained isolation for emerging applications, hardware vendors are gradually evolving toward table-based isolation, such as Intel TDX [2] and ARM CCA [7, 73]. Overcoming the performance issue while retaining fine-grained granularity is a significant challenge.

***Implications of permission table-based isolation.*** To better understand the implications of the extra dimensional page walks caused by the permission table, we conducted a set of experiments based on the RISC-V with FPGA (see §8 for details). We highlight results that inspired our design.

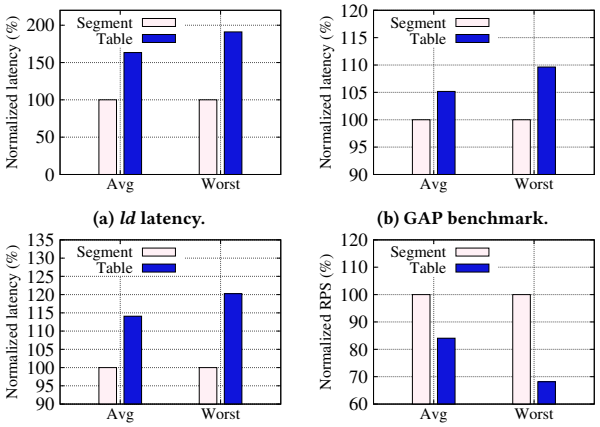



**Figure 3: Preview of experimental results (BOOM). *(a) the end-to-end latencies of a single* ld *instruction. (b) the execution time of the GAP benchmark suite. (c) the execution time of image processing serverless application. (d) the end-to-end RPS (request-per-second) of Redis benchmark. All data is normalized using the Segment's value. "Worst" means the results of a case that the Table has the worst performance.***

First, to understand how the permission table affects memory access performance, we measured the latency of a single *ld* (memory load instruction in RISC-V) under different settings. The result (Figure 3-a) shows that table-based isolation may incur 63.4% higher latency on average and up to 91.1% for a two-level permission table. This confirms that *the permission table can incur significant performance slowdown for single memory access* (**Implication-1**).

Second, as modern microarchitecture adopts many optimizations to reduce TLB miss rates and mitigate the translation costs for applications with good locality, we wanted to know whether we could rely on classic optimizations like TLB to mitigate the slowdown for applications with good locality. As a result, we carefully optimized our implementation of the permission table and extended TLB entries to cache permissions fetched from the permission table (called *TLB inlining*) — the permission table is only required for TLB miss cases. We measured the latency of the GAP benchmark [35], as shown in Figure 3-b, and found that *classic optimizations like TLB inlining can effectively mitigate the costs for computation-intensive workloads* (**Implication-2**), with 5.2% on average and up to 9.6% higher latency compared with segment.

Third, we further measured the performance of real-world cloud applications such as serverless functions and Redis to understand how the optimized table performs with real-world fine-grained and memory-intensive applications. As shown in Figure 3-c and d, the permission table incurs non-trivial costs for the two applications, with up to 20.3% and 31.8% performance slowdown, respectively. As researchers, *we still need to investigate a better way for an efficient and fine-grained physical memory isolation design* (**Implication-3**).

## 3 APPROACH OVERVIEW

***Observation.*** We observe that most memory references caused by the permission table are used to check the validity of page table pages. This observation motivates the key idea of our approach: *use segment registers to manage permission for page table pages (for efficiency), while using permission table for other pages (for scalability and fine-grained granularity)*, as shown in Figure 4.

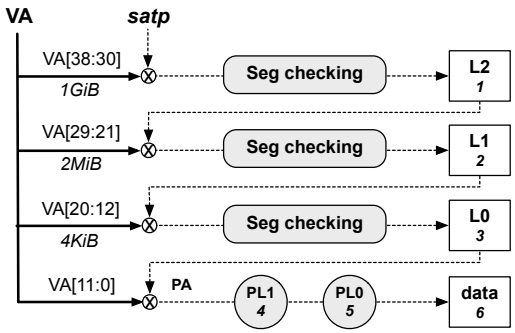



**Figure 4: Hybrid physical memory protection. *By placing the page table pages in a contiguous region (isolated using segment), the total memory references can be reduced from 12 to 6 for RISC-V Sv39.***

***Design overview.*** To support the hybrid isolation, we need to answer two questions: (1) how to design the hardware mechanism that can support both segment and table-based isolation, and (2) how to design the system software to best utilize the hardware mechanism to achieve efficient and flexible isolation. This paper proposes HPMP, a hardware-software co-design with two technical contributions that address the above questions.

First, we propose PMP Table hardware extension that harmonizes segment and table-based isolation, *without* duplicating hardware structures. The key insight is that we can think of a segment as the huge page of a permission table — when a table has the same permission for all its pages, we can directly save the permission in its registers to avoid extra-dimensional page walks. Specifically, HPMP includes multiple entries. Each entry manages permissions of one contiguous physical memory region and can directly record the permission in the entry's *register* (called segment mode) or utilize *permission table* for fine-grained permissions (called table mode). When an entry is enabled, only one mode will be used. The secure monitor of a confidential system can easily switch between the two modes to balance performance and flexibility.

Second, we extend Penglai Enclave [53] and the OS for hybrid isolation (called Penglai-HPMP). Although the OS manages pages of an isolated domain, we cannot trust the OS for security concerns. Therefore, we decouple the policies and mechanisms of physical memory isolation and introduce the general memory segment (GMS) abstraction. One GMS represents a continuous memory region with the same permission and a label. The OS can add labels to GMS (e.g., "fast") but cannot modify the region range and permission, which are enforced by Penglai-HPMP. The Penglai-HPMP will use the labels from the OS as a hint and try to isolate "fast" GMSs using segment mode while other GMSs using table mode.

As a result, HPMP can generally reduce the memory references from 12 to 6 for RISC-V Sv39 with the same granularity as the permission table. Next, we will explain how HPMP hardware (§4) and software (§5) are designed.

***End-to-end example.*** Let's consider an end-to-end example with a secure domain (e.g., an enclave) on RV64 with a three-level page table (Sv39). Now, suppose the domain executes a memory load instruction (e.g., *ld* in RISC-V) using a virtual address. In the traditional table-based physical memory isolation approach, the hardware first checks if the corresponding physical address is cached in the TLB. If it is, the hardware only needs to check the validity of the physical address of the accessed data by traversing the permission table. Notably, the permission table itself can be cached and does not require memory fetch from DRAM in all cases.

However, if there is a TLB miss, the hardware (specifically, the Page Table Walker or PTW) must translate the virtual address to a physical address using the page table. This process incurs additional memory references, potentially three in the worst case, for three page table pages. Each page table page requires traversing the permission tables to check whether it can be accessed by the domain. Notably, PT page accesses can also be optimized by techniques like caching (e.g., PWC).

HPMP follows a similar procedure. However, unlike existing table-based isolation techniques, HPMP utilizes segments to protect permissions of PT pages. This means that, during a TLB miss, HPMP still only needs to check data pages using the (relatively) slow permission table, while it can use segments (i.e., PMP) to check permissions for PT pages. This approach effectively mitigates the overhead caused by table-based isolation while achieving scalability and fine-grained granularity.

# 4 HPMP HARDWARE DESIGN

## 4.1 RISC-V PMP background

RISC-V PMP [105] is a segment-based isolation design that supports up to 16 entries. Each entry includes two registers: *addr* and *config* register, allowing permissions (*read*, *write*, *execute*) to be specified for 16 regions. The range of a region can be represented by either (1) two address registers (current *addr* as the upper bound and prior entry's *addr* as the lower bound) or (2) one address register with an embedded size. Each *config* contains an address field to indicate how the region range is represented and a permission field (three bits for *read*, *write*, and *execute*).

## 4.2 Hybrid Physical Memory Protection

The hardware extension of HPMP is based on RISC-V PMP. Figure 5 presents the hardware design of HPMP. Same to PMP, HPMP includes multiple entries, each entry includes two registers: *addr* and *config* register. One entry can behave as a segment to represent a physical memory region and its permission, e.g., HPMP-0 in Figure 5; or it can work with the following entry as a permission table to manage permissions, e.g., HPMP-1 and HPMP-2 in the figure.

***Switching between segment and table mode.*** Each HPMP *config* register includes an important bit, the T bit (Table mode bit). When this bit is cleared, the entry uses the permission recorded in the *config* register as the effective permission for the entire physical memory region, i.e., segment mode. Otherwise, the entry uses a PMP table (§4.3) to record permissions for each 4KB (or other granularities like 64KB, etc.) physical page. Hardware will use the permission fetched from the table to validate physical addresses

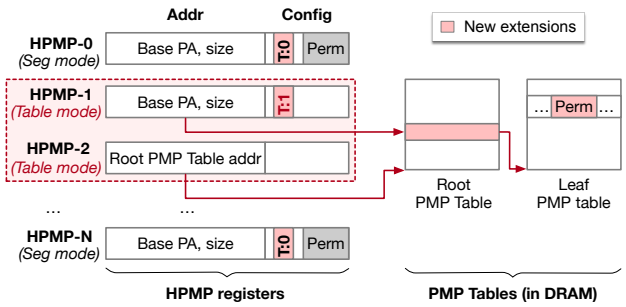



**Figure 5: HPMP hardware design.** ***HPMP carefully harmonizes segment-based and table-based physical memory isolation to achieve efficient and fine-grained protection. Our prototype supports 16 entries, i.e., the "N" is 15.***

covered by the entry. As a result, HPMP can support multiple entries, each with different isolation modes, and can easily switch any entry between segment and table modes by changing T bit.

***Permission checking and ordering.*** HPMP's permission check applies to all memory accesses from the operating system (RISC-V Supervisor mode, denoted by *S*) and applications (RISC-V User mode, denoted by *U*), including instruction fetches and page table walking. HPMP uses the same priority and matching logic as RISC-V PMP for different entries, i.e., all HPMP entries are statically prioritized. The lowest-numbered entry covering any byte of an access determines whether the access succeeds or fails. For example, if both HPMP-0 and HPMP-1 cover an access, HPMP-0 is used to validate whether the access is allowed. If the matching entry is in table mode, i.e., T = 1 in the *config* register, the hardware will utilize the PMP Table (§4.3) to retrieve permissions. Otherwise, the hardware will utilize the permission in the *config* register to validate the access. When an access is not covered by any HPMP entry, the S-mode and U-mode software have no permissions by default. HPMP entries can only be managed by RISC-V M-mode software.

## 4.3 PMP Table

A key contribution of HPMP's hardware design is the PMP Table, which can be embedded in the existing segment-based isolation mechanism, RISC-V PMP. PMP Table is carefully designed to not introduce any new instructions or registers.

