Studies on Evaluation Platforms for DRAM/NVMM Heterogeneous and Secure Computing Memory Systems

DRAMとNVMMによるヘテロジニアスメモリシステム及びセキュアコンピューティングメモリシステム評価環境の研究

February, 2023

Yu OMORI
大森　侑

Studies on Evaluation Platforms for DRAM/NVMM Heterogeneous and Secure Computing Memory Systems

DRAMとNVMMによるヘテロジニアスメモリシステム及びセキュアコンピューティングメモリシステム評価環境の研究

February, 2023

Waseda University Graduate School of Fundamental Science and Engineering

Department of Computer Science and Communications Engineering, Research on Advanced Processor Architecture

Yu OMORI
大森　侑

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Traditional Homogeneous Memory System

For a long time, multiple memory devices have been used complementary in computer systems as depicted in Fig. 1.1. Two types of memory devices have been used: DRAM-based main memory and HDD/SSD-based auxiliary devices. DRAM-based main memory is volatile, high-speed, small capacity devices. HDD/SSD-based devices are non-volatile, low-speed, large capacity devices.

DRAM stores data by holding an electric charge using one transistor and one capacitor. It has higher density than SRAM used in CPU caches, and access latency about tens to hundreds of nanoseconds. Density and operating speed have been improved over the years. JEDEC published the specification of DDR5-8400 [JED22]. On the other hand, DRAM must be always energized to hold data. Data is lost once power is gone. Besides, electric charges must be refilled regularly by refresh operations due to destructive operations and current leakages. Larger DRAM consumes more power by refresh operations. Data on DRAM is written into auxiliary devices for memory capacity and non-volatility.

SSD stores data by holding an electron using a NAND flash. It can hold data across power failures unlike DRAM. Density and operating speed have been rapidly improved by 3D stacking, multi-level cells, and so on. Kioxia corporation achieved HLC (Hexa Level Cell) in April 2021 [ATM+]. Despite these advantages, SSD operations and management are more complex than DRAM. Data must be transferred by block units, and write latency is quite long due to its overwriting mechanism. A program accesses data on SSD using device drivers and page caches provided by an OS. It introduces system complexity and context switch overheads. SSD access latency is about tens of microseconds even on the fastest SSD. Frequent SSD accesses severely degrade system performance.

The traditional memory hierarchy consisting of DRAM-based homogeneous memory systems and SSD-based auxiliary devices has been used in computer systems. However, it is less attractive for recent workloads. The homogeneous memory systems have difficulty in dealing with quite high memory usage of ML/DL-based AI. IoT and cloud computing have diversified computation platforms and requirements, such as low-power, simple, and/or large memory devices. Heterogeneous memory systems are expected approaches. They extend the traditional memory hierarchy with new memory devices. For instance, HBM (High Bandwidth Memory) [AMD15] is a high-performance memory device, between "Cache Memory" and "Main Memory" in Fig. 1.1. High-performance platforms use it to alleviate the memory-wall problem [WM95]. This dissertation focuses on heterogeneous memory systems consisting of DRAM and NVMM (Non-Volatile Main Memory). NVMM is a new memory device, with both characteristics of DRAM and SSD. ("Main Memory" and "Flash Device" in Fig. 1.1). The heterogeneous memory systems are expected as large and simple memory systems on edge devices.

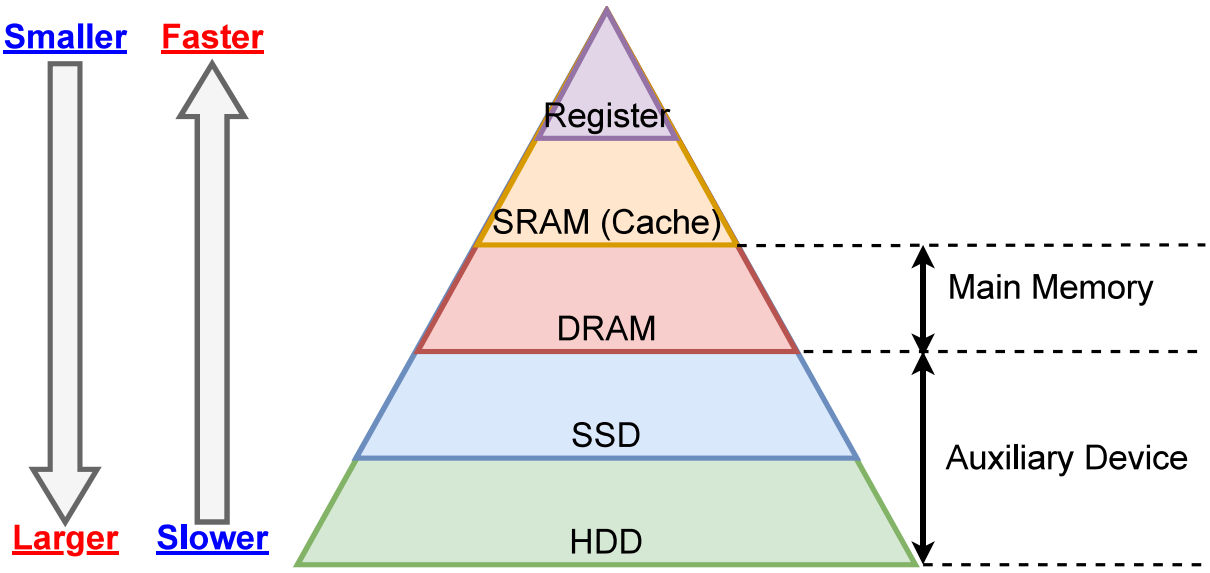


Figure 1.1: Memory Hierarchy

1.2 DRAM/NVMM Heterogeneous Memory System

NVMM (Non-Volatile Main Memory) is a new memory device with new non-volatile memory cells. NVMM has similar characteristics to DRAM rather than SSD. NVMM works on a memory bus with byte accessibility and short latency of about tens to hundreds of nanoseconds. Various non-volatile memory cells have been proposed: Phase Change Memory [WRK+10], Ferroelectric Random Access Memory [SJH+96], Magnetoresistive Random Access Memory [BasFC+90], Spin Transfer Torque-MRAM [CAD+10], Resistive Random Access Memory [SWW71]. While most of them are under research yet, Intel and Micron brought the first commercially available NVMM, Intel Oprante DC Persistent Memory (DCPMM) [Int19] to production in 2019. It has 3D-Xpoint memory cells [Web18], a kind of phase change memory. DCPMM is physically compatible with the DDR4 memory interface. A memory controller can directly access data on DCPMM in hundreds of nanoseconds using a DDR-T protocol. Barlow Pass DCPMM realized 512-GiB memory capacity on the same footprint as a DDR4 DIMM.

NVMM is a large and non-volatile memory device. A memory controller can directly access data on NVMM in similar latency to DRAM. DRAM/NVMM heterogeneous memory systems have become attractive memory systems for contemporary workloads. Larger memory space can be realized than DRAM-based homogeneous memory systems. An Ice Lake SP Xeon CPU with DCPMM can have up to 3TB of memory space on each CPU socket. DCPMM is introduced into in-memory database systems, like SAP HANA. On the other hand, NVMM can be used as auxiliary non-volatile devices. NVMM-based devices can realize simple operations, byte accessibility, and low latency. Complex device drivers provided by an OS are not required unlike traditional block devices. Data persistency on NVMM can be guaranteed much easier than that on block devices. Page caches are used for block devices to alleviate long latency and block-unit accessibility. This mechanism requires two data evictions to guarantee data persistency. Data are evicted from CPU caches to main memory (page caches), then, page caches are evicted to block devices. System calls for page cache evictions introduce context switch overheads. In contrast, page caches should be omitted for NVMM. Data persistency on NVMM can be guaranteed only by CPU cache evictions. Line unit evictions are faster than block unit writebacks to block devices.

DRAM/NVMM heterogeneous memory systems are expected as new sufficient memory systems for various platforms, from large servers to small edge devices. This dissertation focuses on small edge devices like IoT edge or embedded devices. These small devices have been widely used in recent years with the growing IoT. They often have strict requirements for power consumption, size, runtime, and so on. NVMM has attractive features for them: OS-free accessibility, high density, low latency, and high endurance (DCPMM has tens to hundreds Peta-Byte Written).

1.3 Existing Evaluation Platforms for DRAM/NVMM Heterogeneous Memory Systems

To effectively utilize DRAM/NVMM heterogeneous memory systems, the whole system must be optimized for the heterogeneous memory system. NVMM has different characteristics from both DRAM and block devices. NVMM latency (∼ several hundreds of nanoseconds) is several times longer than DRAM (∼ a hundred of nanoseconds), on the other hand, quite shorter than block devices (∼ a hundred of microseconds). In addition, NVMM latency is asymmetric. Write latency is several times longer than read latency. Optimization techniques for DRAM may not be effective on NVMM. Long and asymmetric latency may severely degrade system performance. Optimization techniques for SSD are redundant for NVMM due to byte accessibility and short latency. Page caches and device drivers incur unnecessary overheads. Data persistency can be guaranteed by only CPU cache eviction.

Exploration of system-wide optimization techniques for DRAM/NVMM heterogeneous memory systems is challenging on existing evaluation platforms. Most of NVMM is still under research except DCPMM. However, DCPMM works on only specific Intel server-grade CPUs due to the DDR-T protocol. The lack of NVMM for edge devices results in the widespread use of simulators and emulators. The former evaluates workloads by building the whole system, and the latter evaluates workloads by extending a base system.

Several software simulators have been proposed: NVMain [PX12] [PZX15], NVMainExt [KHC18], HMMSim [BCMM15], NMTSim [GLH+20], SIM-PCM [LLZ12], SpinSim [MWA+21]. They simulate NVMM performance by adjusting memory latency. NVMM simulation is based on modeled NVMM architectures and behaviors. NVMM models in existing simulators represent NVMM with similar structures to DRAM, having banks, rows, and columns. Memory system simulators (NVMain and HMMSim) are often used with system simulators such as gem5 [BBB+11]. Software simulators provide high flexibility, and detailed system statistics. On the other hand, these advantages prolong evaluation time. Simulator's evaluation may take hundreds to tens of thousands longer evaluation time than real hardware. The whole system simulation consisting of multi CPUs, caches, peripherals, and OSs takes tens of hours to days. Therefore, existing software simulators are not suitable for exploring system-wide optimization techniques for DRAM/NVMM heterogeneous memory systems.

Several software emulators have been proposed: PCMSim [WW17], Quartz [VMCL15] [KHA+17], LEAF [ZLWD17], HME [DLLJ18], PMEP [DKK+14]. They treat a part of DRAM memory space as an NVMM region, then inject additional latency into memory accesses to the region. Emulators periodically get the number of NVMM accesses, and suspend program execution to emulate NVMM performance. The suspend time is calculated by "the number of NVMM accesses × (average NVMM latency - average DRAM latency)". Software emulators are implemented as additional modules on base platforms. They can evaluate workloads quite faster than software simulators at the speed of

base platforms. On the other hand, detailed memory access monitoring is difficult on their designs. Their latency injection based on average latency cannot capture the impact of detailed memory accesses. NVMM latency should be non-deterministic depending on the hit ratio to on-module buffers. Intel DCPMM has an internal buffer indeed. DCPMM read latency increases up to 2×, and write latency increases up to 8× when missing the on-module buffer [WLY+20, OK21]. Buffer hit ratio is an essential factor to exploit NVMM advantages while minimizing performance degradation. It is ignored on existing software emulators. Therefore, existing software emulators are not suitable for exploring system-wide optimization techniques for DRAM/NVMM heterogeneous memory systems.

The above discussion shows the tradeoff of existing works between evaluation speed and evaluation details. Hardware simulators are expected solutions. They are built on existing SoCs: TUNA [LKP+14, LY17], and Petropoulos's emulator [PA17] [1]. NVMM simulation is realized by modifying hardware modules. Hardware simulators can run workloads at the speed of base SoCs like software emulators. They can monitor all memory accesses at a memory bus and/or a memory controller like software simulators. TUNA extends the existing ARM SoC on an FPGA. Additional latency is injected on a memory bus or a memory controller. The petropoilos's emulator have focused on only PCM (Phase Change Memory). TUNA v1 [LKP+14] injects the same latency into ALL memory accesses like software emulators. TUNA v2.1 [LY17] introduced a new latency injection technique by extending a memory controller. It should be able to capture an impact of buffer hit ratio unlike existing works. However, the work [LY17] just evaluated average latencies for random memory accesses. The impact was not fully discussed in existing works. Besides, they did not focus on CPU cache eviction. Explicit cache eviction and its overhead must be considered when using NVMM. The effectiveness of the new latency injection technique is still unclear in exploring optimization techniques for DRAM/NVMM heterogeneous memory systems.

In addition, existing works presuppose that NVMM architectures are similar to DRAM. The assumption should be true on some NVMM, however, Intel DCPMM has a different architecture from DRAM [LX19]. Among developing NVMM, DCPMM and its technology should have important roles in the future as the first commercially available NVMM. Exploring optimization techniques for DCPMM is meaningful. Existing evaluation platforms for DCPMM [WLY+20] are software simulators. They have same issues of evaluation speed on system-wide optimizations.

[1] Some papers categorize them into "hardware emulators" because they extend existing SoCs

1.4 Secure Computing on IoT Edge Devices

Secure computing on edge devices is one of the expected DRAM/NVMM heterogeneous memory systems' usecases. With growing IoT, computation platforms for ML/DL-based AI inference have been diversifying. Edge devices traditionally collect data by using sensors and send them to cloud servers. Cloud servers do inference using the given data and trained model, then send results to edge devices. However, AI inference has been gradually shifted to edge devices because of high attention on data privacy. In traditional AI systems, data is exposed to various attacks related to network connections and cloud storage. These threats can be ignored when inference is processed completely on edge devices. Some smart speakers and smartphones already have features to do some inferences, such as face authentication. Autonomous driving cars must do inference locally to be independent of network states. On-device inference requires trained AI models stored on local storage devices. The models must be protected because they contain confidential data and/or possibly personal information. Adversarial attackers can touch the devices much easier than servers in data centers. Edge devices are constantly exposed to not only software attacks but also hardware attacks. Attackers can directly observe and analyze the device's behaviors. The whole system including privileged hardware and software may be unknowingly controlled by attackers. To deal with these attacks, secure computing using TEE (Trusted Execution Environment) is expected.

TEE is a framework to securely run a program on untrusted computation platforms while keeping program code and data secret. Each program on TEE owns a part of memory regions as its isolated private region when launching. Only the owner process can access the isolated region. All memory accesses to the region are checked by a CPU. If non-owner processes try to access the region, the memory accesses are rejected and errors are raised. This hardware-enforced access management is always applied to memory accesses regardless of their privilege levels. Therefore, the code and data are always kept secret when a program on TEE runs on polluted systems. Its memory isolation can be broken by tampering with on-chip CPUs and memory controllers, however, it is impossible [LKS+20]. These strict data protections are attractive for AI inference on edge devices. Three TEE are mainly proposed: Intel SGX [CD16], ARM TrustZone [Alv04], RISC-V Keystone [LKS+20]. SGX and TrustZone work on specific Intel or ARM CPUs. They are not suitable to encompass various design demands of edge devices. Keystone works on a RISC-V CPU, that is open-source and well standardized ISA [WA19, WAH21]. RISC-V CPUs are customizable depending on platform requirements and demands. This dissertation focuses on Keystone TEE as TEE for edge devices. Despite strict data protection, Keystone has two issues with memory systems: auxiliary devices and off-chip memory protection.