***Configuration register.*** HPMP introduces the T field in the previously reserved bit-5 of the *config* register, indicating whether this HPMP entry is in table mode. Figure 6-a shows the layout of the HPMP configuration registers. When an entry is in table mode, the physical memory region managed by the entry is still represented by the *addr* register and the address field (A) in the configuration, which is the same as the existing PMP address matching rules. However, the permission field in the *config* register will be ignored. The hardware will fetch the permission from PMP Table.

***PMP Table.*** A PMP Table is a multi-level radix tree permission table, similar to the page table. When an HPMP entry is in table mode, the next entry's address register records the base address of PMP Table. If the *i*th HPMP entry is in table mode, then the

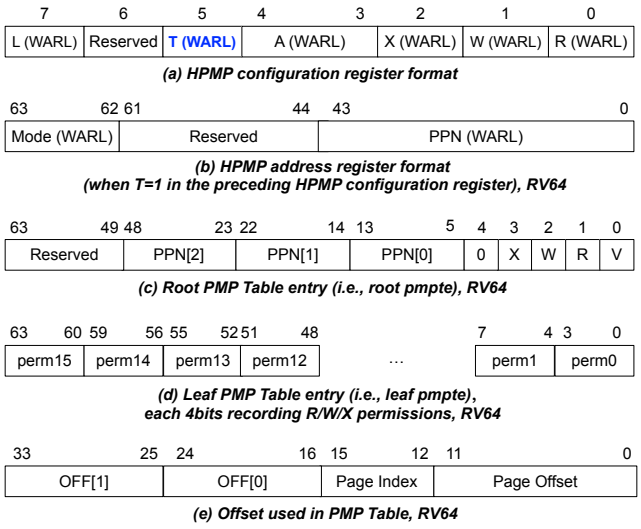



**Figure 6: Hardware structures of HPMP (PMP Table extension). *"WARL" stands for Write-Any-Values and Reads-Legal-Values in RISC-V ISA specification [105].***

*i+1*th entry records the base address of the *i*th entry's HPMP table. Similar to the RISC-V page table register, the *addr* register records the PFN (Physical Frame Number) of the base address of PMP Table (Figure 6-b). The table can be configured through the Mode field. In our prototype, the HPMP table is a 2-level table when Mode's value is 0, and all other values are reserved for future extensions that can support different levels by using reserved modes. The last HPMP entry cannot be in table mode because it has no successor entry to record the table's base address.

The first-level table is called the root PMP Table, and the second-level table is called the leaf PMP Table. Entries of the root table are called root *pmpte*, and entries of the leaf table are called leaf *pmpte*. The format of the root *pmpte* is shown in Figure 6-c. The V field indicates whether the root *pmpte* is valid. If it is 0, all other bits in *pmpte* are ignored, and memory access fails. The R, W, and X fields represent permissions. When they are all zero, the root *pmpte* is a pointer to the next-level table. Otherwise, they constitute the final permission for a memory access. This is similar to huge pages in the page table. Notably, in both RV64 and RV32, one root *pmpte* manages 32MB of physical memory.

The format of the leaf *pmpte* is shown in Figure 6-d. Each leaf *pmpte* has 64 bits and records the access rights for 16 physical pages (4KB) for RV64. The permission of each physical page is 4 bits, of which 3 bits correspond to the R, W, and X fields, and one is reserved for future use.

***Permission indexing using PMP Table.*** To index the permission, the hardware uses the offset of the target address to the start address of the memory region managed by the entry. The offset is split into four parts, containing OFF[1], OFF[0], PageIndex, and PageOffset), as shown in Figure 6-e. The OFF[1] is used to index the root PMP Table, and the OFF[0] is used to index the leaf PMP Table. The PageIndex is used to index the permission in the leaf *pmpte*.

***Why does PMP Table choose a 2-level permission table?*** It enables better performance with fewer costs for the extra dimension of page walks. Although a two-level PMP Table can only manage permissions for a 16GB region (for RV64), PMP Table can protect larger regions by using multiple tables. For example, 16 HPMP entries can support 8 PMP Table and therefore support 128GB of memory. Moreover, future RISC-V processors will support 64 PMP entries with the ePMP extension [17]. With 64 entries, a CPU can use 2-level tables to manage 512GB of memory, which is sufficient in most cases. For scenarios that require a larger region, it is easy to extend PMP Table to support 3-level or 4-level tables by using the reserved values in the Mode field.

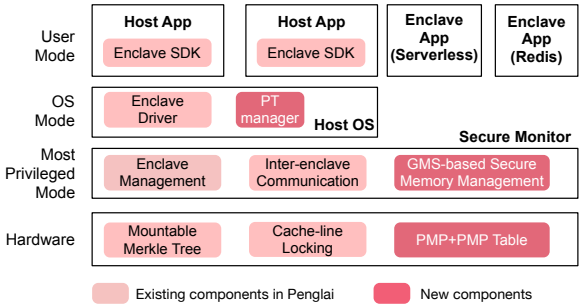



**Figure 7: Penglai-HPMP architecture.**

## 5 PENGLAI-HPMP: SOFTWARE SUPPORT

We propose Penglai-HPMP, a RISC-V confidential system based on Penglai Enclave [53] that supports HPMP-based isolation, as shown in Figure 7.

***Penglai background.*** Penglai is an open-sourced TEE system. It introduces a small software component called the *secure monitor* that operates in the most privileged mode (e.g., M-mode in RISC-V) and utilizes new hardware extensions to establish enclave abstractions. Each enclave is isolated from the untrusted host and other enclaves. The secure monitor manages enclaves, providing APIs for enclave deployment. Resource protection is separated from management to minimize the size of the secure monitor. During system boot, the secure monitor is loaded and verified by the boot ROM (secure boot), taking control of the system and safeguarding itself with hardware-supported memory isolation (e.g., RISC-V PMP). It employs encryption and merkle tree to defend against physical memory attacks.

***General memory segment (GMS).*** Penglai-HPMP uses GMS as the unified abstraction for memory isolation. Penglai-HPMP abstracts the HPMP entries into: *N fast* GMSs and *unlimited slow* GMSs. The *N* is decided by the secure monitor of Penglai-HPMP. For example, in an RV64 system with 32GB memory and 16 HPMP entries, the secure monitor can reserve 1 entry for its private memory and 4 entries for two PMP Table (for the "unlimited" but slow GMSs), and 11 fast GMSs. Penglai-HPMP should consider the total memory to decide the ratio between fast and slow segments, e.g., 2 tables can cover a 32GB region, which is sufficient.

GMS provides clean and effective abstractions to the OS and applications. The OS thinks of each contiguous physical memory region (with the same permission) as one GMS and adds a "fast" or "slow" label. Penglai-HPMP will take the hints (i.e., "fast" and

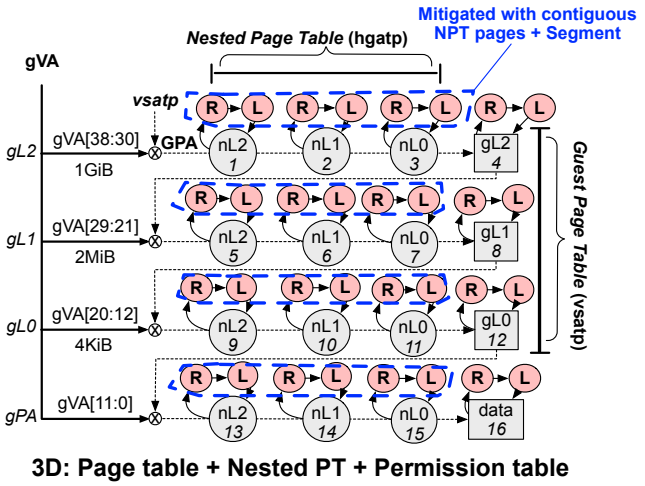



**Figure 8: Memory access (in a guest) with permission table. *"R" and "L" are short for root/leaf PMP Table.***

"slow") from the OS and is responsible for configuring HPMP entries accordingly. As HPMP will cache the permission in TLB, it requires Penglai-HPMP to flush TLB whenever HPMP entries are updated. This is supported by existing TEEs like Keystone [14] and Penglai [16], which will not introduce extra costs for TLB synchronization.

***Cache-based management.*** Penglai-HPMP uses a cache-like structure to manage HPMP. Specifically, the segment mode entries will always have a higher priority than tables by using lower-numbered entries as HPMP entries are statically prioritized. Then, Penglai-HPMP will only put fast GMS to segments and include *all* GMSs in tables, regardless of whether a GMS is "fast" or "slow". This is similar to abstracting the segment entries as a cache of tables. As a result, when the OS updates the labels, Penglai-HPMP only needs to modify the HPMP registers, which is faster than modifying tables.

***Operating system support.*** The OS kernel in our prototype is enhanced to manage all PT pages in a single GMS. The OS kernel will assign a "fast" label to this GMS, and then Penglai-HPMP will try to use a segment entry for the GMS. As a result, this can mitigate all the 6 memory references in Figure 2-c for PT page checking. The memory references are reduced from 12 to 6, with only 2 extra references necessary for checking the addresses of the data, which is a significant improvement with minor OS modifications and no application modifications. HPMP is generic for all applications and operating systems and can significantly improve system performance. The OS modification is acceptable. Although HPMP is the first to utilize contiguous page table pages to optimize the extra-dimensional permission table walks, prior systems [53, 77, 79, 85] have already explored a similar way to organize PTs into a contiguous region for other benefits. For example, Penglai [53] requires a contiguous region for PT pages to trap the modifications to page tables, and ASAP [77] requires contiguous PT pages to enable prefetching.

## 6 VIRTUALIZED ENVIRONMENT

Things become worse in virtualization as one memory access needs to go through a three-dimensional (3D) page walk, i.e., guest page table (GPT), nested page table (NPT), and permission table, as shown in Figure 8. A 2-level permission table will add 32 more memory references for a virtualized environment with RISC-V Sv39 GPT and Sv39x4 NPT.

In a virtualized environment, a similar observation works for NPT pages — most memory references (24 out of 32) from extra-dimensional page walks are for NPT checking. Following the design of HPMP, the hypervisor (e.g., KVM in Linux) can intentionally allocate NPT pages in a single or a few GMSs and label these GMSs as "fast". Therefore, Penglai-HPMP can utilize segment entries to manage these NPT pages. Although HPMP can mitigate 24 memory references, there are still 8 extra memory references caused by 3D page walks. We observe that 6 of the remaining 8 memory references are used to check the GPT pages. Taking one step forward, if the guest kernel adopts the extension of HPMP to manage GPT pages in a contiguous region and notify the hypervisor of the "fast" GMSs in the guest, the hypervisor can further use a contiguous region for the GPT pages. This means we can further reduce 6 memory references, resulting in only 2 extra memory references (called HPMP-GPT), which is a significant improvement.