As described above, Keystone presupposes wholly polluted systems. An untrusted OS and kernel may be under the attacker's control. Once data is passed to them, data may be stolen and/or tampered with. However, Keystone must use their functions to access data on auxiliary devices.

The fully verified Keystone runtime does not have complex device drivers. It is not desirable to introduce device drivers into the runtime because their codes cause verification complexity and may raise vulnerabilities. Therefore, Keystone cannot securely use data on auxiliary devices. To overcome this issue, a DRAM/NVMM heterogeneous memory system is expected. NVMM is a non-volatile storage device that can be directly accessed by a memory controller without the help of complex device drivers (Section 1.2). More applications can be securely executed on Keystone by integrating DRAM/NVMM heterogeneous memory systems into Keystone.

In addition, Keystone mainly focuses on on-chip security and does not provide off-chip memory protection. The hardware-enforced access control uses RISC-V CPU features. Off-chip memory accesses bypassing CPUs are not checked, such as DMA transfers, tapping on off-chip memory buses, and cold-boot attacks. All data becomes untrusted once they are stored on off-chip memory. Secure computing using Keystone can trust only on-chip memory, thus, program code and data must be smaller than on-chip memory (CPU caches). Few applications can be securely executed on Keystone. The off-chip memory accesses are critical to DRAM/NVMM heterogeneous memory systems due their its non-volatility. Data on DRAM will be lost once power is gone except for special techniques such as cold-boot attacks. In contrast, data on NVMM is not lost across power failures. Adversarial attackers can steal data on NVMM after withdrawing the power supply. They also can pollute the system by tampering with data on NVMM. For instance, a tampered AI model should cause incorrect inference results, then possibly cause undefined behavior. Some server-grade CPUs have a feature to protect data on off-chip memory: Intel TME [Int21], AMD SEV [KPW16]. They encrypt cachelines before leaving on-chip memory. Even if attackers can get encrypted data, they cannot decrypt the data without encryption keys. Nevertheless, memory encryption is meaningless to memory tampering. Attackers can tamper with data without any decryption to cause system failures. For these reasons, secure computing on TEE and DRAM/NVMM heterogeneous memory systems also requires a feature to protect data from both stealing and tampering. It can be realized by MPE (Memory Protection Engine) using an integrity tree [SAB15].

1.5 Off-Chip Memory Protection Using Integrity Tree

An integrity tree provides memory encryption and tamper detection on off-chip memory. It is mostly implemented as a hardware MPE (Memory Protection Engine) module in a memory controller. Software-implemented MPE [GDD+20] is not desirable on real systems because it incurs infeasible performance overhead especially for large data exceeding on-chip memory (CPU caches). Various integrity trees have been proposed: Merkle Tree [Mer80], Bonsai Merkle Tree [RCPS07], TEC-Tree [ECL+07], and SGX-style Integrity Tree [Gue16]. BMT (Bonsai Merkle Tree) and SIT (SGX-style Integrity Tree) have been mainly used in existing works. This dissertation uses SIT since it is used on the commercially available TEE, Intel SGX.

SIT encryption and tamper detection use keys, spatial nonces, and temporal nonces. The keys are stored on trusted on-chip memory which is only visible from trusted hardware modules. Attackers cannot decrypt or tamper with data on off-chip memory with the keys. Two nonces are used against replay attacks. Spatial nonces are physical addresses of cachelines. If the same data is written into two cachelines, encrypted or hashed lines differ. Temporal nonces are version numbers of cachelines, in other words, the number of writes to cachelines. If the same data is written into one cacheline multiple times, encrypted or hashed lines differ. Version numbers on SIT nodes are protected by hashed octrees. All cachelines are verified before fetching them from off-chip memory using the version numbers. SIT can detect any tampering on off-chip memory. The hash algorithm of SIT, CWMAC [WC81], has enough collision resistance. Thus, SIT can protect data on off-chip memory even if it is exposed to physical attackers.

1.6 Existing Evaluation Platforms for Secure Edge Computing

Section 1.4 and Section 1.5 described the necessity of TEE, DRAM/NVMM heterogeneous memory systems, and an integrity tree in secure edge computing. Exploration of optimization techniques for the combination is difficult on existing works. There are various RISC-V platforms compatible with Keystone: from production boards such as SiFive Unmatched/Unleashed and Microchip PolarFire, to open-source designs such as Rocket-chip and BOOM. However, they do not support DRAM/NVMM heterogeneous memory systems and MPE. Software simulators with the combination require long evaluation time. The paper [HLC+21, AAPLP21] shows that even simple Keystone benchmarks take tens to hundreds 10^4 seconds on Keystone-compatible gem5. DRAM/NVMM heterogeneous memory systems and MPE prolong their evaluation time. While the hardware simulator PENGLAI [FLD+21] has both Keystone and MPE, DRAM/NVMM heterogeneous memory systems are not supported. It is difficult to extend the closed design for heterogeneous memory systems.

1.7 Dissertation Proposals

This dissertation proposes evaluation platforms for DRAM/NVMM heterogeneous memory systems and secure computing memory systems.

First, this dissertation proposes hardware DRAM/NVMM heterogeneous memory simulators to solve issues of existing works discussed in Section 1.1, Section 1.2 and Section 1.3: (1.a) a tradeoff between evaluation speed and accuracy, (1.b) the lack of DCPMM simulation, (1.c) insufficient validation and discussion about NVMM simulations, and (1.d) the NVMM's impact on real applications. The proposed simulator works on an FPGA to solve the tradeoffs of existing works (1.a). The simulator provides three NVMM simulation models: Coarse-Grain, Fine-Grain, and DCPMM. The first two models are extensions of existing works considering NVMM architectures and behaviors. The DCPMM model is a new model that represents actual DCPMM behaviors (1.b). Chapter 2 proposes a heterogeneous memory simulator on hard processor systems. Chapter 3 extends the NVMM models for soft processor systems, and proposes the simulator employing them. Each NVMM model is validated on the proposed simulators against golden models (1.c). Then, the chapters confirm the effectiveness of NVMM models, and reveal the direction and essential factors for DRAM/NVMM heterogeneous memory systems (1.d). The simulators provide a way to explore optimization techniques for DRAM/NVMM heterogeneous memory systems on edge devices.

Second, this dissertation proposes a secure edge computing simulator employing all of TEE, NVMM, and MPE to solve issues of existing works discussed in Section 1.4, Section 1.5 and Section 1.6: (2.a) the lack of an evaluation platform for secure computing using the combination. Chapter 4 implements MPE based on SIT, then integrates it into the DRAM/NVMM heterogeneous simulator proposed in Chapter 3. It has Keystone-compatible RISC-V CPUs. The chapter validates MPE behavior and the impacts of MPE on memory latency, for DRAM and simulated DCPMM. The simulator widely provides a way to explore optimization techniques for secure computing on edge devices (2.a).

1.8 Dissertation Outline

This dissertation consists of 5 chapters.

Chapter 1 "Introduction", current chapter, describes this dissertation's background, related works, and proposals.

Chapter 2 "DRAM/NVMM Heterogeneous Memory Simulator on Hard Processor Systems" proposes a hardware DRAM/NVMM heterogeneous memory simulator working on hard processor systems, especially ARM SoC. This chapter consists of 4 parts: NVMM behavior models, simulator implementation, validation of the behavior models, and experimental evaluation. The first part discusses and defines three NVMM behavior models that represent NVMM architectures and behaviors. The Coarse-Grain and Fine-Grain models represent NVMM behaviors with similar architectures to traditional DRAM-based memory. Only the Fine-Grain model can capture memory access characteristics to reduce NVMM latency, such as access locality. The DCPMM model is a new behavior model that represents the actual DCPMM behavior with some constraints for edge devices. The second part describes the whole simulator implementation: delay injection techniques to simulate NVMM performance based on three behavior models, kernel modification, kernel module, and management library to use the NVMM region effectively. The third part validates the NVMM behavior models by comparing them with golden models (existing NVMM simulators and a real DCPMM). The validation also confirms that the Fine-Grain behavior model can capture an impact of memory access characteristics that is ignored in the Coarse-Grain behavior model. The fourth part evaluates real applications chosen from SPEC CPU 2017 benchmarks on the proposed simulator using three NVMM behavior models. The result shows that the frequency of NVMM accesses, access locality, and bank parallelism are essential factors in exploiting NVMM performance. The last two factors can alleviate the first factor's impact; however, frequent memory accesses spoil them and severely degrade system performance.

Chapter 3 "DRAM/NVMM Heterogeneous Memory Simulator on Soft Processor Systems" proposes a hardware DRAM/NVMM heterogeneous memory simulator working on soft processor systems, especially RISC-V SoC. The Fine-Grain behavior model proposed in Chapter 3 cannot be directly applied to soft processor systems because the model presupposes that a CPU runs sufficiently faster than a memory system. The Fine-Grain model on soft processor systems shows the same behavior as the Coarse-Grain one, even if an application has high access locality. This chapter proposes a new NVMM behavior model, "Extended Fine-Grain", that can exploit access locality even on soft processor systems. Besides, the simulator's RISC-V core design is modified so a user program can directly evict a cacheline. Validation using micro benchmarks and experimental evaluation using SPEC CPU 2017 benchmarks shows that the Extended Fine-Grain model can capture an impact of access locality even on soft processor systems, unlike the existing Coarse-Grain and Fine-Grain models. The DCPMM model on the simulator is validated against a real DCPMM.

Chapter 4 “Secure Edge Computing Simulator Employing DRAM/NVMM Heterogeneous Memory Systems” proposes a hardware simulator having all of TEE, DRAM/NVMM heterogeneous memory system, and MPE. Among existing TEEs, this dissertation focuses on open-source RISC-V Keystone TEE that can satisfy various requirements for edge devices. This chapter implements an MPE, then integrates it into the DRAM/NVMM simulator proposed in Chapter 3. The Rocket core on the simulator is compatible with Keystone TEE. The MPE is based on SGX-style Integrity Tree used in Intel SGX TEE. Pipelined modules in the MPE cooperatively work as much as possible to maximize throughput. The MPE is designed to cover a large memory region with limited hardware resources by introducing dynamic tree roots and modules while keeping Tree parallelism. Experimental evaluation on the proposed simulator shows that the MPE incurs 2.55×/4.16× for DRAM read/write, respectively. It also showed that MPE incurs 3.05×/5.40× for simulated DCPMM read/write, respectively.

Chapter 5 “Conclusion” concludes this dissertation.

Chapter 2

DRAM/NVMM Heterogeneous Memory Simulator on Hard Processor Systems *

*This chapter is based on the paper "Non-Volatile Main Memory Emulator for Embedded Systems Employing Three NVMM Behaviour Models", The 8th IEEE Non-Volatile Memory Systems and Applications Symposium (IEEE NVMSA 2019), Aug. 2019

2.1 Preface

As described in Section 1.2, DRAM/NVMM heterogeneous memory systems are expected to satisfy diversifying demands of computer architectures and workloads. The heterogeneous memory system is attractive for edge devices thanks to its high density, simple accessibility, and similar latency to DRAM.

This chapter proposes a hardware simulator on an FPGA to explore optimization techniques for DRAM/NVMM heterogeneous memory systems on edge devices. It solves the tradeoff of existing works described in Section 1.3. Besides, the simulator provides DCPMM performance simulation by modeling actual DCPMM behaviors. DCPMM has not been considered in existing works. All NVMM behavior models are validated against golden models. Then, optimization techniques for the heterogeneous memory system are explored using benchmark programs.

This chapter is organized as follows: Section 2.2 discusses the presupposed NVMM architectures and introduces "NVMM Behavior Model"s. Section 2.3 describes the whole simulator implementation: NVMM performance simulations, the modified Linux kernel and a kernel module for NVMM cacheability, and an NVMM management library. Section 2.4 and Section 2.5 validate the NVMM behavior models and reveal essential factors for DRAM/NVMM heterogeneous memory systems. Section 2.6 concludes this chapter.

2.2 NVMM Behavior Model

NVMM performance simulations are realized by injecting additional latency into memory accesses. These latency injections are based on some presupposed NVMM architectures. Various NVMM cells have been proposed, however, there are no standardized or de facto technologies. To simulate NVMM behaviors, NVMM models must be defined. This dissertation defines them as "NVMM Behavior Model"s that represent the NVMM behavior and architecture including on-module buffers, memory cells, and so on. Three NVMM behavior models are proposed in this chapter: Coarse-Grain, Fine-Grain, and DCPMM. The first two models are based on NVMM having similar architectures to DRAM. The last one is based on a real DCPMM.

2.2.1 Overview of traditional DRAM-based Main Memory

NVMM performance simulations proposed in existing works [LKP+14, LY17] presuppose NVMM having similar and extended architectures to DRAM. This section describes overviews of DRAM modules, memory controllers, and DDR protocols. The NVMM behavior models proposed in Section 2.2.2 and Section 2.2.3 are based on these behaviors.

Fig. 2.1 depicts a DRAM-based main memory system. DRAM memory cells consist of several banks, and each bank consists of rows. Data on DRAM are read from or written into by the row-unit. Each bank has one row-buffer that can hold one row. All data are transferred via the row-buffer between memory cells and a memory controller. A memory request from a CPU will be split into several DDR commands [JED12] in a memory controller. Three commands are mainly used: ACTIVATE, READ/WRITE, and PRECHARGE.

- ACTIVATE opens a row and loads data into a row-buffer.
- READ/WRITE reads data from or writes data into the row-buffer.
- PRECHARGE writes the row-buffer into the row, then closes the row.

A row-buffer works as a write-allocate, simple write-back cache. ACTIVATE and PRECHARGE are issued only when a memory request misses a row-buffer. Memory latency depends on row-buffer hit ratio because a row-buffer is much faster than memory cells.

The Coarse-Grain model (Section 2.2.2) and the Fine-Grain model (Section 2.2.3) differ in ways to inject additional latency. The Coarse-Grain model injects additional latency into memory requests on a memory bus, on the other hand, the Fine-Grain model injects additional latency into DDR commands in a memory controller.

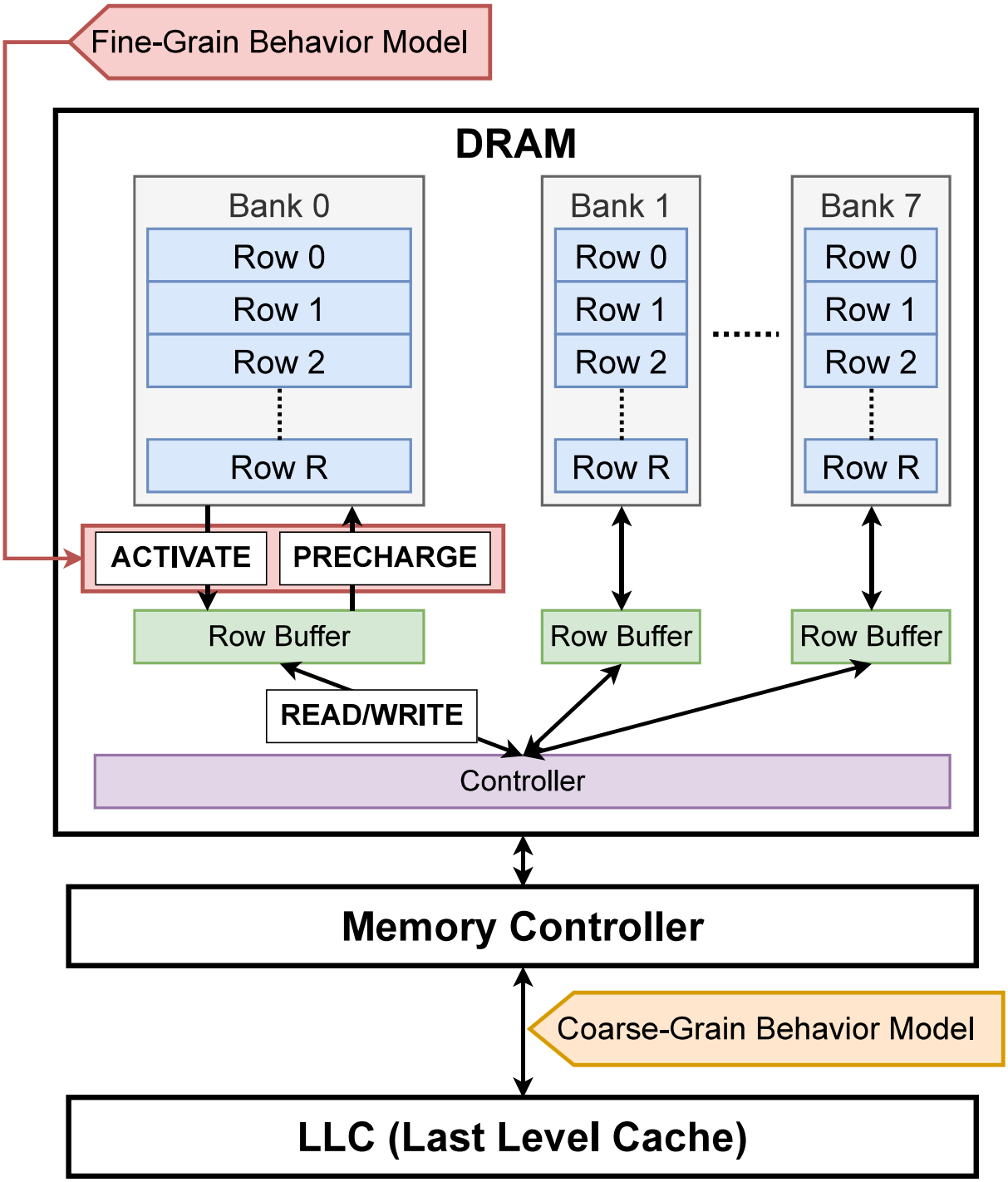


Figure 2.1: Overview of the DRAM-based Main Memory System [OK21]

2.2.2 Coarse-Grain Behavior Model

Most existing works [WW17, VMCL15, LKP+14] inject additional latency into *ALL* memory requests to simulate NVMM performance and latency. They support asymmetric latency by injecting

different latency for read and write requests. The simple latency injection ignores details of memory requests. Whether memory requests can use data on row-buffers, they are always delayed. Therefore, this behavior model can represent an NVMM without any on-module buffer. In comparison to Section 2.2.3, this model coarsely represents NVMM behaviors and architectures. This chapter defines the model as "Coarse-Grain Behaviour Model".

2.2.3 Fine-Grain Behavior Model

The NVMM performance simulation technique proposed in TUNA v2.1 [LY17] injects additional latency into DDR commands in a memory controller. DDR commands and timing parameters are defined as follows:

- READ/WRITE cannot be issued within *tRCD* from the preceding ACTIVATE.
- ACTIVATE cannot be issued within *tRP* from the preceding PRECHARGE.

In other words, data is read from memory cells in *tRCD*, and written back to memory cells in *tRP*. Thus, *tRCD* and *tRP* correspond to read and write latency of memory cells, respectively. By injecting additional latency into these timing parameters, memory accesses are delayed only when missing row-buffers. This behavior model can represent an NVMM with on-module buffers like DRAM. In comparison to Section 2.2.3, this model finely represents NVMM behaviors and architectures. This chapter defines the model as "Fine-Grain Behaviour Model".

This chapter extends the existing Fine-Grain behavior model as follows:

1. Memory cells are accessed only by ACTIVATE and PRECHARGE.
2. A memory controller is extended to manage the dirt of row-buffers. Only dirty row-buffers are written back to memory cells by PRECHARGE.

In addition to long latency, NVMM cells are worn out especially by write operations. All row-buffers must not be always written back into memory cells because NVMM operations are not destructive unlike DRAM. For these reasons, a memory controller of NVMM should be extended to maximize memory lifetime and reduce memory latency.

2.2.4 Overview of Intel Optane DC Persistent Memory

Fig. 2.2 depicts the DCPMM architecture [LX19]. Its memory cells, Optane Media, are based on 3D Xpoint [Web18]. Data on Optane Media is cached on Optane Buffer. Data transfers between Optane Media and Optane Buffer are managed by Optane Controller. Besides, DCPMM has an address translation table (AIT) for wear leveling. Data is transferred between iMC (integrated Memory Controller) and DCPMM in cacheline unit (64-Bytes) like DRAM. In contrast, data is transferred between Optane Controller and Optane Media in a 256-Byte unit. These architectures and behaviors

show different characteristics from DRAM-based memory. This chapter models DCPMM behaviors and architectures as "DCPMM Behavior Model" with some constraints for edge devices.

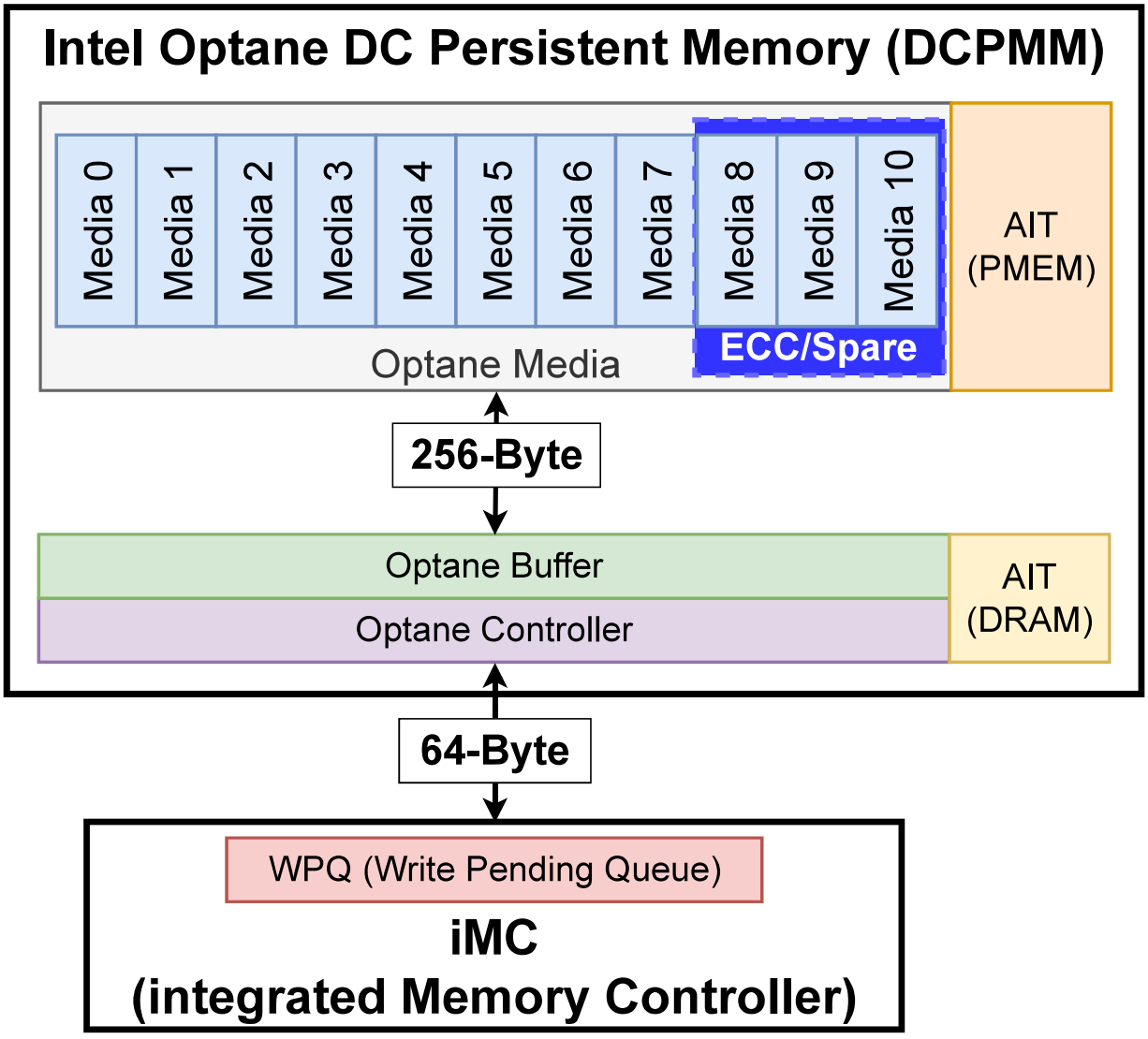


Figure 2.2: Intel Optane DC Persistent Memory Architecture [LX19]

Fig. 2.3 shows the actual DCPMM behavior while changing access strides. It is measured by the micro benchmark shown in Algorithm 1, on the machine shown in Table 2.1. The result is measured under the following conditions to reduce various noises:

- The execution core is bonded by taskset to prevent task migration.
- Hardware prefetchers are disabled to prevent speculative memory access optimization.
- The micro benchmark uses non-temporal instruction [Int22b] to prevent caching.

- The allocated region is filled with zero in advance to prepare page tables. Then, all CPU cachelines are evicted.

Algorithm 1 Pseudocode for Measuring Average Latency on a DCPMM [OK21]

```
1: base := the pointer to the allocated region
2: size := the allocated size to base
3:
4: timer_0 ← current timer
5: addr ← 0
6: while addr < size do
7:     ptr ← base + addr
8:     if Read then
9:         movntdqa(ptr)
10:    else
11:        movntdq(ptr, 0)
12:    end if
13:
14:    if with MFENCE then
15:        _mm_mfence()
16:    end if
17:
18:    addr ← addr + STRIDE
19: end while
20: timer_1 ← current timer
21: average latency = (timer_1 − timer_0)/(size/STRIDE)
```

Table 2.1: DCPMM Configuration [OK21]

CPU	Xeon Gold 5222 @3.80 GHz
DCPMM Module	1st Gen (Apache Pass) 128 GiB
DCPMM Memory Type	AppDirect (not Interleaved)
DCPMM Mode	device DAX (devdax)
Operating System	Ubuntu 18.04LTS

When looking over Fig. 2.3 left to right, latency trends change at three points: 256-Byte, 4-KByte, and 64-KByte. Memory accesses of "with MFENCE" are strictly ordered by memory barriers. They are always issued in the expected (program) order. Asynchronous and/or out-of-order processing in iMC and Optane Controller does not affect the results. Memory accesses of "without MFENCE" are loosely ordered. They may be re-ordered in iMC and/or Optane Controller to maximize throughput by asynchronous and/or out-of-order processing. "Read with MFENCE" (plotted in blue circles) steeply increases to 256-Byte, slowly increases to 4-KByte, then shows constant latency. "Read without MFENCE" (plotted in orange rectangles) slowly increases to 4-

KByte, decreases to 64-KByte, then shows constant latency. "Write with MFENCE" (plotted in gray diamonds) sharply increases to 4-KByte, decreases to 64-KByte, then shows constant latency. "Write without MFENCE" (plotted in green triangles) shows almost the same behavior as "Write with MFENCE" except for the behavior from 64-KByte.

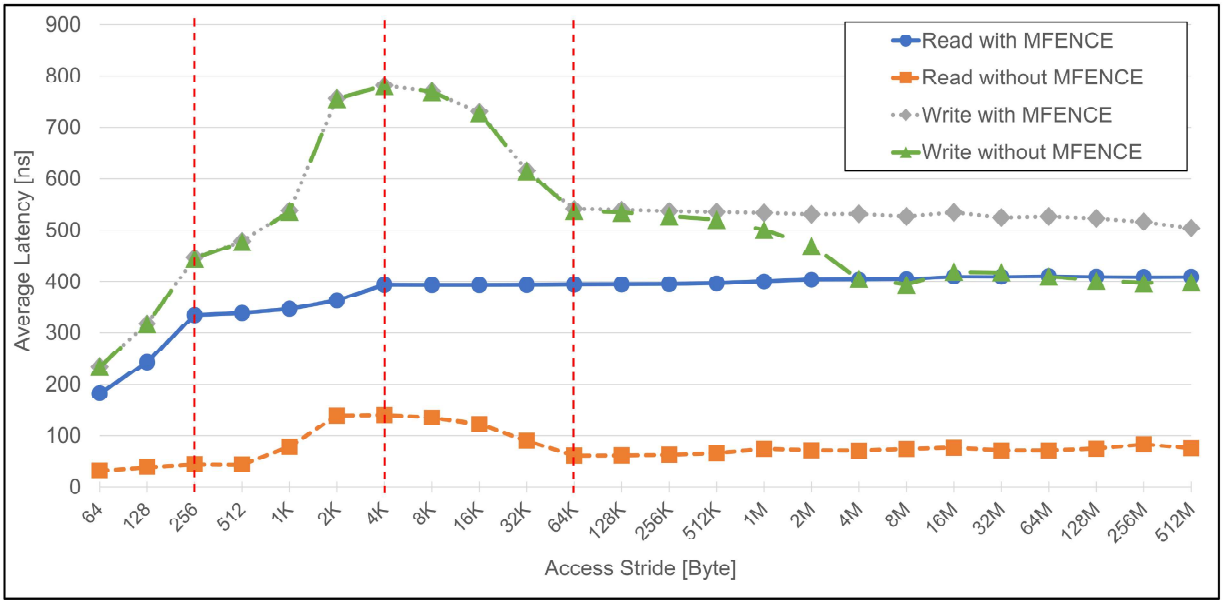


Figure 2.3: Average Latency on a Real DCPMM[OK21]

2.2.5 DCPMM Behavior Model

This dissertation focuses on "with MFENCE" in Fig. 2.3 to model the DCPMM behavior for edge devices considering constraints of power consumption and chip area. The results of "without MFENCE" are caused by advanced controls of rich iMC and on-module controllers. It is not expected on edge devices. Besides, the behavior of "Write with MFENCE" from 4-KByte must be discussed. While detailed DCPMM architecture is unclear, the behavior should derive from the advanced Optane Controller for wear leveling, address indirection, and/or some interleaving. Bank parallelism on 8 chips in Optane Media (Fig. 2.2) should not be the factor. If it is true, latency increases at some points after 4-KByte due to bank conflicts. Therefore, this chapter defines "DCPMM Behavior Model" by modeling the actual DCPMM behavior as follows:

- Read latency steeply increases between 64 and 256-Byte, slowly increases between 256-Byte and 4-KByte, then shows constant latency.
- Write latency steeply increases between 64 to 4-KByte, then shows constant latency.
- The bank parallelism does not exist.