**Table 1: Simulation configurations. *The LLC and DRAM are simulated through FireSim's DRAM model [38, 65].***

| | Parameter | Value/Description |
|---|---|---|
| **Rocket** | Processor | In-order RISC-V CPU @ 1GHz |
| | L1 Cache | 16KiB I/D cache |
| | L2 Cache | 512KiB (8-way set-associative) |
| | LLC | 4MB |
| | L1 I/D TLB | 32 entries each, fully-associative |
| | L2 TLB | 1024 entries, direct-mapped |
| | PTECache | 8 entries |
| **BOOM** | Processor | OoO RISC-V CPU @ 3.2GHz |
| | Front-end | 8-wide fetch, 32-entry fetch buffer, 4-wide decode, 28KB TAGE branch predictor, 40-entry fetch target queue, max 20 outstanding branches |
| | Execute | 128-entry ROB, 128 int/fp physical registers, 24-entry dual-issue MEM queue, 40-entry 4-issue INT queue, 32-entry dual-issue FP queue |
| | LSU | 32-entry load/store queue |
| | L1 Cache | 32KiB 8-way I-cache, 32KiB 8-way D-cache w/ 8MSHRs |
| | L2 Cache | 512KiB (8-way) w/ 12MSHRs |
| | LLC | 4MB (8-way) w/ 8MSHRs |
| | L1 I/D TLB | 32 entries each, fully-associative |
| | L2 TLB | 1024 entries, direct-mapped |
| | PTECache | 8 entries |
| **Memory** | 16GB DDR3 FR-FCFS quad-rank, 25.6 GB/s maximum bandwidth, 14-14-14 (CAS-RCD-RP) latencies @ 1GHz, 8 queue depth, 32 max reads/writes | |
| **OS** | Buildroot, Linux 5.10, OpenSBI 1.0 | |

## 7 IMPLEMENTATION

***Hardware.*** The HPMP prototype is implemented based on two widely used RISC-V implementations, RocketCore [28] (a 5-stage in-order scalar processor) and BOOM [114] (a 4-way superscalar out-of-order core). As shown in Figure 9, we extended the existing PMPchecker with a new module called PMP Table Checker, which includes a PMPTW and PMPTW-Cache. The PMPTW finds a matching HPMP entry for an address and determines its validity using inline permission if the entry is not in Table mode. Otherwise, it walks through a 2-level permission table to retrieve the permission.

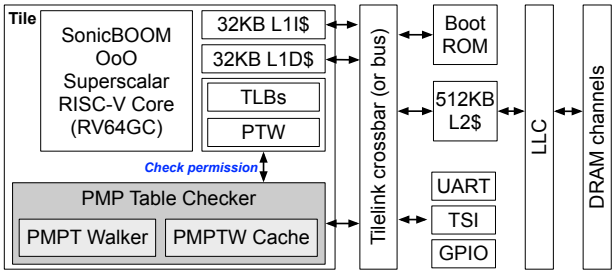



**Figure 9: HPMP implementation based on BOOM.**

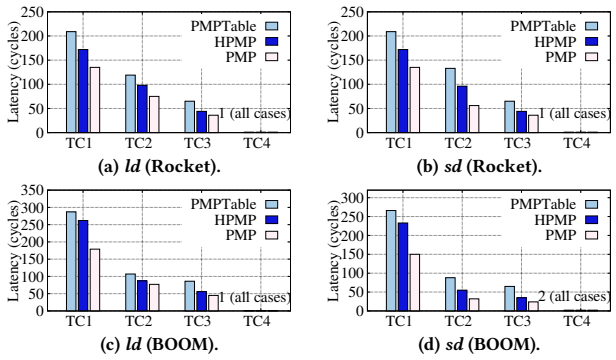



**Figure 10: Memory access latency. *The latency is evaluated using four test cases (Table 2). PMPTW-Cache disabled.***

**Table 2: Test case configuration for memory access latency. *PWC indicates whether the related PTE is cached.***

| Cases | Cache | PWC (L2) | PWC (L1) | PWC (L0) | TLB |
|---|---|---|---|---|---|
| TC1 | Cold | Miss | Miss | Miss | Miss |
| TC2 | Warm | Miss | Miss | Miss | Miss |
| TC3 | Warm | Hit | Hit | Miss | Miss |
| TC4 | Warm | Hit | Hit | Hit | Hit |

The PMPTW-Cache is a dedicated cache similar to PWC (page walk caches), which can improve the PMPTW performance by caching hot entries. We also modified the TLB and PTW module to utilize the new PMP Table Checker for permission checking and added optimization in TLB to inline the physical memory isolation to avoid permission checking in the case of TLB hit (for both baseline and our systems). We did not need to modify any lines in the core pipeline.

***Software.*** We implement Penglai-HPMP based on the Penglai Enclave [15, 53] (PMP version, v0.2 release), which is the state-of-the-art RISC-V confidential system, and extend it with HPMP hardware management and support for GMS abstractions. We do not utilize the guarded page table and other new hardware features proposed in Penglai [52]. Additionally, the OS support for HPMP adds about 700 lines of C code to the Linux kernel (v5.10).

***Methodology.*** We use Firesim [65], an FPGA-accelerated and cycle-accurate simulator, on AWS EC2 F1 for evaluation. The FPGA simulated a 1GHz SoC for RocketCore and a 3.2GHz SoC for BOOM. The detailed configuration is shown in Table 1. Additionally, we use Verilator [101], a cycle-accurate simulator (Verilog simulator), for microbenchmark and performance analysis. We compare HPMP with PMP (as the segment-based isolation) and PMP Table (as the table-based isolation) to illustrate the benefits of performance and flexibility. For application benchmarks, we compare Penglai-HPMP (Penglai extended with HPMP) with two systems, Penglai-PMPT (modified to utilize PMP Table for better scalability) and Penglai (unmodified version based on PMP). To understand the performance impacts, we port LMBench, RV8, GAP benchmark, serverless applications (including FunctionBench [66] and image processing [23, 113]), and Redis to our confidential systems for evaluation. We disable PMPTW-Cache by default, and will analyze the benefits of caching in §8.9. The details will be explained in §8.

# 8 EVALUATION

## 8.1 Memory Access Latency

***Methodology.*** We analyze the performance impact of HPMP on memory access instructions. Specifically, we evaluate the latency of a single memory load and store instruction (i.e., RISC-V *ld* and *sd*) under four test cases, as shown in Table 2. TC1 represents the access latencies when all states are cold, usually the worst-case latency. In TC2, data and page table pages are cached in the system cache, but no state is cached in the TLB and PWC (Page table Walker Cache) [36, 85]. TC2 represents the access latency when the TLB is flushed (e.g., the OS issues *sfence.vma*). TC3 represents the access latency when most states have been warmed up. The corresponding real case is an application (with good locality) that jumps from one page to an adjacent page. TC4 represents the best-case latency where all states are pre-warmed, i.e., the TLB hits, and the data cache also hits.

***Results.*** The results are shown in Figure 10. We find that PMP Table encounters more latency than PMP in all cases, e.g., in BOOM, 38.9%–91.1% more cycles for *ld* instruction and 77.3%–175.0% for *sd* instruction. HPMP can mitigate 23.1%–73.1% costs of extra dimentional page walks on BOOM and 47.7%–72.4% costs on Rocket. Our experiments also confirm that the optimizations to inline permissions into TLB entries can effectively mitigate extra-dimensional page walk costs (the same latencies for load and store during TLB hit).

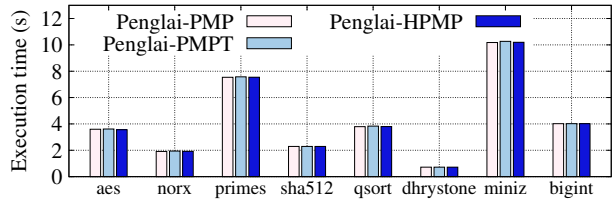
## 8.2 OS Operations

***Methodology.*** When confidential computing is enabled, normal applications and the Host OS are impacted because of the memory access checking. To understand the impact, we measure the performance of core OS operations with physical memory isolation using LMBench [81]. We compare Penglai-HPMP with Penglai-PMP and Penglai-PMPT. Penglai-PMP represents the results of both secure and non-secure baseline.

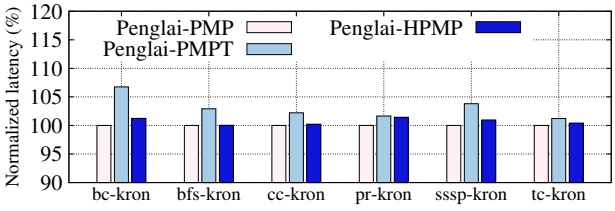
***Results.*** Table 3 shows the mean latencies of system calls on BOOM. Overall, physical memory isolation incurs non-trivial costs to OS operations — PMP Table has up to 60.33% higher latency than PMP (39.03% on average). The *null* operation incurs the least cost, while

**Table 3: Costs of OS operations (BOOM).** ***We use LMBench to evaluate each syscall cost with different isolation methods. PMP represents the results of both secure and non-secure baseline. PMPT is short for PMP Table.***

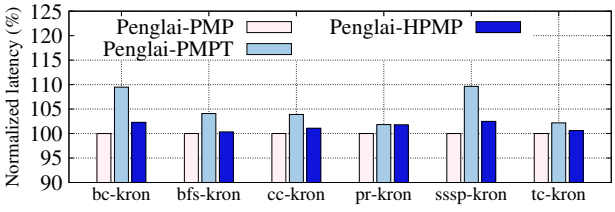
| Syscall (ms) | PMP | PMPT | HPMP | PMPT/HPMP |
|---|---|---|---|---|
| null | 0.12 | 0.12 | 0.12 | 100.00% |
| read | 0.27 | 0.41 | 0.41 | 141.38% |
| write | 0.16 | 0.21 | 0.17 | 123.53% |
| stat | 1.17 | 1.87 | 1.31 | 142.69% |
| fstat | 0.25 | 0.33 | 0.27 | 122.22% |
| open/close | 2.62 | 4.12 | 2.94 | 140.14% |
| pipe | 6.05 | 8.20 | 6.40 | 128.13% |
| fork+exit | 389.85 | 558.19 | 432.83 | 128.96% |
| fork+exec | 407.93 | 570.69 | 442.92 | 128.85% |
| **Avg** | | | | 128.43% |




**(a) RV8 benchmark on RocketCore.**




**(b) GAP benchmark on RocketCore.**




**(c) GAP benchmark on BOOM.**

**Figure 11: Benchmarks.** ***Evaluated using RV8 and GAP (Graph Algorithm Performance) benchmarks.***

*stat* and *open/close* incur the most cost. We obtain similar results on RocketCore: PMP Table has 11.34%–35.38% higher latency than PMP and 26.46% higher latency on average. HPMP can effectively mitigate the costs and achieve very close performance to PMP. Compared with HPMP, PMP Table incurs 28.43% higher latency (on average). Although it is a great improvement, we believe there is still room for further optimizations, and future researchers can explore other strategies for HPMP to achieve better latency.

## 8.3 Benchmarks Suites

***Methodology.*** We use RV8 and GAP (Graph Algorithm Performance) [35] benchmark suites to evaluate the performance of computation-intensive workloads with physical memory isolation. RV8 covers typical scenarios such as encryption (AES), computation-intensive workload (dhrystone), and hash-checking (sha512). GAP covers a collection of high-performant implementations for graph processing. We use the firesim ported version of GAP with Kron input graph, and the graph size is configured as *graph500* ($2^{20}$ vertices). These experiments use three methods: Penglai-PMP, Penglai-PMPT, and Penglai-HPMP.

***Results.*** The results are shown in Figure 11. As we can see, since most of the applications in the RV8 and GAP are computation-intensive and have good locality, even Penglai-PMPT can achieve great performance with 2-level permission table design and TLB inlining optimization. Specifically, compared with PMP, Penglai-PMPT introduces 1.2%–6.7% higher latencies on RocketCore and 1.8%–9.6% on BOOM for GAP, and 0.0%–1.7% on RocketCore for RV8. For RV8, *Norx* has the largest performance overhead (1.7%) and *bigint* has the smallest (0.0%). Penglai-HPMP can reduce the costs to 0.0%–0.5%. For GAP, *bc-kron* has the largest performance overhead on both RocketCore and BOOM, while *pr-kron* and *tc-kron* have the smallest. Penglai-HPMP can reduce the costs to 0.02%–1.4% on RocketCore and 0.6%–2.4% on BOOM. The results confirm that HPMP's elaborate design performs well for table-based isolation. Moreover, we believe prior efforts [24, 33] on increasing TLB hit rates will become more important for future systems with permission tables.

## 8.4 Case Study: Serverless Computing

Serverless computing is a new trending paradigm in cloud computing and has already been supported by many platforms [3–6]. Serverless applications are fine-grained and short-lived. This section shows how physical memory isolation affects the performance of short-lived functions.

***FunctionBench.*** We evaluate HPMP with applications from FunctionBench [66], a widely-used serverless benchmark. To provide a clear comparison, we normalize the results (the execution latency) and label the concrete numbers in the figure. As shown in Figure 12-a and b, when compared with Penglai-PMP, Penglai-PMPT exhibits a latency increase of 1.0% to 14.3% (with an average of 5.1%) for RocketCore. For BOOM, it requires a latency increase of 5.5% to 20.3% (with an average of 14.1%). Penglai-HPMP significantly reduces the costs to 0.0%–4.3% (2.0% on average) on RocketCore and 0.0%–6.4% (3.5% on average) on BOOM. We also evaluate the non-secure baseline by running functions on the Host domain using BOOM, labeled as Host-PMP in the figure. The secure and non-secure baselines exhibit similar results as they both utilize PMP for memory protection.

***Chained application.*** We port a multi-function serverless application, image processing [23, 113], from AWS Serverless Repository [10] to evaluate the end-to-end latency of a function chain. The application consists of four functions and will accept an image file from a client, process the image, and return a new image. The size of processed image is changed from 32x32 to 256x256. The

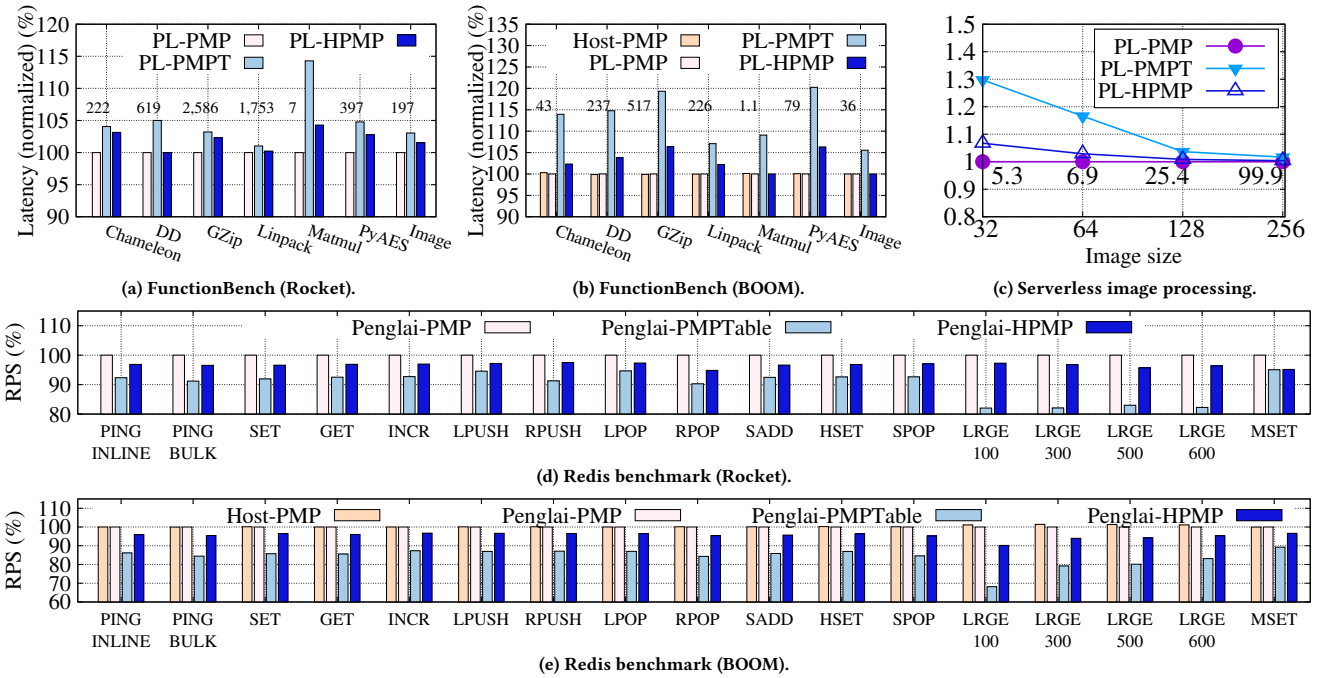



**Figure 12: Applications.** ***Figure-a/b present the end-to-end latencies of FunctionBench. Figure-c shows the end-to-end latencies of image processing serverless applications. Figure-d/e present the requests-per-second of Redis's benchmark. Host-PMP represents the non-secure baseline without TEE deployed, however, PMP is still implemented by Rocket/BOOM.***

results are shown in Figure 12-c. Compared to PMP, Penglai-PMPT introduces performance overheads of 1.6%–29.7%. As the size of the processed image increases, the computation costs grow faster than the memory access costs, so the overall performance gap gradually decreases (29.7% to 1.6%). Penglai-HPMP introduces only 0.3%–6.7% performance overhead.

## 8.5 Case Study: In-Memory Data Store

Redis [18] is a widely-used in-memory data store for many scenarios, e.g., data caching, database, message bus, etc. In this section, we use Redis as an example to demonstrate how physical memory isolation impacts the performance of long-running memory-intensive applications like Redis.

***Methodology.*** We use Redis's benchmark tool, *redis-benchmark*, for evaluation. The tool simultaneously simulates an arbitrary number of clients connecting to the Redis server. These clients perform actions and measure the latency for each request. The output results provide the average number of requests per second. We maintain the default settings for the number of clients (i.e., 50) and data sizes in bytes (i.e., 3).

***Results.*** The results are shown in Figure 12-d and e. When compared with Penglai-PMP, Penglai-PMPT exhibits lower throughput, ranging from 5.9% (*MSET*) to 18.0% (*LRANGE_100*) with an average of 10.5% on RocketCore. On BOOM, the performance gap is more significant, ranging from 10.8% (*MSET*) to 31.8% (*LRANGE_100*) with an average of 16.0%. These results confirm that the permission table incurs non-trivial costs for memory-intensive applications like Redis. However, with Penglai-HPMP, we observe a significant reduction in costs, with only 3.3% (on average) on RocketCore and 4.5% (on average) on BOOM. As a result, Penglai-HPMP effectively mitigates the costs associated with the extra-dimensional page walk, even for memory-intensive applications.

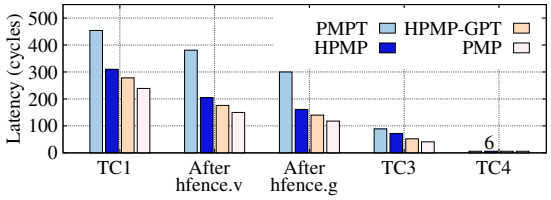
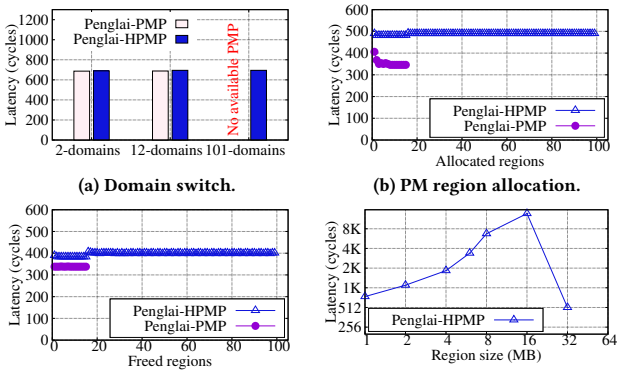



**Figure 13: Memory access lantecy for virtualized environment.** ***"hfence.v" and "hfence.g" represent an access after hfence.vvma and hfence.gvma.***

## 8.6 Virtualization

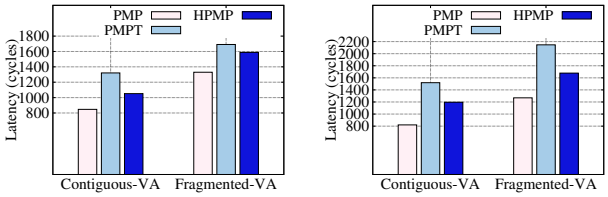
We follow the methodology in §8.1 to evaluate the memory access latency in a virtualized environment on RocketCore.

***Methodology.*** The virtualized environment has Sv39x4 nested PT and Sv39 guest PT. We use RISC-V's *hlv.d* instruction to access a guest VA in the host environment to avoid interferences from guest system software. We compare four methods, PMP, PMP Table, HPMP, and HPMP-GPT. HPMP means NPT pages are isolated using segment mode, while data pages are managed by table mode. HPMP-GPT further utilizes segment mode to isolate the GPT pages. We consider five cases: three of them are similar to TC1 (cold),




(a) Domain switch. (b) PM region allocation. (c) PM region release. (d) Allocation with different sizes.