2.3 Details of Simulator Implementation

This chapter proposes the DRAM/NVMM heterogeneous memory simulator. It works on an FPGA, Xilinx Zynq-7000 SoC ZC706 board. The simulator specification is listed in Table 2.2. ZC706 has two parts: PS (Processing System) and PL (Programmable Logic). PS has a dual core ARM Cortex-A9 SoC and peripherals. PL has an FPGA module. Each part has one DIMM. Only the PL DIMM is treated as NVMM. The NVMM region is controlled by a memory controller, Xilinx MIG, implemented on the PL as depicted in Fig. 2.4. This section describes the simulator implementation as follows:

1. Latency injection techniques based on NVMM behavior models
2. Kernel modification to make the NVMM region cacheable
3. Kernel module to call cache flush operations from user space
4. NVMM management library

Table 2.2: The Proposed Simulator Specification [OK21]

FPGA	Xilinx Zynq-7000 SoC ZC706
Device	Zynq-7000 XC7Z045-2FFG900C SoC
CPU Core	Cortex-A9 Dual Core, 667 MHz
L1 Cache	I=32 KiB/core, D=32 KiB/core
L2 Cache	512 KiB/SoC
PS DRAM	1 GiB, DDR3-1066, 16b×2 components
PL DRAM	1 GiB, DDR3-1600, 8b×8, SO-DIMM
PL Frequency	200 MHz
Linux Kernel	GNU/Linux 4.14.0-xilinx-00081-g88cc987 [Xil]
Operating System	Ubuntu 16.04 LTS

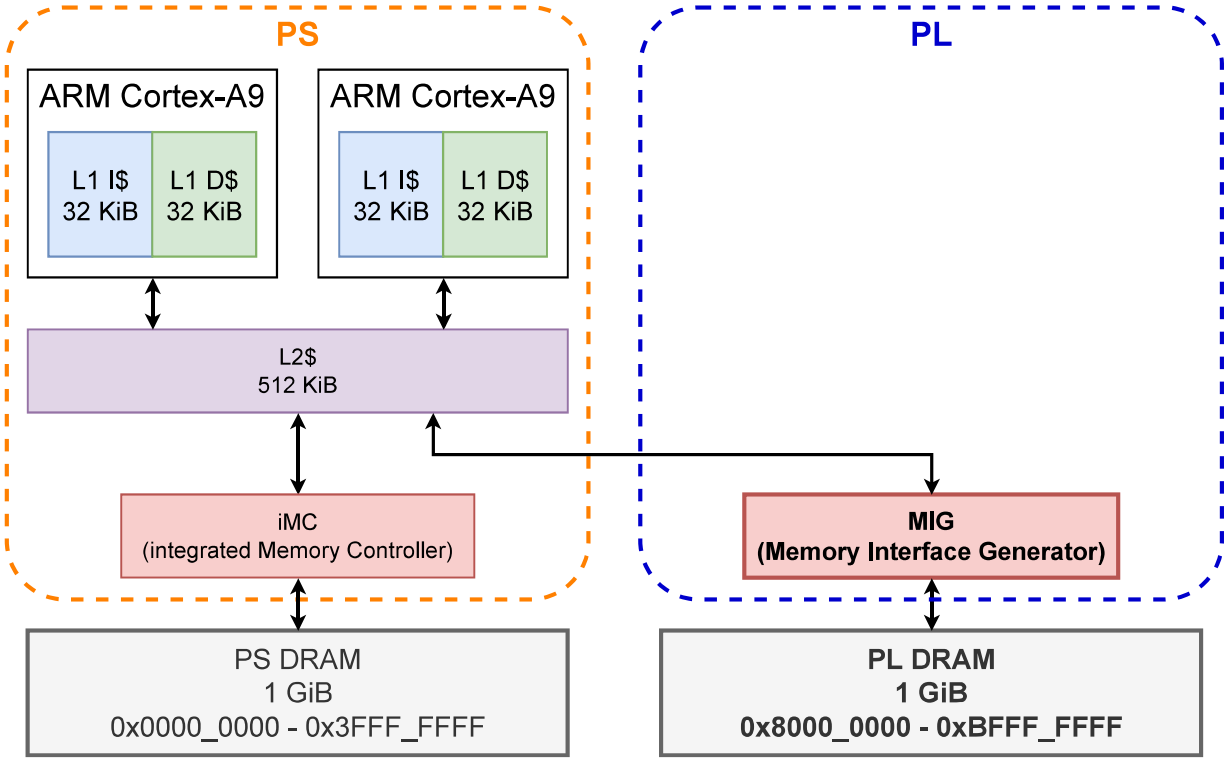


Figure 2.4: Overview of the Proposed Simulator [OK21]

2.3.1 Coarse-Grain Delay Injection

The coarse-grain delay injection is based on the coarse-grain behavior model (Section 2.2.2). Additional latency is injected by an additional module on a memory bus, between the LLC and the MIG (Fig. 2.1). Data is transferred between them using the AXI4 protocol [ARM13]. The Coarse-grain delay injection delays AXI4 protocol handshakes. When the module accepts a memory request from the LLC, the module keeps the request for specified delay clocks, then sends it to the MIG. The module does not interfere with any data communication. Read delay clocks and write delay clocks can be specified independently to represent asymmetric NVMM performance.

2.3.2 Fine-Grain Delay Injection

The fine-grain delay injection is based on the fine-grain behavior model (Section 2.2.3). Additional latency is injected by the modified MIG (Fig. 2.1). The MIG waits for *tRCD* nanoseconds after issuing ACTIVATE, and *tRP* nanoseconds after issuing PRECHARGE. The modified MIG waits for additional cycles after issuing them. The clocks can be specified independently as the same as coarse-grain delay injection. Each bank on DIMMs corresponds to one module in the MIG. The modules work independently, thus, bank parallelism can be explored in the fine-grain delay injection.

2.3.3 DCPMM Delay Injection

The DCPMM delay injection is based on the DCPMM behavior model (Section 2.2.5). The delay injection is the extension of the coarse-grain delay injection. Additional latency is injected into memory requests when crossing 256-Byte and/or 4-KByte boundaries. This technique realizes the behavior depending on memory request intervals (Section 2.2.4). For instance, when a CPU issues two memory requests to 0x8000_0000 and 0x8000_0004, only the former one will be delayed.

2.3.4 Kernel Modification for NVMM Cacheability

The proposed simulator has the DRAM/NVMM heterogeneous memory systems (Table 2.2). The DRAM region and NVMM region must be individually managed to exploit the advantages of the memory system. If the NVMM region is included in system RAM, the region may be unintentionally allocated to processes by a Linux kernel. Its memory allocation may severely degrade system performance. However, the Linux kernel [Xil] provided by Xilinx on the proposed simulator treats only system RAM as cacheable. If the NVMM region is excluded from system RAM, NVMM accesses bypass CPU caches. CPU caches must be applied to NVMM accesses to reduce its long latency.

This dissertation modifies the Linux kernel to make the NVMM region outside system RAM cacheable. A process uses the mmap system call to map arbitrary memory regions into its memory space. On the modified kernel, mapped region cacheability can be specified through the mmap system call. The "O_SYNC" flag on the given file descriptor specifies cacheability. The mapped region becomes cacheable only if the flag is NOT specified. This modification does not interfere with the default behavior. If "O_SYNC" is not specified, memory accesses may be asynchronous.

2.3.5 Kernel Module for User-Space Cache Flush

Data persistency is guaranteed only when data reaches NVMM. When CPU caches are enabled, modified data is written into only CPU caches. Thus, explicit cache eviction is required to guarantee data persistency. While ARM Cortex-A9 on the simulator has cache flush instructions as a part of ARM-v7 ISA [ARM08], they are privileged instructions. A user process cannot call them directly.

This chapter implements the kernel module to expose an API to evict cachelines from user processes. The API takes two arguments: the start virtual address, and the region size. The ARM-v7 cache flush instructions evict one cacheline at a time. If the instructions are directly exposed, multiple context switches between the "User mode" (equivalent to ring-3 in x86) and the "Privileged mode" (equivalent to ring-0 in x86) incur severe overheads. Evicted lines will be written back into NVMM in parallel as much as the hardware allows. Memory barriers are issued before and after the eviction to guarantee data ordering.

2.3.6 NVMM Allocation Library

As described in Section 2.3.4, the NVMM region is excluded from system RAM. The non system RAM region is not managed by the Linux kernel. The region cannot be allocated and deallocated by using dynamic memory allocation functions provided by libc (malloc, calloc, realloc, and free) because libc functions internally borrow a part of system RAM. While the mmap system call can be used, its low-level API has difficulty in allocating arbitrary size regions.

This chapter implements the NVMM management library with libc-compatible APIs. The library provides arbitrary size allocation and deallocation by wrapping mmap/munmap system calls. The APIs are as follows:

```
1  void *NVMM_Malloc(size_t size)
2  void *NVMM_Calloc(size_t nmemb, size_t size)
3  void *NVMM_Realloc(void *ptr, size_t size)
4  void  NVMM_Free(void *ptr)
```

2.4 Validation of NVMM Behavior Models

This section validates three behavior models: Coarse-Grain (Section 2.2.2), Fine-Grain (Section 2.2.3), and DCPMM (Section 2.2.5) by comparing them with golden models. In other words, this section confirms that the models show expected behaviors. The following golden models are used in this chapter:

- Coarse-Grain Behavior Model: The existing NVMM hardware simulator TUNA [LKP+14].
- Fine-Grain Behavior Model: The existing software NVMM simulator gem5 [BBB+11] and NVMain2.0 [PZX15].
- DCPMM Behavior Model: The real DCPMM.

2.4.1 Coarse-Grain Behavior Model

The coarse-grain behavior model and delay injection are the same as the technique proposed in TUNA [LKP+14]. It validates the implementation by confirming that the measured latency follows the expected (configured) latency. This section uses the same validation method.

Algorithm 2 shows the pseudo code of the micro benchmark. The access stride (*STRIDE* in Algorithm 2) is set to 32 or 8192. These two strides should result in different row-buffer hit ratios. The DIMM on the simulator has 8192-Byte row-buffers. Most memory requests hit row-buffers when *STRIDE* is 32. In contrast, all memory requests miss row-buffers when *STRIDE* is 8192. If the coarse-grain delay injection can capture the impact of row-buffer hit ratio, *STRIDE*=32 shows shorter latency than *STRIDE*=8192.

Table 2.3a and Table 2.3b show the measured results. 32 and 8192 are access strides (*STRIDE* in Algorithm 2). "Expected Latency" in the Table is the expected (configured) latency. The latency is injected into only read or write memory requests. "Measured Latency" in the Table is the measured latency. If the coarse-grain delay injection works as expected, the measured latency matches the expected one. First, there are small errors between the expected and the measured latency. These errors should derive from the detailed SoC architecture. The expected latency is injected into memory requests on memory buses. The measured latency observed from CPUs includes overheads in CPU cores, caches, and on-chip buses, in addition to the expected latency. These overheads should be almost constant independent from expected latency. The errors follow this discussion. Second, in Table 2.3a and Table 2.3b, access strides have no impact on measured latency. It indicates that the coarse-grain behavior model cannot capture the impact of row-buffer hit ratio as described in Section 2.2.2.

Through this section, the coarse-grain delay injection was validated in comparison to the TUNA v1's evaluation [LKP+14]. The coarse-grain delay injection shows the following behavior as expected:

Algorithm 2 Pseudocode for Measuring Average Latency [OK21]

```
1: base := the pointer to the allocated region
2: size := the allocated size to base
3:
4: timer_0 ← current timer
5: addr ← 0
6: while addr < size do
7:     ptr ← base + addr
8:     if Read then
9:         v ← *ptr
10:    else
11:        0 → *ptr
12:    end if
13:    addr ← addr + STRIDE
14: end while
15: timer_1 ← current timer
16: average latency = (timer_1 − timer_0)/(size/STRIDE)
```

- ALL memory requests are delayed for the specified clocks.
- No row-buffer hit ratio can be explored.

Table 2.3: Average Latency on the Proposed Simulator (Coarse-Grain Behavior Model) [OK21]

(a) Read Latency

Expected Latency [ns]	Measured Latency [ns]	
	$STRIDE = 32$	$STRIDE = 8192$
200	211	211
400	411	416
600	612	615
800	812	815
1,000	1,012	1,015

(b) Write Latency

Expected Latency [ns]	Measured Latency [ns]	
	$STRIDE = 32$	$STRIDE = 8192$
200	215	217
400	414	422
600	616	623
800	817	817
1,000	1,019	1,023

2.4.2 Fine-Grain Behavior Model

This section validates the fine-grain behavior model and delay injection by comparing it with existing NVMM software simulators: NVMain2 [PZX15] with gem5 [BBB+11]. They are configured as listed in Table 2.4. A CPU model on the gem5 is configured to represent the ARM Cortex-A9 core according to the paper [ECC14]. NVMain2 is modified for the fine-grain behavior model (additional *tRCD* and *tRP*). In addition, this section confirms the effectiveness of the fine-grain behavior model by comparing it with the coarse-grain behavior model. If the fine-grain one works as expected, it can capture the impact of the row-buffer hit ratio unlike the coarse-grain one (Section 2.2.3).

First, the average latency while changing additional latency is measured by using the same micro benchmark as the coarse-grain one (Algorithm 2). The additional latency is injected to *tRCD* or *tRP* in the MIG. Fig. 2.5 and Fig. 2.6a show the results on the proposed simulator, and the software simulators (gem5 + NVMain2), respectively. "32" and "8192" are access strides (*STRIDE* in Algorithm 2). Results in Fig. 2.6a are calibrated based on the raw results shown in Fig. 2.6b. Even if the same benchmark was run, the number of ACTIVATE and PRECHARGE differed. For instance, when the micro benchmark issues 1,048,576 write requests, the number of ACTIVATE/PRECHARGE on the proposed simulator was 2,050,429, in contrast, that on the software simulators was 1,656,898. These differences derive from the detailed memory controller architecture

Table 2.4: gem5 and NVMain2 Configurations [OK21]

gem5	
Simulation Mode	Syscall Emulation
CPU Frequency	667 MHz
CPU Core	O3_ARM_v7a_3 ×1
Cache Line Size	32 Byte
L1 Cache	I=32 KiB / D=32 KiB
L2 Cache	512 KiB

NVMain2	
Frequency	166 MHz
Size	1 GiB
Command Queue	READ=32 entries, WRITE=32 entries
Page Policy	Relax Page

including command queues, arbiters, internal buses, and so on. To remove these differences, average latency on Fig. 2.6b is calibrated against the number of ACTIVATE/PRECHARGE.