**Figure 14: TEE performance.**




(a) Contiguous physical pages. (b) Fragmented physical pages.

**Figure 15: Memory fragmentation.**

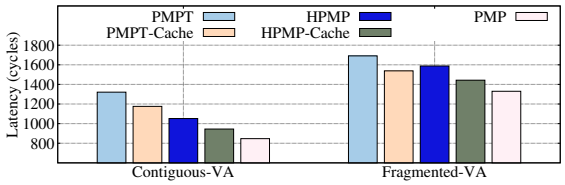



**Figure 16: Caching for permission table.**

TC3 (access neighbor page), and TC4 (TLB hit) in Table 2. In "After hfence.v" and "After hfence.g", we access an address and then perform *hfence.vvma* or *hfence.gvma*, and then access the address again and measure the second access's latency. These five cases show the access latencies with different system states.

***Results.*** As shown in Figure 13, compared with a non-virtualized environment, PMP Table incurs higher costs for all cases, 89.9%–155.0% higher latencies compared with PMP. HPMP achieve lower latency than PMP Table, reducing the costs to 29.7%–75.6%. If the guest could notify the hypervisor to allocate GPT pages in a contiguous region, i.e., HPMP-GPT, we could further reduce the costs to 16.3%–26.8%. As a result, HPMP can also significantly benefit the virtualized environment.

## 8.7 Performance of TEE Operations

We evaluate the costs of basic operations of a RISC-V TEE using Penglai-HPMP and Penglai-PMP.

***Domain switching.*** We evaluate the switching costs between two domains when multiple domains are concurrently running, as shown in Figure 14-a. The results demonstrate that Penglai-HPMP incurs negligible additional costs (less than 1%) compared to Penglai-PMP. This is because HPMP leverages the existing PMP entries for permission tables, allowing the secure monitor to employ a similar routine as Penglai-PMP to switch between domains by updating the HPMP registers. Moreover, the switching costs remain stable even with numerous concurrently running instances, as the secure monitor only needs to update HPMP registers to switch tables and segments. Notably, Penglai-HPMP can support over 100 domains with HPMP.

***Memory region allocation and release.*** We continuously allocate and release physical memory regions (64KB) for a domain and evaluate the latency, as shown in Figure 14-b and c. The results indicate that Penglai-PMP can support fewer regions for a domain due to the limited number of PMP entries, while Penglai-HPMP can support more than 100 regions using the permission table. In both allocation and release processes, Penglai-HPMP incurs slightly higher latency because it needs to modify both the permission tables and registers. However, it's important to note that dynamic memory allocation is infrequent in real-world scenarios.

***Allocation with different sizes.*** We evaluate the latency of memory region allocation with varying sizes, ranging from 1MB to 32MB, as shown in Figure 14-d. The latency increases as the region size grows because the secure monitor needs to modify the permission table entries to update the permissions accordingly. Additionally, Penglai-HPMP incorporates optimizations by utilizing a large permission table page, enabling the modification of a single entry to update the permission for a 32MB region.

## 8.8 Memory Fragmentation

Memory fragmentation is normal with on-demand paging, application colocation, and virtualization, which has been studied by prior works like PTEMagnet [79]. We analyze how HPMP behaves in four possible fragmented cases: (1) an ideal case where continuous virtual pages are mapped to continuous physical pages; (2) a common case in a non-virtualized environment where continuous virtual pages are mapped to fragmented physical pages (due to on-demand paging or other features); (3) a case in a virtualized environment where fragmented host virtual pages are mapped to continuous physical pages; and (4) a common case in a virtualized environment where fragmented host virtual pages are mapped to fragmented physical pages. The last two cases result in fragmented PTEs [79].

We conduct a microbenchmark (on Rocket) to access the same number of virtual pages using four settings and compare end-to-end latencies. Figure 15-a shows the results of cases (1) and (3), which have contiguous physical pages, while Figure 15-b shows the results of cases (2) and (4). In "Fragmented-VA", we access the next virtual page with an 8GB+4KB offset. The results show that fragmentation significantly affects the end-to-end latency, e.g., the case of "Fragmented physical pages" with "Fragmented-VA" has the worst latency. However, HPMP still outperforms PMP Table in all cases and can effectively reduce the latency even in a fragmented environment.

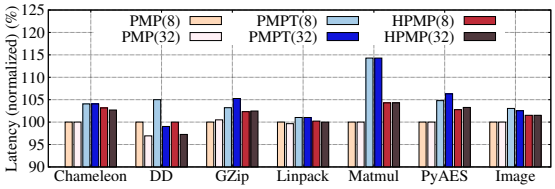



**Figure 17: FunctionBench with different PWC entries (Rocket).**

**Table 4: Hardware resource costs of top module in FPGA. *"+H" means Hypervisor extension enabled.***

| Resource | Baseline | HPMP | Cost | Base+H | HPMP+H | Cost |
|---|---|---|---|---|---|---|
| LUT | 248,292 | 250,636 | 0.94% | 249,026 | 251,965 | 1.18% |
| LUTRAM | 14,290 | 14,290 | 0.00% | 14,290 | 14,290 | 0.00% |
| FF | 258,498 | 258,909 | 0.16% | 260,073 | 262,106 | 0.78% |
| RAMB36 | 336 | 336 | 0.00% | 336 | 336 | 0.00% |
| RAMB18 | 90 | 90 | 0.00% | 90 | 90 | 0.00% |
| DSP | 18 | 18 | 0.00% | 18 | 18 | 0.00% |

## 8.9 Caching Permission Table

***Benefits of caching.*** To analyze the impact of microarchitecture optimizations, such as Page Walk Cache [36], on reducing the costs associated with the permission table, we introduce PMPTW-Cache. This dedicated cache consists of 8 entries specifically designed for the permission table (fully-associative, same replacement rule as PWC). We conduct the same test as described in §8.8 to demonstrate how caching affects latency. The results are shown in Figure 16.

As we can see, caching proves to be effective in reducing the latency of the permission table. PMP Table-Cache achieves even better performance for the "Fragmented-VA" scenario compared to HPMP. This improvement can be attributed to caching's ability to mitigate accesses to data pages that are not handled by HPMP, thereby boosting performance in fragmented cases. However, HPMP has the advantage of theoretically eliminating accesses to page table pages during permission table walking, which cannot be achieved by caching. Therefore, the combination of HPMP and cache, referred to as HPMP-Cache, can achieve the best performance in all cases.

***Cache entries.*** Next, we investigate the impact of different entries in the Page Walk Cache (PWC) on performance. Specifically, we extend the Rocket implementation to utilize a 32-entry PWC and evaluate the FunctionBench using PMP, PMP Table, and HPMP. The results, as shown in Figure 17, demonstrate that a larger PWC can reduce page-table walks in some cases, thereby mitigating overall costs. However, PMP Table still incurs costs due to permission table overhead during memory access. On the other hand, HPMP consistently outperforms the naive PMP Table design due to its ability to mitigate permission table checks. We observe that a larger PWC does not significantly improve the performance of FunctionBench. This could be attributed to the fact that the costs primarily stem from the dynamic and short-running nature of serverless functions rather than insufficient caching size. Given that HPMP eliminates all permission table checks caused by PT pages by design, it proves to be more effective and efficient for serverless computing and other dynamic and short-lived scenarios.

## 8.10 Hardware Costs

The resource utilization report in the FPGA was generated from Vivado [21], as shown in Table 4. The overall hardware costs were small (0.94%/1.18% in LUT, 0.16%/0.78% in FF, and 0 for others for the top module).

# 9 DISCUSSION AND FUTURE WORK

***Generality.*** Although this paper only implements HPMP on RISC-V, the design is general in architecture (especially ISA-level) and could be utilized to optimize the permission table walking in other ISAs. For example, ARM CCA's GPT [9] has a similar structure as HPMP's permission table, which is shown to have significant management and runtime costs by a recent work [72, 73]. CCA could be optimized by introducing multiple GPTs and allowing each GPT to be turned into a segment, which could be used to protect frequently accessed physical memory regions like page table pages.

***I/O protection using table-based physical memory isolation.*** HPMP offers the ability to isolate MMIO regions for different domains. This approach surpasses segment-based isolation by providing support for a wider range of I/O devices and their distinct regions. Additionally, HPMP (or PMP) can be employed for DMA protections, such as IOPMP [22], effectively safeguarding against malicious I/O devices.

***Efficient isolation through new abstractions.*** Introducing new interfaces for applications to specify hot and cold regions can further improve performance. In our prototype, we have developed three new ioctls within the TEE driver, enabling user applications to create, delete, and query memory range hints using virtual addresses. These hints are then conveyed to the secure monitor through labels. By leveraging these abstractions, we can further minimize the overhead of permission checking for data pages, in addition to the already optimized page table pages.

***Limitations and future work.*** This paper presents a novel design that successfully addresses the costs associated with permission tables by leveraging application behavior. However, there are still limitations and potential areas for future investigation. While our analysis and optimization efforts concentrate on the RISC-V architecture, it is essential to account for variations in results on other architectures such as Intel/ARM due to distinct microarchitectural designs and optimizations. Additionally, while providing new interfaces for applications to indicate hot and cold regions can enhance performance, it may also impose a higher burden on application developers. Further research is needed to explore these limitations and strike a balance between performance improvements and developer convenience.

# 10 RELATED WORK

***Hardware-assisted confidential systems.*** Some prior work uses a secure processor to implement confidential computing systems [1, 39, 42, 44, 46, 54, 74, 75, 89, 97, 110]. These systems utilize encryption and integrity engines to support compartments that are immune to both modification and observation. However, using encryption to isolate memory space brings non-negligible overhead, and the key management and traditional integrity scheme may restrict

the enclave size and number. As a result, emerging solutions like Intel TDX and ARM CCA prefer to utilize permission tables for isolation and can benefit from the design of HPMP.

***Physical memory isolation designs.*** Systems like CHERI [90, 106, 107], CODOMs [103], PUMP [49], and others [45, 91, 92, 99] utilize hardware capability or tagged memory for fine-grained isolation. Unlike HPMP, which is proposed for system-wide physical memory isolation, these systems are mostly used for intra-address space isolations for lightweight domains, such as libraries in an application or drivers in kernel space. We believe the hybrid design can also help systems like PUMP to manage tags more efficiently.

MMP [108, 109] proposes a highly compressed table structure to reduce space and two levels of caching to reduce runtime overheads. Compressed permission structure and caching are both valuable techniques: compressed permission is already used in two RISC-V extensions, ePMP and sPMP, and HPMP will follow the way to introduce more fine-grained and compressed permissions. However, MMP does not allow the software to change the method (segment or table) used for isolation, which is the major contribution of HPMP to enable flexible design.

***Direct segment.*** Direct segment [33, 55, 59, 68] utilizes segment registers to skip translation for a hot primary region — forming a hybrid design of paging-based and segmentation-based translation. HPMP is the first to utilize the hybrid design in physical memory isolation and has two major contributions. First, PMP Table achieves better flexibility because each hardware entry can be configured as a segment or permission table during runtime. Second, we observe that most memory references caused by the permission table are for checking PT pages and proposing the Penglai-HPMP system.

***Address translation.*** Although works on optimizing address translation [24, 30, 32, 56–58, 63, 69, 76–79, 82–87, 94–96, 100, 102, 112] do not directly address our problem, their efforts on prefetching [77, 78], new PT structures [85, 94], caching [31, 79], lowering TLB miss rates [24, 32, 56, 86, 87, 100, 102] and huge pages [58, 63, 69, 76, 82–84] can effectively reduce memory references and therefore mitigate costs of extra page walks. The implementation of PMP Table also takes these optimizations into account.

PTEMagnet [79] observes the issue of fragmented hPTEs in a virtualized environment and proposes an allocator to reserve pages to best utilize the PTE locality. HPMP leverages a similar observation on PTE locality to mitigate the costs of permission table and can be used together with PTEMagnet. Other virtual memory optimizations [25, 26, 29, 37, 64, 71, 87, 111] can effectively reduce the TLB miss rates, but can not avoid permission table walks in case of TLB misses. HPMP is still necessary even after these optimizations are deployed.

Flattend page table [85] technique reduces the depth and number of page table traversals by merging two levels of 4KB page tables into a single 2MB page table. This approach reduces the overhead associated with permission tables, as there are fewer page table pages to validate. Besides, HPMP adopts a similar concept of a flattened structure, utilizing a single 64-bit entry for 16 4KB pages.

## 11 CONCLUSION

This paper proposes HPMP, a new memory isolation design that provides great flexibility to software by harmonizing segment and table-based isolation in a single hardware structure. HPMP isolates page table pages with its segment mode while data pages are in table mode, which effectively mitigates the costs of extra-dimensional page walks and improves performance for real-world confidential applications.

## ACKNOWLEDGMENTS

We sincerely thank the anonymous MICRO'23 reviewers for their insightful suggestions. We would also like to extend our thanks to the members of the RISC-V International Community, particularly the Security HC, including Zeyu, Dingji, Jiahao, Nick, Allen, Ravi, Andy, Siqi, Robin, Mark, Sandro, and others, for their early feedback. This work was supported in part by National Key Research and Development Program of China (No. 2020AAA0108500), National Natural Science Foundation of China (No. 62302300, 61972244, U19A2060), Startup Fund for Young Faculty at SJTU (SFYF at SJTU), and CCF-Huawei Populus Grove Fund. Corresponding author: Yubin Xia (xiayubin@sjtu.edu.cn).

## REFERENCES

[1] 2019. AMD Secure Encrypted Virtualization (SEV) - AMD. https://developer.amd.com/sev/.
[2] 2020. Intle TDX. https://software.intel.com/content/www/us/en/develop/articles/intel-trust-domain-extensions.html. Referenced Apr 2022.
[3] 2021. Apache OpenWhisk is a serverless, open source cloud platform. http://openwhisk.apache.org/. Referenced 2021.
[4] 2021. AWS Lambda - Serverless Compute. https://aws.amazon.com/lambda/. Referenced Jan. 2021.
[5] 2021. Azure Functions Serverless Architecture. https://azure.microsoft.com/en-us/services/functions/. Referenced Jan. 2021.
[6] 2021. Google Cloud Function. https://cloud.google.com/functions/. Referenced Jan. 2021.
[7] 2021. HASP: Arm Confidential Compute Architecture. https://haspworkshop.org/2021/slides/HASP-2021-Session2-Arm-CCA.pdf.
[8] 2022. AMD SEV-SNP: Strengthening VM Isolation with Integrity Protection and More. https://www.amd.com/system/files/TechDocs/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and-more.pdf.
[9] 2022. Arm Confidential Compute Architecture. https://www.arm.com/architecture/security-features/arm-confidential-compute-architecture. Referenced Apr 2022.
[10] 2022. AWS Serverless Application Repository. https://serverlessrepo.aws.amazon.com/applications. Referenced Apr 2022.
[11] 2022. Hex Five Security: The quick and safe way to add security and separation to embedded systems. https://hex-five.com/.
[12] 2022. Intel 5-Level Paging and 5-Level EPT white paper. https://www.intel.com/content/www/us/en/develop/download/5-level-paging-and-5-level-ept-white-paper.html. Referenced Apr 2022.
[13] 2022. Intel Trust Domain Extensions White Paper. https://cdrdv2.intel.com/v1/dl/getContent/690419. Referenced Apr 2022.
[14] 2022. Keystone-sm. https://github.com/keystone-enclave/sm/blob/8b8e6141256da97e005bc1d34a0bd1bf79210e34/src/pmp.h#L48. Referenced Jun 2022.
[15] 2022. Penglai-Enclave-sPMP. https://github.com/Penglai-Enclave/Penglai-Enclave-sPMP. Referenced Apr 2022.
[16] 2022. Penglai-sPMP Secure Monitor. https://github.com/Penglai-Enclave/Penglai-Enclave-sPMP/blob/767be067e8cd190f2a8ab1c7c7d9bb05a7be926a/opensbi-0.9/include/sm/pmp.h#L30. Referenced Jun 2022.
[17] 2022. PMP Enhancements for memory access and execution prevention on Machine mode (Smepmp). https://github.com/riscv/riscv-tee/blob/main/Smepmp/Smepmp.pdf.
[18] 2022. Redis. https://redis.io/. Referenced Apr 2022.
[19] 2022. RISC-V S-Mode Physical Memory Protection Unit (SPMP) Task Group. https://github.com/riscv-admin/spmp. Referenced Apr 2022.
[20] 2022. TEE-based confidential computing | Alibaba Cloud Documentation. https://www.alibabacloud.com/help/en/container-service-for-kubernetes/latest/tee-based-confidential-computing-tee-based-confidential-computing. Referenced Apr 2022.
[21] 2022. Vivado Design Suite. https://www.xilinx.com/products/design-tools/vivado.html. Referenced Apr 2022.

[22] 2023. RISC-V IOPMP Specification. https://github.com/riscv-non-isa/iopmp-spec. Referenced Sep 2023.

[23] Istemi Ekin Akkus, Ruichuan Chen, Ivica Rimac, Manuel Stein, Klaus Satzke, Andre Beck, Paarijaat Aditya, and Volker Hilt. 2018. *SAND*: Towards High-Performance Serverless Computing. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. 923–935.

[24] Chloe Alverti, Stratos Psomadakis, Vasileios Karakostas, Jayneel Gandhi, Konstantinos Nikas, Georgios Goumas, and Nectarios Koziris. 2020. Enhancing and exploiting contiguity for fast memory virtualization. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 515–528.

[25] Nadav Amit, Amy Tai, and Michael Wei. 2020. Don't Shoot down TLB Shootdowns!. In *Proceedings of the Fifteenth European Conference on Computer Systems* (Heraklion, Greece) *(EuroSys '20)*. Association for Computing Machinery, New York, NY, USA, Article 35, 14 pages. https://doi.org/10.1145/3342195.3387518

[26] Nadav Amit, Dan Tsafrir, and Assaf Schuster. 2014. VSwapper: A Memory Swapper for Virtualized Environments. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems* (Salt Lake City, Utah, USA) *(ASPLOS '14)*. Association for Computing Machinery, New York, NY, USA, 349–366. https://doi.org/10.1145/2541940.2541969

[27] Sergei Arnautov, Bohdan Trach, Franz Gregor, Thomas Knauth, Andre Martin, Christian Priebe, Joshua Lind, Divya Muthukumaran, Dan O'keeffe, Mark L Stillwell, et al. 2016. SCONE: Secure linux containers with intel SGX. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 689–703.

[28] Krste Asanovic, Rimas Avizienis, Jonathan Bachrach, Scott Beamer, David Biancolin, Christopher Celio, Henry Cook, Daniel Dabbelt, John Hauser, Adam Izraelevitz, et al. 2016. The rocket chip generator. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17* (2016).

[29] Rachata Ausavarungnirun, Joshua Landgraf, Vance Miller, Saugata Ghose, Jayneel Gandhi, Christopher J. Rossbach, and Onur Mutlu. 2018. Mosaic: Enabling Application-Transparent Support for Multiple Page Sizes in Throughput Processors. *SIGOPS Oper. Syst. Rev.* 52, 1 (aug 2018), 27–44. https://doi.org/10.1145/3273982.3273986

[30] Rachata Ausavarungnirun, Timothy Merrifield, Jayneel Gandhi, and Christopher J. Rossbach. 2020. PRISM: Architectural Support for Variable-Granularity Memory Metadata. In *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques* (Virtual Event, GA, USA) *(PACT '20)*. Association for Computing Machinery, New York, NY, USA, 441–454. https://doi.org/10.1145/3410463.3414630

[31] Thomas W. Barr, Alan L. Cox, and Scott Rixner. 2010. Translation Caching: Skip, Don'T Walk (the Page Table). In *Proceedings of the 37th Annual International Symposium on Computer Architecture* (Saint-Malo, France) *(ISCA '10)*. ACM, New York, NY, USA, 48–59. https://doi.org/10.1145/1815961.1815970

[32] Thomas W Barr, Alan L Cox, and Scott Rixner. 2011. SpecTLB: A mechanism for speculative address translation. *ACM SIGARCH Computer Architecture News* 39, 3 (2011), 307–318.

[33] Arkaprava Basu, Jayneel Gandhi, Jichuan Chang, Mark D. Hill, and Michael M. Swift. 2013. Efficient Virtual Memory for Big Memory Servers. In *Proceedings of the 40th Annual International Symposium on Computer Architecture* (Tel-Aviv, Israel) *(ISCA '13)*. ACM, New York, NY, USA, 237–248. https://doi.org/10.1145/2485922.2485943

[34] Andrew Baumann, Marcus Peinado, and Galen Hunt. 2014. Shielding Applications from an Untrusted Cloud with Haven. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. USENIX Association, Broomfield, CO, 267–283. https://www.usenix.org/conference/osdi14/technical-sessions/presentation/baumann

[35] Scott Beamer, Krste Asanovic, and David A. Patterson. 2015. The GAP Benchmark Suite. *CoRR* abs/1508.03619 (2015). arXiv:1508.03619 http://arxiv.org/abs/1508.03619

[36] Ravi Bhargava, Benjamin Serebrin, Francesco Spadini, and Srilatha Manne. 2008. Accelerating Two-Dimensional Page Walks for Virtualized Systems. In *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems* (Seattle, WA, USA) *(ASPLOS XIII)*. Association for Computing Machinery, New York, NY, USA, 26–35. https://doi.org/10.1145/1346281.1346286

[37] Abhishek Bhattacharjee. 2017. Translation-Triggered Prefetching. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems* (Xi'an, China) *(ASPLOS '17)*. Association for Computing Machinery, New York, NY, USA, 63–76. https://doi.org/10.1145/3037697.3037705

[38] David Biancolin, Sagar Karandikar, Donggyu Kim, Jack Koenig, Andrew Waterman, Jonathan Bachrach, and Krste Asanovic. 2019. FASED: FPGA-Accelerated Simulation and Evaluation of DRAM. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (Seaside, CA, USA) *(FPGA '19)*. Association for Computing Machinery, New York, NY, USA, 330–339. https://doi.org/10.1145/3289602.3293894

[39] Risk Boivie and Perter Williams. 2012. SecureBlue++: CPU support for secure execution. IBM, IBM Research Division, 1–9.