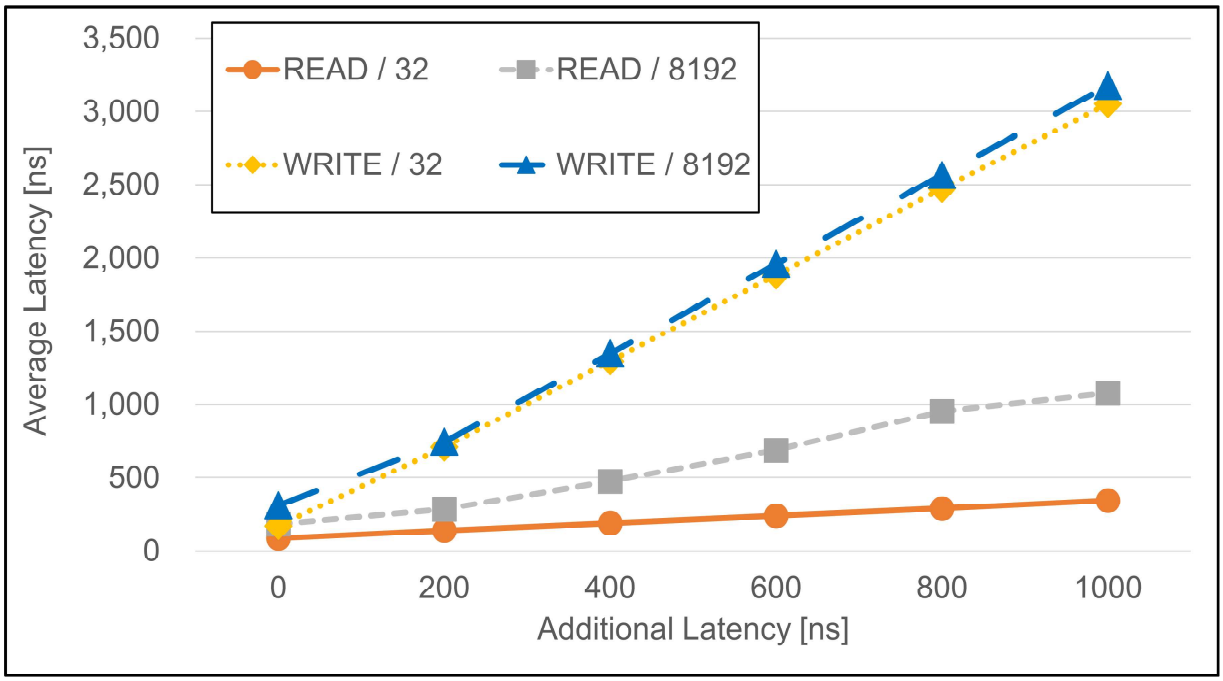
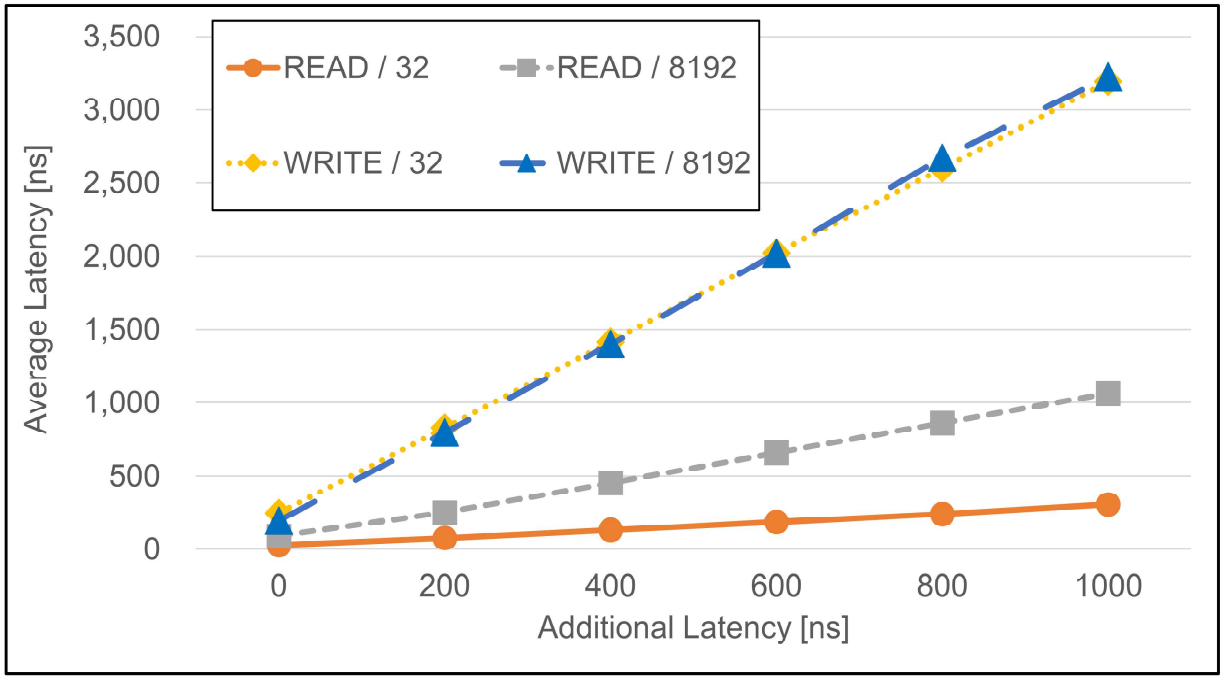
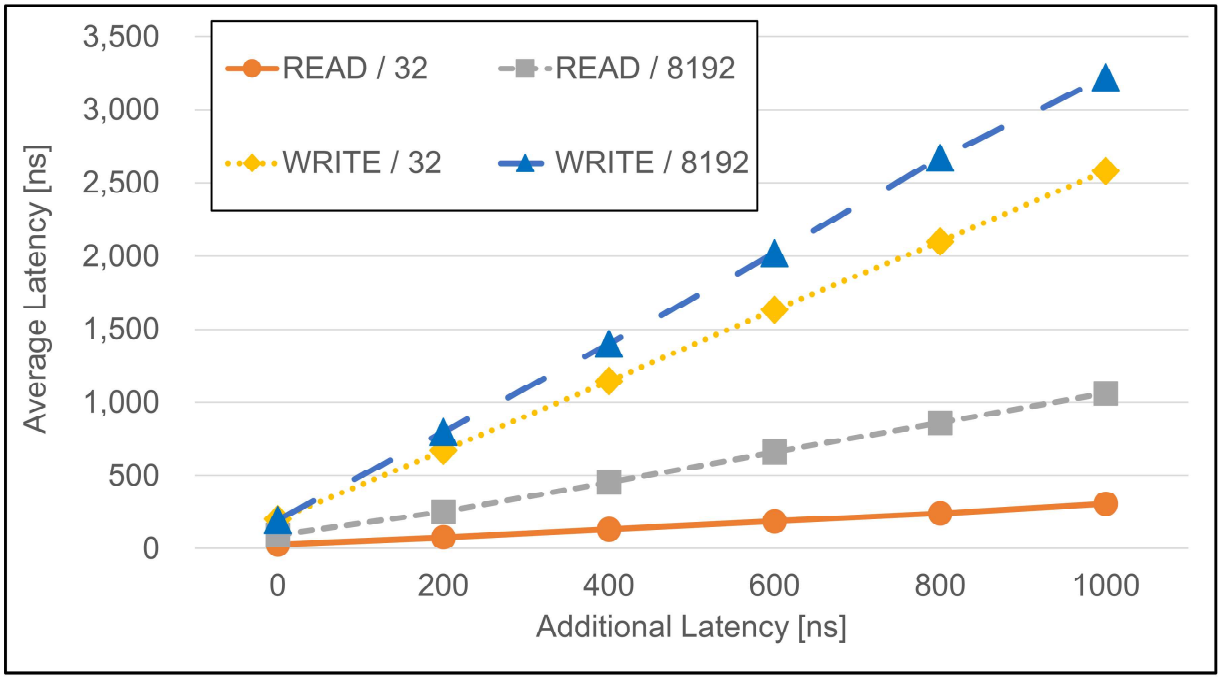


Figure 2.5: Average Latency while Changing Additional Latency (Fine-Grain on the Proposed Simulator) [OK21]

These graphs show four important results. First, the two graphs show almost the same behaviors. It confirms that the fine-grain behavior model works as expected in comparison to existing golden models. Second, the impact of additional latency depends on memory requests and access strides.



(a) Calibrated Average Latency



(b) Raw Average Latency

Figure 2.6: Average Latency while Changing Additional Latency (Fine-Grain on the Software Simulators) [OK21]

"READ/32" is slightly affected by additional latency, on the other hand, "READ/8192" is heavily affected. The read benchmark with 32-Byte stride should frequently hit row-buffers. In this case, less ACTIVATE/PRECHARGE are issued, and less total additional latency is injected. In contrast, the benchmark with 8192-Byte stride always misses row-buffer, then ACTIVATE/PRECHARGE are always issued. This behavior confirms that the fine-grain delay injection can capture the impact of row-buffer hit ratio. Third, "WRITE" results show about 3,000 nanoseconds. When CPU caches are enabled, cachelines are always fetched before writes, then data is written only into CPU caches and written back into a memory module later by line replacements. These behaviors require 1 memory read request for fetching, and 1 memory write request for eviction. A memory controller receives them in random order due to random cacheline replacements. It lowers row-buffer hit ratio, then ACTIVATE and PRECHARGE are issued for most memory requests. Additional latency should be injected four times (4,000 nanoseconds) in total, however, the results show about 3,000 nanoseconds. It confirms that the row-buffers state management works as expected (Section 2.2.3). Memory read requests for fetching do not modify row-buffers, thus, additional latency for PRECHARGE is not injected. Fourth, the write results on Fig. 2.5 and Fig. 2.6a show almost the same behavior for different access strides. It is due to CPU caches. Access locality is mostly absorbed in CPU caches. Memory write requests are almost randomly issued to a memory controller. This behavior is allowable because such a pure write benchmark is not suitable for NVMM.

Second, the average latency while changing the number of banks accessed in parallel is measured. Algorithm 3 shows the pseudo code of the micro benchmark. The number of banks accessed in parallel (*NBANK* in Algorithm 3) is set to 1, 2, 3, or 4. If the behavior model can capture bank parallelism, the average latency decreases as increasing *NBANK*. Fig. 2.7a and Fig. 2.7b show the normalized average read/write latency, respectively. "Coarse" are the result of the coarse-grain behavior model on the proposed simulator. "Fine(Proposed)" and "Fine(gem5+NVMain2)" are the results of the fine-grain behavior model on the proposed simulator and the software simulators, respectively. For "Coarse", additional 1,000 nanoseconds are injected into read or write requests. For "Fine", additional 1,000 nanoseconds are injected into *tRCD* or *tRP*. All results are normalized against the result of *NBANK=1*. Comparison between "Coarse" and "Fine(Proposed)" reveals that only the latter is affected by *NBANK*. The "Fine(Proposed)" read and write latency decrease to 60% and 50% respectively, on the other hand, the "Coarse" latency shows constant. Besides, "Fine(Proposed)" and "Fine(gem5+NVMain2)" show almost the same behaviors. While there are some errors between them, the errors are allowable because they derive from the different detailed internal architecture as described above.

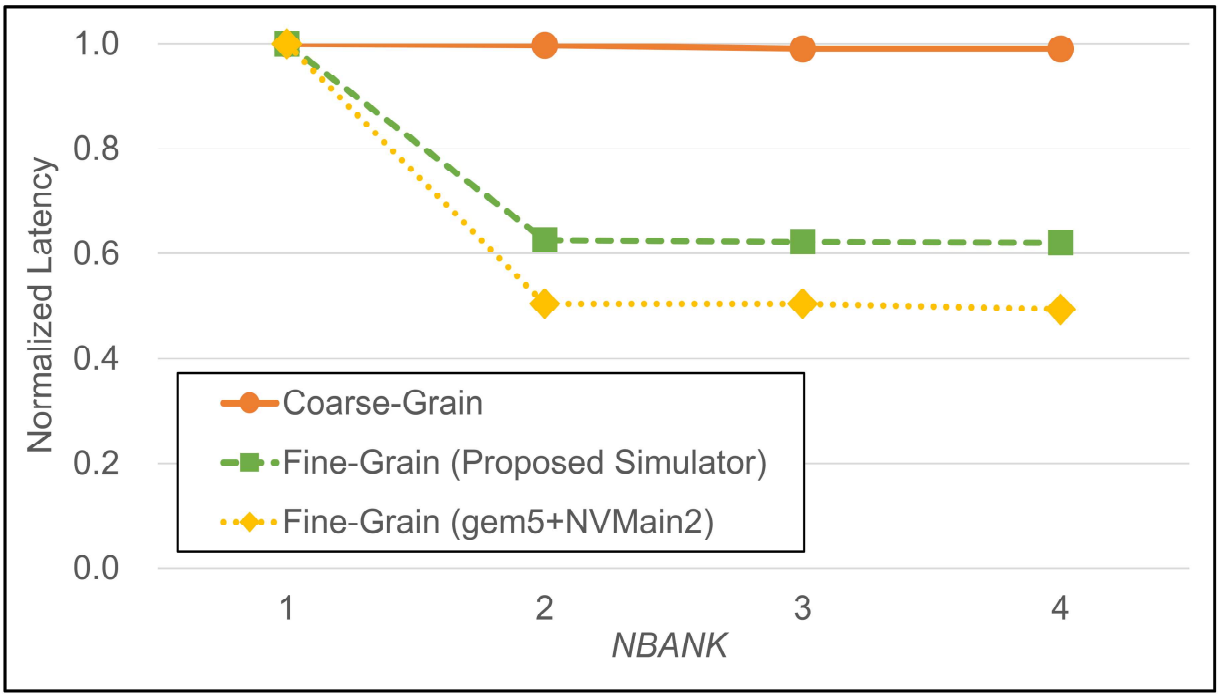
Through this section, the fine-grain delay injection was validated in comparison to the existing simulators, gem5 and NVMain2. The fine-grain behavior model shows the following behaviors as expected:

- Only memory accesses missing row-buffers are delayed for the specified cycles.

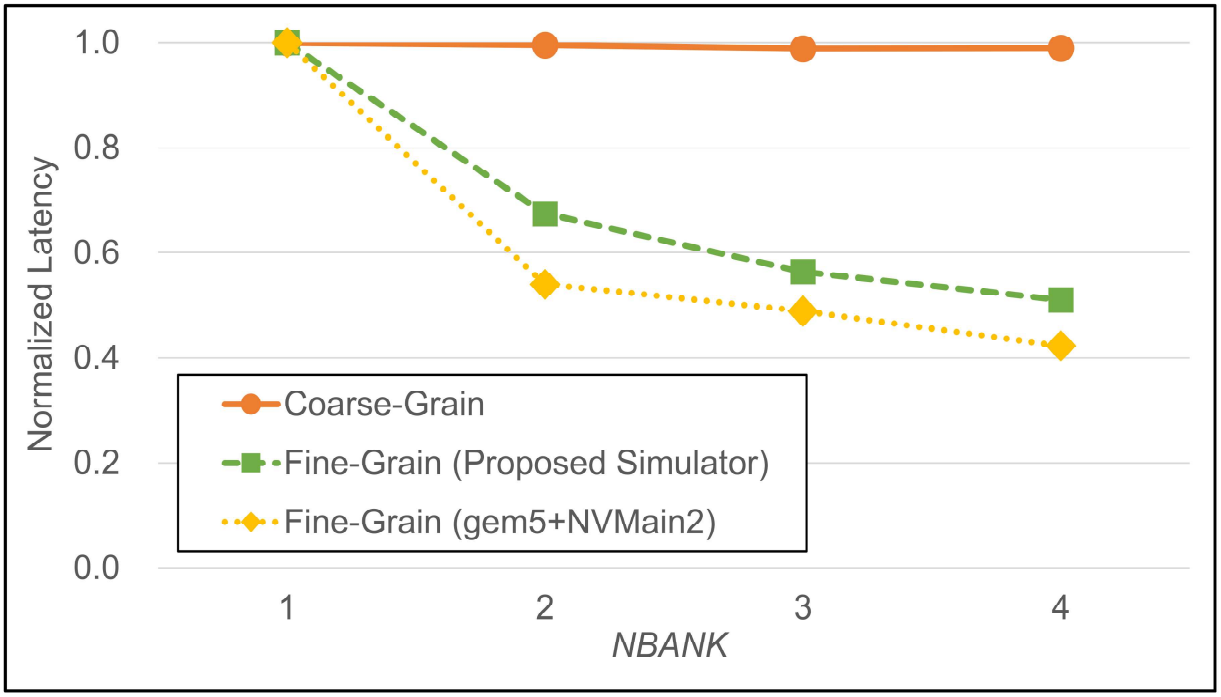
- Row-buffer hit ratio and bank parallelism can be explored on the fine-grain behavior model unlike the coarse-grain behavior model.
- The fine-grain behavior model can show the same behaviors as the existing NVMM simulators. The proposed simulator can provide reliable evaluations as the existing simulators.