[40] Sol Boucher, Anuj Kalia, David G. Andersen, and Michael Kaminsky. 2018. Putting the "Micro" Back in Microservice. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. USENIX Association, Boston, MA, 645–650. https://www.usenix.org/conference/atc18/presentation/boucher

[41] Stefan Brenner, Colin Wulf, David Goltzsche, Nico Weichbrodt, Matthias Lorenz, Christof Fetzer, Peter Pietzuch, and Rüdiger Kapitza. 2016. Securekeeper: confidential zookeeper using intel sgx. In *Proceedings of the 17th International Middleware Conference*. 1–13.

[42] David Champagne and Ruby B Lee. 2010. Scalable architectural support for trusted software. In *HPCA-16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture*. IEEE, 1–12.

[43] Chia che Tsai, Donald E. Porter, and Mona Vij. 2017. Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. USENIX Association, Santa Clara, CA, 645–658. https://www.usenix.org/conference/atc17/technical-sessions/presentation/tsai

[44] Xiaoxin Chen, Tal Garfinkel, E Christopher Lewis, Pratap Subrahmanyam, Carl A Waldspurger, Dan Boneh, Jeffrey Dwoskin, and Dan RK Ports. 2008. Overshadow: a virtualization-based approach to retrofitting protection in commodity operating systems. *ACM SIGOPS Operating Systems Review* 42, 2 (2008), 2–13.

[45] Yaohui Chen, Sebassujeen Reymondjohnson, Zhichuang Sun, and Long Lu. 2016. Shreds: Fine-Grained Execution Units with Private Memory. In *2016 IEEE Symposium on Security and Privacy (SP)*. 56–71. https://doi.org/10.1109/SP.2016.12

[46] Siddhartha Chhabra, Brian Rogers, Yan Solihin, and Milos Prvulovic. 2011. SecureME: a hardware-software approach to full system security. In *Proceedings of the international conference on Supercomputing*. 108–119.

[47] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. *IACR Cryptol. ePrint Arch.* 2016 (2016), 86.

[48] Ankur Dave, Chester Leung, Raluca Ada Popa, Joseph E. Gonzalez, and Ion Stoica. 2020. Oblivious Coopetitive Analytics Using Hardware Enclaves. In *Proceedings of the Fifteenth European Conference on Computer Systems* (Heraklion, Greece) *(EuroSys '20)*. Association for Computing Machinery, New York, NY, USA, Article 39, 17 pages. https://doi.org/10.1145/3342195.3387552

[49] Udit Dhawan, Catalin Hritcu, Raphael Rubin, Nikos Vasilakis, Silviu Chiricescu, Jonathan M. Smith, Thomas F. Knight, Benjamin C. Pierce, and Andre DeHon. 2015. Architectural Support for Software-Defined Metadata Processing. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems* (Istanbul, Turkey) *(ASPLOS '15)*. Association for Computing Machinery, New York, NY, USA, 487–502. https://doi.org/10.1145/2694344.2694383

[50] Dong Du, Qingyuan Liu, Xueqiang Jiang, Yubin Xia, Binyu Zang, and Haibo Chen. 2022. Serverless Computing on Heterogeneous Computers. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Lausanne, Switzerland) *(ASPLOS '22)*. Association for Computing Machinery, New York, NY, USA, 797–813. https://doi.org/10.1145/3503222.3507732

[51] Dong Du, Tianyi Yu, Yubin Xia, Binyu Zang, Guanglu Yan, Chenggang Qin, Qixuan Wu, and Haibo Chen. 2020. Catalyzer: Sub-Millisecond Startup for Serverless Computing with Initialization-Less Booting. 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, 467–481. https://doi.org/10.1145/3373376.3378512

[52] Erhu Feng, Dong Du, Yubin Xia, and Haibo Chen. 2023. Efficient Distributed Secure Memory with Migratable Merkle Tree. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 347–360. https://doi.org/10.1109/HPCA56546.2023.10071130

[53] Erhu Feng, Xu Lu, Dong Du, Bicheng Yang, Xueqiang Jiang, Yubin Xia, Binyu Zang, and Haibo Chen. 2021. Scalable Memory Protection in the PENGLAI Enclave. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*. USENIX Association, 275–294. https://www.usenix.org/conference/osdi21/presentation/feng

[54] Christopher W Fletcher, Marten van Dijk, and Srinivas Devadas. 2012. A secure processor architecture for encrypted computation on untrusted programs. In *Proceedings of the seventh ACM workshop on Scalable trusted computing*. 3–8.

[55] Jayneel Gandhi, Arkaprava Basu, Mark D Hill, and Michael M Swift. 2014. Efficient memory virtualization: Reducing dimensionality of nested page walks. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 178–189.

[56] Jayneel Gandhi, Vasileios Karakostas, Furkan Ayar, Adrián Cristal, Mark D Hill, Kathryn S McKinley, Mario Nemirovsky, Michael M Swift, and Osman S Ünsal. 2016. Range translations for fast virtual memory. *IEEE Micro* 36, 3 (2016), 118–126.

[57] Siddharth Gupta, Atri Bhattacharyya, Yunho Oh, Abhishek Bhattacharjee, Babak Falsafi, and Mathias Payer. 2021. Rebooting virtual memory with midgard. In

*2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 512–525.

[58] Faruk Guvenilir and Yale N Patt. 2020. Tailored page sizes. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 900–912.

[59] Swapnil Haria, Mark D Hill, and Michael M Swift. 2018. Devirtualizing Memory in Heterogeneous Systems. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 637–650.

[60] Joseph M Hellerstein, Jose Faleiro, Joseph E Gonzalez, Johann Schleier-Smith, Vikram Sreekanti, Alexey Tumanov, and Chenggang Wu. 2018. Serverless Computing: One Step Forward, Two Steps Back. *arXiv preprint arXiv:1812.03651* (2018).

[61] Zhichao Hua, Jinyu Gu, Yubin Xia, Haibo Chen, Binyu Zang, and Haibing Guan. 2017. vTZ: Virtualizing ARM TrustZone. In *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, 541–556.

[62] Tyler Hunt, Zhiting Zhu, Yuanzhong Xu, Simon Peter, and Emmett Witchel. 2018. Ryoan: A distributed sandbox for untrusted computation on secret data. *ACM Transactions on Computer Systems (TOCS)* 35, 4 (2018), 1–32.

[63] A.H. Hunter, Chris Kennelly, Paul Turner, Darryl Gove, Tipp Moseley, and Parthasarathy Ranganathan. 2021. Beyond malloc efficiency to fleet efficiency: a hugepage-aware memory allocator. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*. USENIX Association, 257–273. https://www.usenix.org/conference/osdi21/presentation/hunter

[64] Vasileios Karakostas, Jayneel Gandhi, Furkan Ayar, Adrián Cristal, Mark D. Hill, Kathryn S. McKinley, Mario Nemirovsky, Michael M. Swift, and Osman Ünsal. 2015. Redundant Memory Mappings for Fast Access to Large Memories. In *Proceedings of the 42Nd Annual International Symposium on Computer Architecture* (Portland, Oregon) *(ISCA '15)*. ACM, New York, NY, USA, 66–78. https://doi.org/10.1145/2749469.2749471

[65] 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, 29–42. https://doi.org/10.1109/ISCA.2018.00014

[66] Jeongchul Kim and Kyungyong Lee. 2019. Practical cloud workloads for serverless faas. In *Proceedings of the ACM Symposium on Cloud Computing*. 477–477.

[67] Ana Klimovic, Yawen Wang, Patrick Stuedi, Animesh Trivedi, Jonas Pfefferle, and Christos Kozyrakis. 2018. Pocket: Elastic Ephemeral Storage for Serverless Analytics. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. USENIX Association, Carlsbad, CA, 427–444. https://www.usenix.org/conference/osdi18/presentation/klimovic

[68] Nikhita Kunati and Michael M Swift. 2018. Implementation of Direct Segments on a RISC-V Processor. In *Proceedings of Second Workshop on Computer Architecture Research with RISC-V*.

[69] Youngjin Kwon, Hangchen Yu, Simon Peter, Christopher J Rossbach, and Emmett Witchel. 2016. Coordinated and efficient huge page management with ingens. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 705–721.

[70] Dayeol Lee, David Kohlbrenner, Shweta Shinde, Krste Asanović, and Dawn Song. 2020. Keystone: An Open Framework for Architecting Trusted Execution Environments. In *Proceedings of the Fifteenth European Conference on Computer Systems* (Heraklion, Greece) *(EuroSys '20)*. Association for Computing Machinery, New York, NY, USA, Article 38, 16 pages. https://doi.org/10.1145/3342195.3387532

[71] Gyusun Lee, Wenjing Jin, Wonsuk Song, Jeonghun Gong, Jonghyun Bae, Tae Jun Ham, Jae W. Lee, and Jinkyu Jeong. 2020. A Case for Hardware-Based Demand Paging. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. 1103–1116. https://doi.org/10.1109/ISCA45697.2020.00093

[72] Dingji Li, Zeyu Mi, Yubin Xia, Binyu Zang, Haibo Chen, and Haibing Guan. 2021. TwinVisor: Hardware-Isolated Confidential Virtual Machines for ARM. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles* (Virtual Event, Germany) *(SOSP '21)*. Association for Computing Machinery, New York, NY, USA, 638–654. https://doi.org/10.1145/3477132.3483554

[73] Xupeng Li, Xuheng Li, Christoffer Dall, Ronghui Gu, Jason Nieh, Yousuf Sait, and Gareth Stockwell. 2022. Design and Verification of the Arm Confidential Compute Architecture. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*.

[74] David Lie, Chandramohan Thekkath, Mark Mitchell, Patrick Lincoln, Dan Boneh, John Mitchell, and Mark Horowitz. 2000. Architectural support for copy and tamper resistant software. *Acm Sigplan Notices* 35, 11 (2000), 168–177.