Algorithm 3 Pseudocode for Measuring Bank Parallelism [OK21]

```
1: // mem is an array of ROWs
2: // mem has BANK_PER_MEM banks, and each bank has ROW_PER_BANK rows
3: mem := ROW[BANK_PER_MEM][ROW_PER_BANK]
4:
5: timer_0 ← the current timer
6: // iterate each row, each bank
7: for row = 0 ... ROW_PER_BANK do
8:     for bank = 0 ... NBANK do
9:         if Read then
10:            v ← mem[bank][row]
11:        else
12:            0 → mem[bank][row]
13:        end if
14:    end for
15: end for
16: timer_1 ← the current timer
17: average latency = (timer_1 − timer_0)/(ROW_PER_BANK × NBANK)
```



(a) Read Latency



(b) Write Latency

Figure 2.7: Normalized Latency while Changing *NBANK* [OK21]

2.4.3 DCPMM Behavior Model

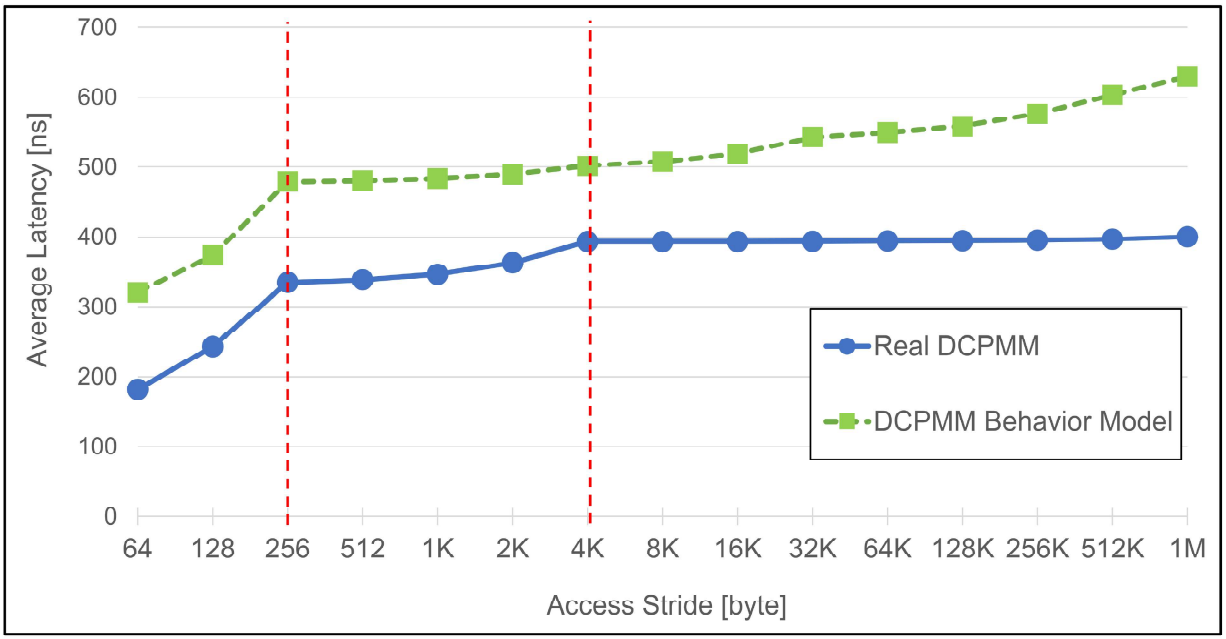
This section validates the DCPMM behavior model and delay injection by comparing it with a real DCPMM (Table 2.1). The latency trends are measured using the same benchmark shown in Algorithm 1 with some adjustments. *STRIDE* is set up to 1-MiB to get enough results on the smaller NVMM region on the simulator than a real DCPMM. CPU caches were disabled instead of non-temporal instructions used in Section 2.2.4.

Fig. 2.8a and Fig. 2.8a show the results. "Real DCPMM" (plotted in blue circles) shows the result of a real DCPMM ("with MFENCE" on Fig. 2.3). "DCPMM Behavior Model" (plotted in green squares) shows the result of the DCPMM behavior model. For "DCPMM Behavior Model" results, read latency for 256-Byte and 4-KByte are 200ns and 225ns, and Write latency for 256-Byte and 4-KByte are 500ns and 800ns, respectively. In Fig. 2.8a, a real DCPMM and the proposed simulator show almost the same read latency trends between 64-Byte and 4-KByte. However, they differ above 4-KByte. This error is allowable on the DCPMM behavior model on the simulator (Section 2.2.5). The used micro benchmark issues continuous heavy read requests. Such memory requests with wide strides (low access locality) are not desirable and should be avoided on NVMM, thus the erroneous situation in Fig. 2.8a rarely happens.

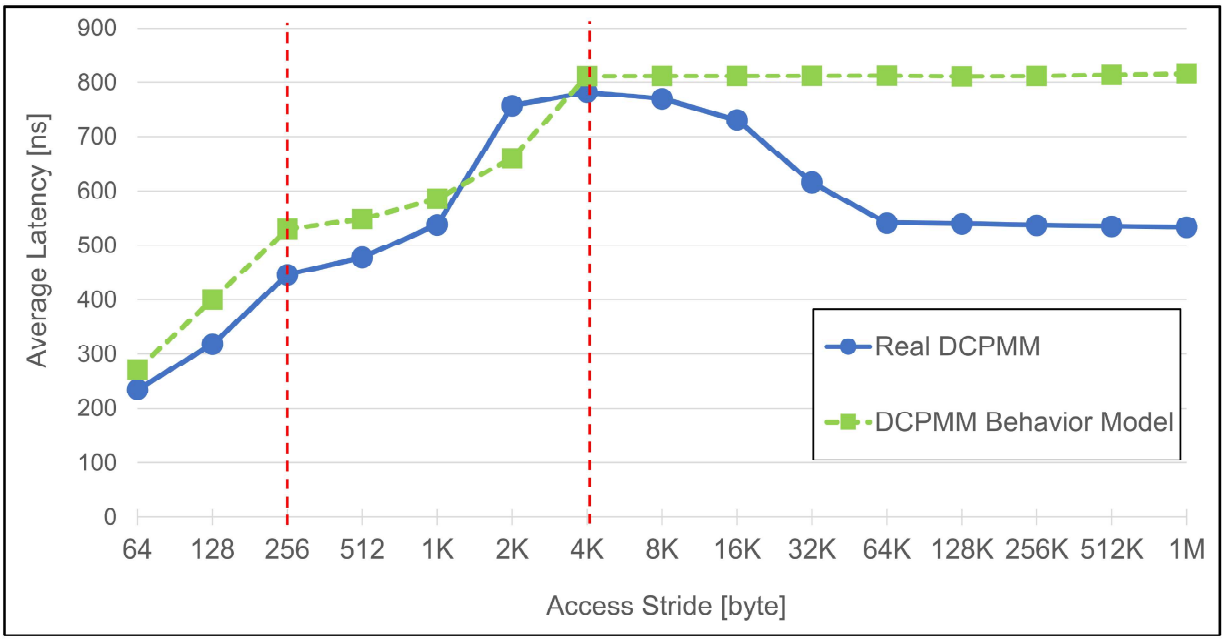
In Fig. 2.8b, a real DCPMM and the proposed simulator show almost the same write latency trends between 64-Byte and 4-KByte. The behavior above 4-KByte is out of scope in the DCPMM behavior model (Section 2.2.5). Such advanced controllers are not expected on edge devices for some limitations and costs.

Through this section, the DCPMM behavior model was validated in comparison to a real DCPMM. The DCPMM behavior model shows the following behavior:

- Read latency steeply increases between 64 and 256-Byte, slowly increases between 256-Byte and 4-KByte, then shows constant latency.
- Write latency steeply increases between 64 to 4-KByte, then shows constant latency.
- The bank parallelism does not exist since the DCPMM behavior model is an extension of the coarse-grain behavior model.



(a) Read Latency



(b) Write Latency

Figure 2.8: Average Latency while Changing *STRIDE* [OK21]

2.5 Experimental Evaluation of NVMM Behavior Models with SPEC CPU Benchmarks

This section explores essential factors for the DRAM/NVMM heterogeneous memory system with SPEC 2017 CPU benchmarks [Sta]. Three NVMM delay injections are configured as follows:

- Coarse-Grain: 1,000ns for both read and write requests
- Fine-Grain: 1,000ns for both *tRCD* and *tRP*
- DCPMM: 1,000ns for base latency, additional 2,000ns/2,500ns for reads crossing 256-Byte/4-KByte, additional 5,000ns/8,000ns for writes crossing 256-Byte/4-KByte, respectively

On the DCPMM behavior model, additional latency is calculated as follows: Section 2.2.4 and Intel [Int22a] indicate that read requests to DCPMM become twice slower when missing Optane Buffer. In other words, a read request accesses a different Optane line (256-Byte). Thus, 2,000ns are injected for 256-Byte boundaries. The latency for other boundaries is defined according to Fig. 2.3.

2.5.1 Normalized Execution Time

This section shows how three NVMM behavior models affect real application performance, then explores factors for optimizations on DRAM/NVMM heterogeneous memory systems. Programs with various characteristics from SPEC 2017 CPU benchmark [Sta] are used. Fourteen of 24 benchmarks are chosen from CPU rate benchmark programs. They can be compiled and executed on the proposed simulator. All memory allocations in the programs are replaced with the NVMM management library (Section 2.3.6) to allocate heap objects on the NVMM region.

Fig. 2.9 shows the results. The horizontal axis shows the normalized execution time on the simulated NVMM against that on DRAM (no additional latency). The bars filled with blue dots, green diagonal lines, and orange cross stripes are the results of coarse-grain, fine-grain, and DCPMM behavior models. The bars are sorted in the ascending order of "Fine-Grain" from left to right. In the graph, "Coarse-Grain" and "DCPMM" show the same trends since the latter is an extension of the former one. Throughout this section, "Coarse-Grain" is discussed on behalf of "Coarse-Grain" and "DCPMM".

Fig. 2.9 shows that behavior models differently affect each program execution time. For instance, 544.nab_r and 511.povray_r are not almost affected by different behavior models. They show almost 1.0 for both the coarse-grain and fine-grain behavior models. However, for 519.lbm_r, the normalized execution time of the coarse-grain model is 8.3 while that of the fine-grain model is 13.4. The fine-grain model heavily affects execution time 1.61 times more than the coarse-grain one. Besides, in the graph, the fine-grain model generally shows longer normalized execution time than coarse-grain. Four applications do not follow the trend: 531.deepsjeng_r, 520.omnetpp_r, 505.mcf_r, and

510.parest_r. The coarse-grain model on them shows higher execution time than the fine-grain model.

For detailed investigations, two application characteristics are measured: access locality and bank parallelism. Access locality is the ratio of ACTIVATE to memory requests. It is calculated as "(1 - (the number of ACTIVATE/the number of requests))". The number of ACTIVATE is counted in the MIG, and the number of requests is counted on a memory bus (between the LLC and the MIG). High access locality indicates that more memory requests hit row-buffers. Bank parallelism is the ratio of memory requests accessing the different banks from the preceding request. Higher bank parallelism indicates that more memory requests are processed in parallel.

Table 2.5 shows access locality and bank parallelism. The exceptions shown in the above discussion are underlined. The programs are similarly sorted as Fig. 2.9 from top to bottom. In the 4 underlined exceptions, 531.deepsjeng_r, 520.omnetpp_r, and 505.mcf_r show high access locality (0.367, 0.118, and 0.191), and 520.omnetpp_r and 505.mcf_r show high bank parallelism (0.270, 0.280). The fine-grain model can capture these characteristics as confirmed in Section 2.4.2. The results show the importance of access locality and bank parallelism in optimization for NVMM, besides, prove the fine-grain delay injection.

However, these two factors cannot explain the behavior of 510.parest_r. Its access locality and bank parallelism are not good unlike others. The behavior is due to the memory requests' read/write ratio. In the coarse-grain delay injection, while the read and write memory bus work in parallel, each bus can keep only one request at a time. An application having an excessive read/write ratio is heavily affected by NVMM latency. 510.parest_r shows a high read/write ratio (25.0). The significantly high read/write ratio for the coarse-grain behavior model spoils the parallelism of memory accesses, resulting in a longer execution time than expected.

The above discussion shows the effectiveness of the fine-grain behavior model. An important question exists yet: Which of the program characteristics mainly affects the execution time? In other words, which factors mainly determine the order of programs in Fig. 2.9 from left to right? Table 2.5 shows that 505.mcf_r has higher access locality (0.191) and bank parallelism (0.280) than 538.imagick_r, however, Fig. 2.9 shows that the latter one is more affected by NVMM latency. To investigate this question, cache hit ratio for the LLC and the memory access frequency to the NVMM are also measured. They are measured without any additional latency. The memory access frequency is measured on a memory bus (between the LLC and the MIG).

Table 2.6 shows the measurement results for each program. The programs are similarly sorted as Fig. 2.9 from top to bottom. In other words, the normalized execution time of the fine-grain model gets longer from top to bottom. Longer execution time derives from a lower cache hit ratio and higher memory request frequency except for some underlined benchmarks. Considering Table 2.5, the impact of memory access frequency on them is reduced by access locality and bank parallelism.

Regarding the relationship between 505.mcf_r and 538.imagick_r, the former accesses NVMM

twice as often as the latter one. Although the impact of NVMM latency on 505.mcf_r is reduced by its characteristics (Table 2.5), it is more heavily affected than 538.imagick_r. This result implies that the impact of the memory access frequency exceeds the reduction by access locality and bank parallelism. The same situation is found in 519.lbm_r and 510.parest_r.

This section reveals that the impact of NVMM latency are affected by the following factors:

- Memory access frequency increases the impact of NVMM latency on program execution time.
- High access locality reduces the impact by effectively using on-module buffers.
- High bank parallelism reduces the impact by processing memory requests in parallel.
- Quite high memory access frequency exceeds the reduction by access locality and bank parallelism.