[75] David Lie, Chandramohan A Thekkath, and Mark Horowitz. 2003. Implementing an untrusted operating system on trusted hardware. In *SOSP*.

[76] Martin Maas, David G Andersen, Michael Isard, Mohammad Mahdi Javanmard, Kathryn S McKinley, and Colin Raffel. 2020. Learning-based memory allocation for C++ server workloads. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. 541–556.

[77] Artemiy Margaritov, Dmitrii Ustiugov, Edouard Bugnion, and Boris Grot. 2019. Prefetched Address Translation. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture* (Columbus, OH, USA) *(MICRO '52)*. Association for Computing Machinery, New York, NY, USA, 1023–1036. https://doi.org/10.1145/3352460.3358294

[78] Artemiy Margaritov, Dmitrii Ustiugov, Edouard Bugnion, and Boris Grot. 2019. Prefetched address translation. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 1023–1036.

[79] Artemiy Margaritov, Dmitrii Ustiugov, Amna Shahab, and Boris Grot. 2021. Ptemagnet: Fine-grained physical memory reservation for faster page walks in public clouds. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 211–223.

[80] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R Savagaonkar. 2013. Innovative instructions and software model for isolated execution. *Hasp@ isca* 10, 1 (2013).

[81] Larry W McVoy, Carl Staelin, et al. 1996. lmbench: Portable Tools for Performance Analysis.. In *USENIX annual technical conference*. San Diego, CA, USA, 279–294.

[82] Ashish Panwar, Sorav Bansal, and K Gopinath. 2019. Hawkeye: Efficient fine-grained os support for huge pages. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. 347–360.

[83] Ashish Panwar, Aravinda Prasad, and K Gopinath. 2018. Making huge pages actually useful. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*. 679–692.

[84] Chang Hyun Park, Sanghoon Cha, Bokyeong Kim, Youngjin Kwon, David Black-Schaffer, and Jaehyuk Huh. 2020. Perforated page: Supporting fragmented memory allocation for large pages. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 913–925.

[85] Chang Hyun Park, Ilias Vougioukas, Andreas Sandberg, and David Black-Schaffer. 2022. *Every Walk's a Hit: Making Page Walks Single-Access Cache Hits*. Association for Computing Machinery, New York, NY, USA, 128–141. https://doi.org/10.1145/3503222.3507718

[86] Binh Pham, Abhishek Bhattacharjee, Yasuko Eckert, and Gabriel H Loh. 2014. Increasing TLB reach by exploiting clustering in page translations. In *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 558–567.

[87] Binh Pham, Viswanathan Vaidyanathan, Aamer Jaleel, and Abhishek Bhattacharjee. 2012. Colt: Coalesced large-reach tlbs. In *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 258–269.

[88] Christian Priebe, Kapil Vaswani, and Manuel Costa. 2018. Enclavedb: A secure database using SGX. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 264–278.

[89] Brian Rogers, Siddhartha Chhabra, Milos Prvulovic, and Yan Solihin. 2007. Using address independent seed encryption and bonsai merkle trees to make secure processors os-and performance-friendly. In *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*. IEEE, 183–196.

[90] Vasily A. Sartakov, Lluís Vilanova, David Eyers, Takahiro Shinagawa, and Peter Pietzuch. 2022. CAP-VMs: Capability-Based Isolation and Sharing in the Cloud. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. USENIX Association, Carlsbad, CA, 597–612. https://www.usenix.org/conference/osdi22/presentation/sartakov

[91] Vasily A. Sartakov, Lluís Vilanova, and Peter Pietzuch. 2021. CubicleOS: A Library OS with Software Componentisation for Practical Isolation. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Virtual, USA) *(ASPLOS 2021)*. Association for Computing Machinery, New York, NY, USA, 546–558. https://doi.org/10.1145/3445814.3446731

[92] David Schrammel, Samuel Weiser, Stefan Steinegger, Martin Schwarzl, Michael Schwarz, Stefan Mangard, and Daniel Gruss. 2020. Donky: Domain Keys – Efficient In-Process Isolation for RISC-V and x86. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, 1677–1694. https://www.usenix.org/conference/usenixsecurity20/presentation/schrammel

[93] Felix Schuster, Manuel Costa, Cédric Fournet, Christos Gkantsidis, Marcus Peinado, Gloria Mainar-Ruiz, and Mark Russinovich. 2015. VC3: Trustworthy data analytics in the cloud using SGX. In *2015 IEEE Symposium on Security and Privacy*. IEEE, 38–54.

[94] 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*. 1093–1108.

[95] Jovan Stojkovic, Dimitrios Skarlatos, Apostolos Kokolis, Tianyin Xu, and Josep Torrellas. 2022. Parallel Virtualized Memory Translation with Nested Elastic

Cuckoo Page Tables. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Lausanne, Switzerland) *(ASPLOS '22)*. Association for Computing Machinery, New York, NY, USA, 84–97. https://doi.org/10.1145/3503222.3507720

[96] Brian Suchy, Simone Campanoni, Nikos Hardavellas, and Peter Dinda. 2020. CARAT: A Case for Virtual Memory through Compiler- and Runtime-Based Address Translation. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation* (London, UK) *(PLDI 2020)*. Association for Computing Machinery, New York, NY, USA, 329–345. https://doi.org/10.1145/3385412.3385987

[97] G Edward Suh, Dwaine Clarke, Blaise Gassend, Marten Van Dijk, and Srinivas Devadas. 2003. AEGIS: architecture for tamper-evident and tamper-resistant processing. In *ACM International Conference on Supercomputing 25th Anniversary Volume*. 357–368.

[98] Chia-Che Tsai, Jeongseok Son, Bhushan Jain, John McAvey, Raluca Ada Popa, and Donald E Porter. 2020. Civet: An Efficient Java Partitioning Framework for Hardware Enclaves. In *29th USENIX Security Symposium (USENIX Security 20)*.

[99] Anjo Vahldiek-Oberwagner, Eslam Elnikety, Nuno O. Duarte, Michael Sammler, Peter Druschel, and Deepak Garg. 2019. ERIM: Secure, Efficient In-process Isolation with Protection Keys (MPK). In *28th USENIX Security Symposium (USENIX Security 19)*. USENIX Association, Santa Clara, CA, 1221–1238. https://www.usenix.org/conference/usenixsecurity19/presentation/vahldiek-oberwagner

[100] Georgios Vavouliotis, Lluc Alvarez, Vasileios Karakostas, Konstantinos Nikas, Nectarios Koziris, Daniel A Jiménez, and Marc Casas. 2021. Exploiting page table locality for agile TLB prefetching. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 85–98.

[101] Veripool. [n. d.]. Verilator. https://www.veripool.org/verilator/.

[102] Nandita Vijaykumar, Abhilasha Jain, Diptesh Majumdar, Kevin Hsieh, Gennady Pekhimenko, Eiman Ebrahimi, Nastaran Hajinazar, Phillip B Gibbons, and Onur Mutlu. 2018. A case for richer cross-layer abstractions: Bridging the semantic gap with expressive memory. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 207–220.

[103] Lluïs Vilanova, Muli Ben-Yehuda, Nacho Navarro, Yoav Etsion, and Mateo Valero. 2014. CODOMs: Protecting software with code-centric memory domains. In *ACM SIGARCH Computer Architecture News*, Vol. 42. IEEE Press, 469–480.

[104] Liang Wang, Mengyuan Li, Yinqian Zhang, Thomas Ristenpart, and Michael Swift. 2018. Peeking behind the curtains of serverless platforms. In *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*. 133–146.

[105] Andrew Waterman, Yunsup Lee, Rimas Avizienis, David A. Patterson, and Krste Asanović. 2016. *The RISC-V Instruction Set Manual Volume II: Privileged Architecture Version 1.9*. Technical Report UCB/EECS-2016-129. EECS Department, University of California, Berkeley. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-129.html

[106] Robert NM Watson, Ben Laurie, Steven J Murdoch, Robert Norton, Michael Roe, Stacey Son, Munraj Vadera, Jonathan Woodruff, Peter G Neumann, Simon W Moore, et al. 2015. Cheri: A hybrid capability-system architecture for scalable software compartmentalization. In *2015 IEEE Symposium on Security and Privacy (SP)*. IEEE, 20–37.

[107] Robert NM Watson, Robert M Norton, Jonathan Woodruff, Simon W Moore, Peter G Neumann, Jonathan Anderson, David Chisnall, Brooks Davis, Ben Laurie, Michael Roe, et al. 2016. Fast protection-domain crossing in the cheri capability-system architecture. *IEEE Micro* 36, 5 (2016), 38–49.

[108] Emmett Witchel, Josh Cates, and Krste Asanović. 2002. Mondrian Memory Protection. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems* (San Jose, California) *(ASPLOS X)*. ACM, New York, NY, USA, 304–316. https://doi.org/10.1145/605397.605429

[109] Emmett Witchel, Junghwan Rhee, and Krste Asanović. 2005. Mondrix: Memory Isolation for Linux Using Mondriaan Memory Protection. In *Proceedings of the Twentieth ACM Symposium on Operating Systems Principles* (Brighton, United Kingdom) *(SOSP '05)*. ACM, New York, NY, USA, 31–44. https://doi.org/10.1145/1095810.1095814

[110] Yubin Xia, Yutao Liu, and Haibo Chen. 2013. Architecture support for guest-transparent VM protection from untrusted hypervisor and physical attacks.. In *HPCA*. 246–257.

[111] Zi Yan, Daniel Lustig, David Nellans, and Abhishek Bhattacharjee. 2019. Translation Ranger: Operating System Support for Contiguity-Aware TLBs. In *Proceedings of the 46th International Symposium on Computer Architecture* (Phoenix, Arizona) *(ISCA '19)*. Association for Computing Machinery, New York, NY, USA, 698–710. https://doi.org/10.1145/3307650.3322223

[112] Idan Yaniv and Dan Tsafrir. 2016. Hash, Don't Cache (the Page Table). In *Proceedings of the 2016 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Science* (Antibes Juan-les-Pins, France) *(SIGMETRICS '16)*. Association for Computing Machinery, New York, NY, USA, 337–350. https://doi.org/10.1145/2896377.2901456

[113] Tianyi Yu, Qingyuan Liu, Dong Du, Yubin Xia, Binyu Zang, Ziqian Lu, Pingchao Yang, Chenggang Qin, and Haibo Chen. 2020. Characterizing Serverless Platforms with ServerlessBench. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC '20)*. Association for Computing Machinery. https://doi.org/10.1145/3419111.3421280

[114] Jerry Zhao, Ben Korpan, Abraham Gonzalez, and Krste Asanovic. 2020. SonicBOOM: The 3rd Generation Berkeley Out-of-Order Machine. (May 2020).