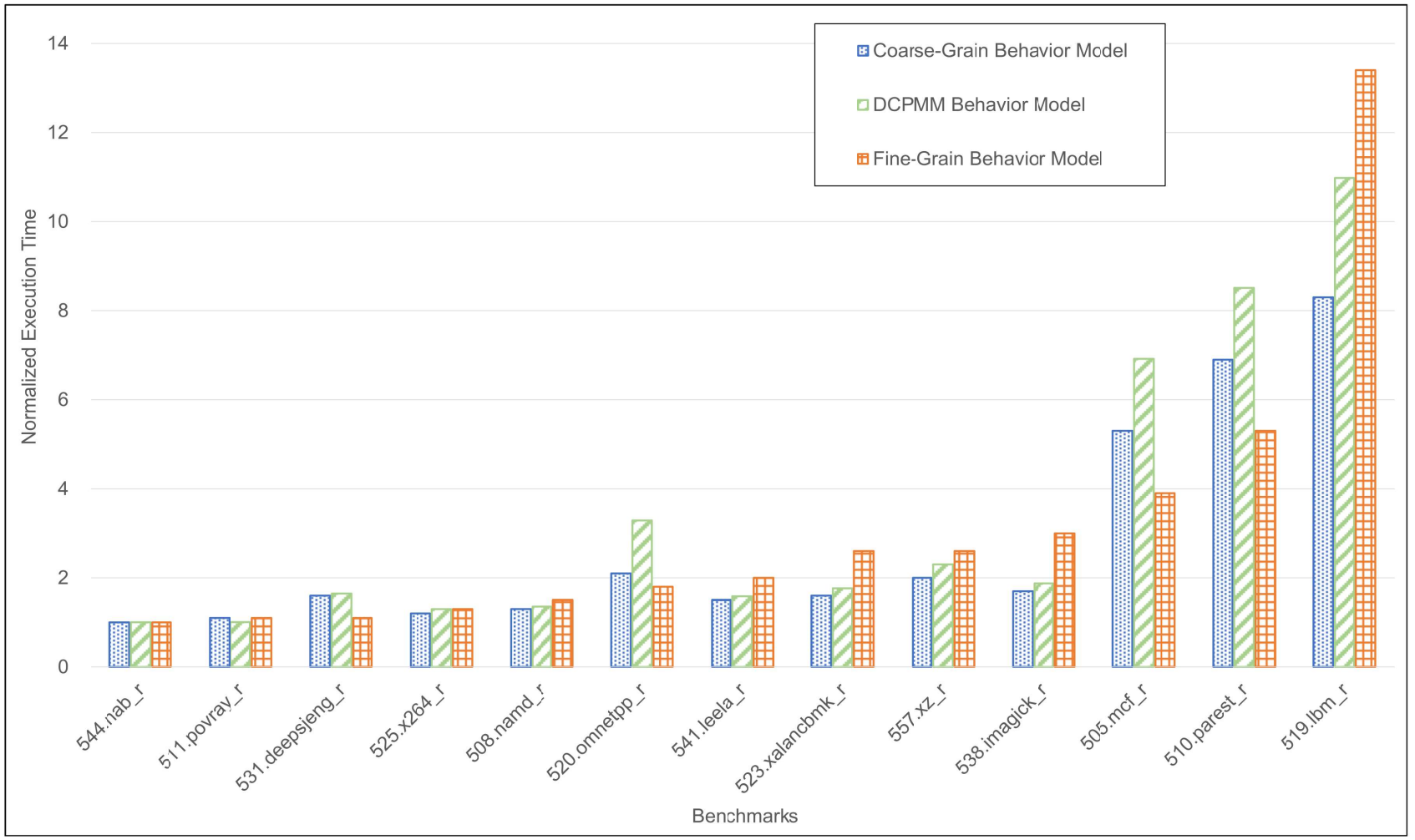


Figure 2.9: Normalized Execution Time of SPEC CPU 2017 Programs [OK21]

Table 2.5: Access Locality and Bank Parallelism [OK21]

Benchmark	Access Locality	Bank Parallelism
544.nab_r	0.011	0.000
511.povray_r	0.156	0.000
531.deepsjeng_r	0.367	0.170
525.x264_r	0.036	0.070
508.namd_r	0.055	0.080
520.omnetpp_r	0.118	0.270
541.leela_r	0.087	0.000
557.xz_r	0.079	0.050
523.xalancbmk_r	0.030	0.000
538.imagick_r	0.013	0.000
505.mcf_r	0.191	0.280
510.parest_r	0.064	0.001
519.lbm_r	0.066	0.220

Table 2.6: Cache Hit Ratio and Memory Access Frequency[OK21]

Benchmark	Cache Hit Ratio [%]	Memory Access Frequency [/s]
544.nab_r	99.998	2,615
511.povray_r	99.983	85,219
531.deepsjeng_r	99.784	623,954
525.x264_r	99.926	493,471
508.namd_r	99.858	669,040
520.omnetpp_r	97.968	1,561,328
541.leela_r	99.785	852,818
557.xz_r	99.596	1,824,788
523.xalancbmk_r	99.516	1,295,606
538.imagick_r	99.356	1,540,642
505.mcf_r	93.501	4,170,876
510.parest_r	95.384	5,967,728
519.lbm_r	88.551	11,812,742

2.5.2 Cache Flush Overhead

As described in Section 2.3.5, cachelines must be explicitly evicted to guarantee data persistency on NVMM. This section measures cache flush overheads and what factors affect the overheads. Four applications having different characteristics are used. 508.namd_r has high data parallelism. 541.leela_r allocates a lot of small regions (20 Byte × 200,000). 557.xz_r is an in-memory application using a few large regions. 519.lbm_r requires quite a high bandwidth. Cache flush instructions are inserted into their computation kernels to make their main data structure durable. Table 2.7 shows the overhead caused by the cache flush. The overheads of "zero" denotes the additional execution

time caused by the cache flush operations with no additional latency. Similarly, the overheads of "coarse" and "fine" are the additional execution time when additional latency is injected by using their models. "Total Flushed Lines" is the number of total cachelines flushed by the inserted flush instructions.

Table 2.7: Cache Flush Overhead and Flushed Lines [OK21]

	Cache Flush Overhead [s]			
Benchmark	zero	coarse	fine	Total Flushed Lines
508.namd_r	0.31	0.33	0.27	922,288
541.leela_r	0.30	0.35	0.28	248,525
557.xz_r	0.03	0.04	0.02	166,898
519.lbm_r	5.49	5.55	5.46	1,859,045

This table shows that "fine" is less than "coarse" and "coarse" is more than "zero". The former behavior is due to high data locality. Memory requests caused by cache flush operations often access nearby addresses because of spatial locality on CPU caches. The cache flush overhead is reduced by high access locality. The latter observation shows that overhead is affected by additional latency.

Regarding the amount of the overhead for each program, Table 2.7 indicates that the number of total flushed lines affects the overhead. However, 508.namd_r flushes about four times more lines than 541.leela_r and shows almost the same overhead. This behavior is due to the cache flush operation frequency. 508.name_r specifies a large array at a time, therefore, most cachelines have been already evicted from the cache by line replacement, resulting in the small number of NVMM access. On the other hand, 541.leela_r specifies small nodes many times, thus, most cachelines exist on CPU caches and are evicted by flush operation. These cases indicate that the overhead caused by the explicit data eviction is affected by the cache flush granularity. However, it must be noticed that data durability cannot be ensured until the end of a cache flush operation and the following memory barrier operation. There is a tradeoff between cache flush operation frequency and additional execution overheads.

2.6 Conclusion

This chapter proposed the hardware DRAM/NVMM heterogeneous memory simulator on the Xilinx ZC706 board that has two parts: an ARM-based SoC and an FPGA module. The simulator provides NVMM simulations based on three NVMM behavior models: Coarse-Grain, Fine-Grain, and DCPMM. The coarse-grain one represents an NVMM without any on-module caches. The fine-grain one represents an NVMM with a similar architecture to DRAM. Its memory cells consisting of banks, rows, and columns are accessed via on-module write-back caches. The DCPMM behavior model represents a real DCPMM. NVMM simulation was implemented on the FPGA module. The proposed simulator provides the whole evaluation platform for the DRAM/NVMM heterogeneous memory systems. Ubuntu runs on the ARM-based SoC. It has features for the DRAM/NVMM heterogeneous memory systems: the modified Linux kernel for cacheable NVMM region, the kernel module for user-space cache eviction, and the NVMM management library. The proposed simulator provides a fast, reliable, and useful way to explore optimizations of the DRAM/NVMM heterogeneous memory systems.

All NVMM behavior models were validated against golden models: the existing NVMM hardware simulator TUNA for the Coarse-Grain, the existing software simulators gem5+NVMain2 for the Fine-Grain, and a real DCPMM for DCPMM. The validation showed that NVMM simulations on the simulator work as expected. It also showed that only the fine-grain can capture the impact of access locality and bank parallelism. Then, the experimental evaluation using SPEC CPU 2017 benchmarks revealed three essential factors in optimization for NVMM: memory access frequency, access locality, and bank parallelism. The last two factors can alleviate the first factor's impact; however, frequent memory accesses spoil them and severely degrade system performance.

Chapter 3

DRAM/NVMM Heterogeneous Memory Simulator on Soft Processor Systems *

*This chapter is based on the paper, "Open-Source RISC-V Linux-Compatible NVMM Emulator", Sixth Workshop on Computer Architecture Research with RISC-V (CARRV 2022), Jun. 2022.

3.1 Preface

Soft processors on SoCs have become important in IoT and reconfigurable computing. The fully open-source ISA, RISC-V also has accelerated the trend. Many RISC-V cores and SoCs have been proposed since its first proposal in 2011. They are expected to encompass various requirements on edge devices as an open-source, free, customizable ISA. DRAM/NVMM heterogeneous memory systems may be used on RISC-V soft processor systems in the future. However, the fine-grain behavior model (Section 2.2.3) presupposes that a CPU runs sufficiently faster than its memory system. Soft processor systems do not always satisfy the assumption unlike ARM-based hard processor systems.

This chapter proposes the DRAM/NVMM heterogeneous memory simulator employing RISC-V soft processor systems on an FPGA. The existing NVMM behavior models are extended for soft processor systems. The simulator also provides a Linux-based evaluation platform. The extended fine-grain behavior model is validated against the existing behavior models. Then, optimization techniques for the heterogeneous memory system are explored using benchmark programs like Chapter 3.

This chapter is organized as follows: Section 3.2 discusses the existing NVMM behavior models on soft processor systems, then introduces "Extended Fine-Grain Behavior Model". Section 3.3 describes the whole simulator implementation: NVMM performance simulations, logical memory partitioning, and the modified RISC-V core for user-space cache evictions. Section 3.4 and Section 3.5 validate the NVMM behavior models and show the effectiveness of the extended fine-grain behavior model on soft processor systems. Section 3.6 concludes this chapter.

3.2 NVMM Behavior Model

This section describes the existing three NVMM behavior models (Coarse-Grain, Fine-Grain, and DCPMM) on soft processor systems, then discusses and proposes the new NVMM behavior model.

3.2.1 Existing NVMM Behavior Models on Soft Processor Systems

Chapter 2 proposed two NVMM behavior models based on the traditional DRAM behavior and architecture: Coarse-Grain and Fine-Grain. Only the coarse-grain based injection can be directly applied to soft processor systems.

The coarse-grain behavior model injects additional latency into memory requests on a memory bus. Memory requests are kept for specified clocks in the additional delay module. In this case, the behavior of delay injections is not affected by the operating frequencies of CPUs and the memory controller. Regardless of their frequency, CPUs always observe constant memory stalls for specified additional clocks.

On the other hand, the fine-grain behavior model does not work as expected on slow, soft processor systems. As described in Section 2.2.3, a high row-buffer hit ratio shows less ACTIVATE and PRECHARGE. Row-buffers can be re-used only when two successive memory requests hit the same, opened row-buffer. However, row-buffers on DRAM are implicitly closed and written back to memory cells to strictly stabilize DRAM memory cells. The interval from row-buffer opening to closing is small. Soft processors cannot issue multiple memory requests in the interval, then, additional latency is injected into ALL memory requests. The behavior is the same as the coarse-grain behavior model. Therefore, the existing fine-grain behavior model must be extended to explore optimization techniques for the DRAM/NVMM heterogeneous memory system on soft processor systems.

3.2.2 Extended Fine-Grain Behavior Model

Fig. 3.1 depicts the DDR3 timing parameters. A memory controller issues an implicit PRECHARGE after *tRTP* or *tRP* from the preceding READ or WRITE, and/or *tRAS* from the preceding ACTIVATE. Slow processors cannot issue successive memory requests within these timing parameters. To capture the impact of row-buffer hit ratio, row-buffers must be kept open long enough. The "Extended Fine-Grain Behavior Model" also adjusts these timing parameters to prevent an implicit PRECHARGE. By setting *tRAS* enough longer than the CPU memory request frequency, they can hit the same row-buffer. Fig. 3.2 depicts the comparison between the fine-grain delay behavior model and the extended one. On the original fine-grain, the PRECHARGE for the read request-1 is issued before the read request-2, then ACTIVATE is required for request-2. In contrast, on the extended fine-grain, the PRECHARGE is postponed, then ACTIVATE for the request-2 is omitted.

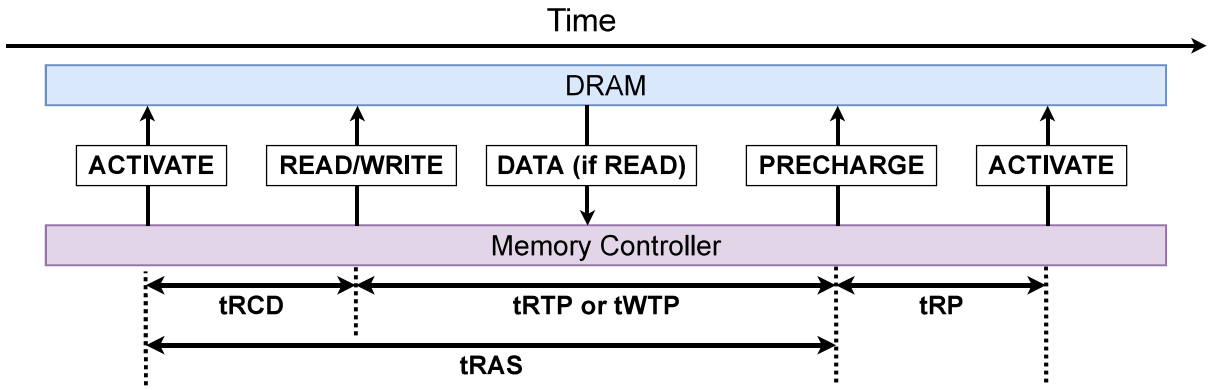


Figure 3.1: DDR3 Timing Parameters [OK22b]

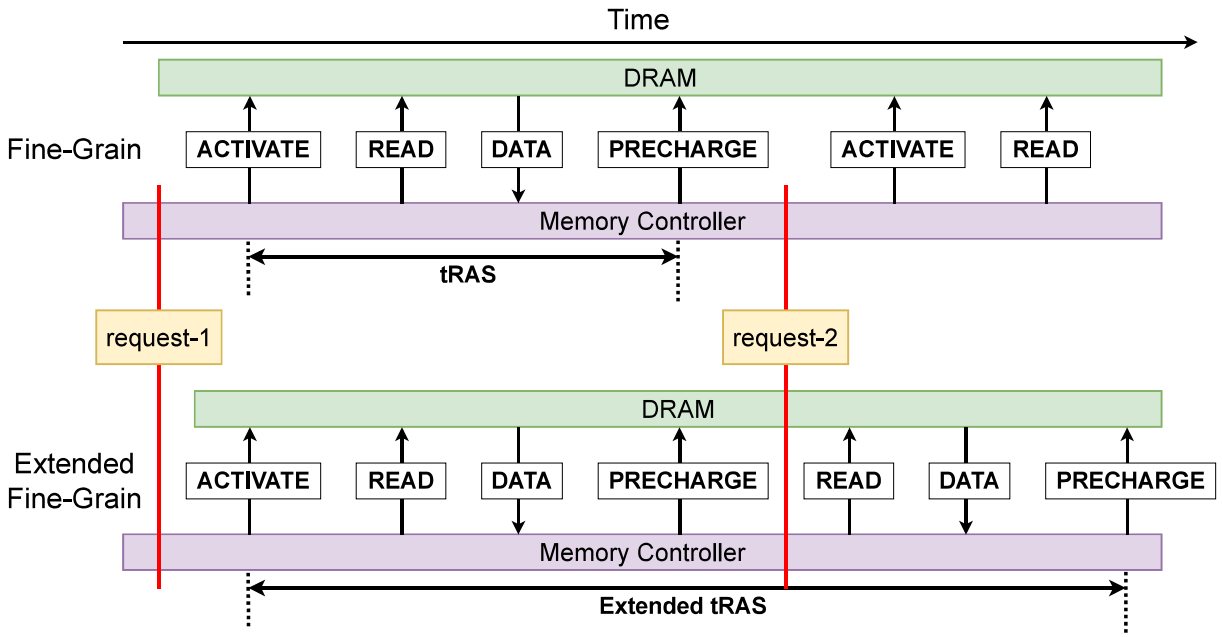


Figure 3.2: Behavior Comparison of the Fine-Grain Behavior Model and the Extended Fine-Grain Behavior Model [OK22b]

The extended fine-grain behavior model adjusts three DDR timing parameters: *tRAS*, *tRTP* and *tWTP*.

- PRECHARGE cannot be issued within *tRAS* from the preceding ACTIVATE.
- PRECHARGE cannot be issued within *tRTP* and *tWTP* from the preceding READ and WRITE, respectively.

Timing constraints of these three parameters must be satisfied before issuing PRECHARGE. Adjustment on only *tRCD* should work as expected, however, some memory controllers only check *tRTP* and *tWTP* before issuing PRECHARGE. ACTIVATE is issued to load data on memory

cells into row-buffers, then, at least one successive READ or WRITE is issued. In other words, "$tRCD + tRTP/tWP$" always elapsed before issuing PRECHARGE. $tRAS$ can be ignored if "$tRCD + tRTP/tWTP > tRAS$" is always satisfied depending on memory controller implementation and/or frequency. Therefore, the extended fine-grain also adjusts $tRTP$ and $tWTP$ in addition to $tRAS$ to strictly ensure timing constraints. $tRTP$ and tRP are calculated as follows:

$$\begin{aligned} tRTP &= max\{\ tRTP(spec),\ tRAS(rest)\ \} \\ tWTP &= max\{\ tWTP(spec),\ tRAS(rest)\ \} \end{aligned}$$

"(spec)" is the value defined in the DDR3 protocol. "(rest)" is the rest of $tRAS$ when READ or WRITE is issued. These definitions force to satisfy both $tRAS$ and $tRTP/tWTP$ before issuing PRECHARGE. These adjustments and the extended fine-grain behavior model do not violate the DDR3 protocol. Table 3.1 shows the DDR3 timing parameters defined in its specification. The proposed simulator uses DDR3-1600. Maximum values are not defined except for $tRAS$ in the specification. $tRAS$ is not set to such a big value when simulating NVMM performance since the NVMM latency should be hundreds of nanoseconds.

Table 3.1: DDR3-1600 Timing Parameter Specifications [OK22b]

Timing Parameter	Minimum [ns]	Maximum [ns]
tRCD	13.75	*undef.*
tRP	13.75	*undef.*
tRAS	35	70,200
tRTP	7.5	*undef.*
tWTP (tWR)	15	*undef.*

3.3 Details of Simulator Implementation

The proposed simulator is implemented on the SiFive Freedom U500 VC707 FPGA Dev Kit [SiF19]. It works on the Xilinx VC707 FPGA board. An overview of the simulator is listed in Table 3.2. Fig. 3.3 depicts the simulator block diagram. Freedom SoC has RISC-V Rocket cores employing RV64GC ISA, single issue 5-stage pipeline, in-order execution. It is fully open-source and compatible with Keystone TEE. Linux kernel was built on the e448fa3 commit of the Keystone repository. It is the extended Linux 5.6.0. The operating system is Debian RISC-V Ports.

This section describes the simulator implementation as follows:

1. Latency injection based on existing NVMM behavior models.
2. Latency injection based on the extended fine-grain behavior model.
3. Logically partitioned memory space.
4. Modification on RISC-V Rocket core for user-space cache flush.

Table 3.2: The Proposed NVMM Simulator Specifications [OK22b]

FPGA	Xilinx Virtex-7 FPGA VC707
Device	Virtex-7 XC7VX485T-2FFG1761
CPU	Rocket Core × 4
Rocket Core ISA	RV64GC (Unpriv 2.1, Priv 1.11)
L1 Cache	I=16 KiB/core, D=16 KiB/Core
System RAM	1 GiB, DDR3-1600, SO-DIMM
Configurable NVMM	3 GiB, DDR3-1600, SO-DIMM
SoC Frequency	50 MHz
Memory System Frequency	200 MHz
Linux Kernel	GNU/Linux riscv64 5.6.0-dirty
Operating System	Debian GNU/Linux bullseye/sid

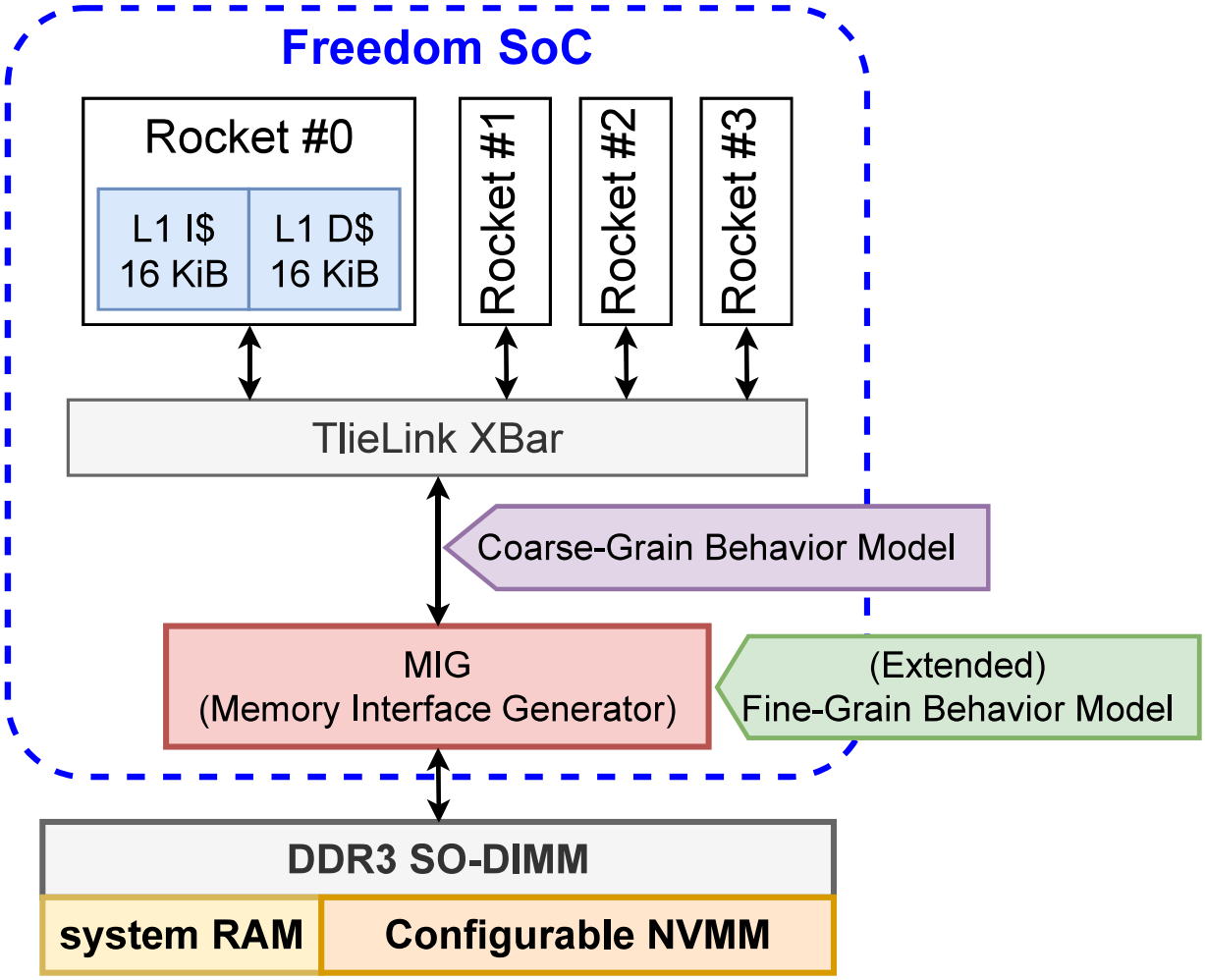


Figure 3.3: Block Diagram of the Proposed NVMM Simulator [OK22b]

3.3.1 Existing Delay Injections

The simulator has the existing three delay injections: Coarse-Grain (Section 2.3.1), Fine-Grain (Section 2.3.2), and DCPMM (Section 2.3.3). Their implementation is ported from the simulator described in Chapter 2.

3.3.2 Extended Fine-Grain Delay Injection

The CPUs (50MHz) on the simulator are slower than the memory system (200MHz) (Table 3.2). The extended fine-grain delay injection is based on the extended fine-grain behavior model (Section 3.2.2). It is implemented by modifying the MIG behavior like the original fine-grain delay injection. The modified MIG waits for additional *tRAS* after the original *tRAS*. This delay injection is an extension of the original one, thus, *tRCD* and *tRP* can be also specified to represent NVMM cells' latency. These three timing parameters (*tRCD*, *tRP*, and *tRAS*) can be specified

independently.

3.3.3 Logically Partitioned Memory Space

The ZC706 board used in Chapter 2 has two DIMMs. One DIMM is used as system RAM, and the other is used as NVMM. However, the VC707 board has only one DIMM. The one memory space is logically divided to realize heterogeneous memory systems. The memory is divided into two regions: system RAM, and user-controlled RAM as depicted in Fig. 3.3. The Linux kernel only manages the former region. The latter region can be used as the NVMM region. The proposed simulator has MMIO registers to configure the NVMM address range in the user-controlled RAM. The delay injections are applied to only memory accesses to the NVMM region.

3.3.4 RISC-V Design Modification for User-Space Cache Flush

As described in Section 2.3.5, cache flush instructions are required to use NVMM. While the instructions are not standardized in RISC-V ISAs (Unprivileged 20191213, Privileged 20211203) as of Nov 2022, Rocket cores on the proposed simulator have the custom cache management instructions ported from the SiFive RISC-V cores. The instruction, CFLUSH.D.L1 flushes a cacheline using its virtual address. However, user programs cannot call the privileged instruction. The kernel module like Section 2.3.5 is not a solution to this case because the instruction is available only on firmware privilege level (M-mode). A dedicated API to call it from a user-space program should cause large overheads due to multiple context switches. The simulator's Rocket cores are modified. On the proposed simulator, user programs and kernel programs can directly call CFLUSH.D.L1 without any context switch.

Besides, the instruction only focuses on only L1D$. It only evicts the specified line from L1D$ to the next level memory system. The instruction cannot guarantee data persistency on hierarchical CPU caches like an L2 cache. Rocket cores are often used with SiFive L2 inclusive cache. This chapter modifies the instruction behavior to completely evict the specified cacheline from CPU caches.

A program can directly call the modified CFLUSH.D.L1 regardless of privilege level. The specified cacheline is evicted to corresponding memory devices. An inline assembly of CFLUSH.D.L1 is shown below. The target virtual address is specified in reg. CFLUSH.D.L1 evicts the cacheline that contains data for the reg. If the zero register is specified (reg is 0), all L1D$ lines will be evicted.

```
(".insn i 0x73, 0, x0, %0, -0x340 :: "r"(reg));
```

3.4 Validation of NVMM Behavior Models

This section confirms the effectiveness of the extended fine-grain behavior model by comparing it with the existing NVMM behavior models. Besides, this section confirms the DCPMM delay injection by comparing it with a real DCPMM.

3.4.1 Extended Fine-Grain Behavior Model

As described in Section 3.2.2, the extended fine-grain behavior model should be able to capture the impact of row-buffer hit ratio (access locality). Algorithm 2 shows the pseudo code of the micro benchmark. The access stride (*STRIDE* in Algorithm 2) is set to 4096 or 8192. The NVMM on the simulator has 8192-Byte row-buffers. When *STRIDE* is set to 4096, half of the memory requests should hit row-buffers, then average latency is reduced by access locality. Additional latency is injected using three delay injections as follows:

- Coarse-Grain: 1,000ns for both read and write requests
- Fine-Grain: 1,000ns for both *tRCD* and *tRP*
- Extended Fine-Grain: 7,000ns for *tRAS*, in addition to the Fine-Grain

Table 3.3a shows average read latency in nanoseconds measured by using the micro benchmark (Algorithm 2). "(×N)" is the normalized latency against 8192. Only the extended fine-grain behavior model can capture the impact of access locality. However, the latency reduction ratio (0.67) is smaller than the expected ratio (0.50). If *STRIDE* is half of the row-buffer size, half of the memory requests hit row-buffers, then average latency should decrease by 50%. It is due to detailed behaviors of memory requests, a memory controller, and a memory module. Fig. 3.4 shows the details. In the figure, a CPU issues three memory read requests. "addr(X)" is a read request to address X. "data(X)" is a data acknowledgement corresponding with the "addr(X)". At first, the row-buffer for address 0-8191 is activated for "addr(0)". When the memory controller accepts "addr(4096)", it hits the row-buffer and "data(4096)" is returned without any memory cell accesses. The latency is 1,200 ns when viewed from the CPU (from "addr(4096)" to "data"). The next read request to address 8192 misses the row buffer. Although PRECHARGE and ACTIVATE are required to activate a new row buffer, *tRAS* must be satisfied. The analysis on the proposed simulator reveals that the request for 8192 takes about 6,000 ns (from "addr(8192)" to "data"). When $STRIDE$ is set to 4,096, two successive memory requests hit or miss row-buffers in turn. Half of the memory requests takes 1,200ns, the others take 6,000ns due to *tRAS*. As a result, the average latency when *STRIDE* is set to 4096 should be about $(1,200 + 6,000)/2 = 3,600$ ns. The result in Table 3.3a follows the expected behavior.

Table 3.3b shows average write latency in nanoseconds measured by using the micro benchmark (Algorithm 2). Unlike Table 3.3a, the extended fine-grain shows the same behavior as the original fine-grain. This behavior is due to CPU caches. The Rocket cores on the proposed simulator have write-back caches. Even if physically sequential write requests are issued, they will be randomized on the memory bus by cacheline replacement. Thus, access locality becomes lower than expected. The additional measurement for row-buffer hit ratio reveals that less than 5% of memory requests hit row-buffers when *STRIDE* is set to 4096. The behavior is allowable for NVMM performance simulation because such pure and heavy write accesses are not desirable for NVMM.

This section confirms that the extended fine-grain behavior model can capture the impact of row-buffer hit ratio even on slow soft processor systems. The extended fine-grain behavior model can explore optimization techniques for DRAM/NVMM heterogeneous memory systems focusing on access locality, which was impossible on the original extended fine-grain behavior model.

Table 3.3: Average Latency while Changing *STRIDE* [OK22b]

(a) Read Latency

	Average Latency [ns]		
STRIDE	Coarse-Grain	Fine-Grain	Extended Fine-Grain
4,096	2,702 (×0.94)	2,708 (×0.94)	4,081 (×0.67)
8,192	2,881	2,872	6,064

(b) Write Latency

	Average Latency [ns]		
STRIDE	Coarse-Grain	Fine-Grain	Extended Fine-Grain
4,096	4,399 (×0.98)	4,879 (×0.91)	14,457 (×0.91)
8,192	4,479	5,334	15,913

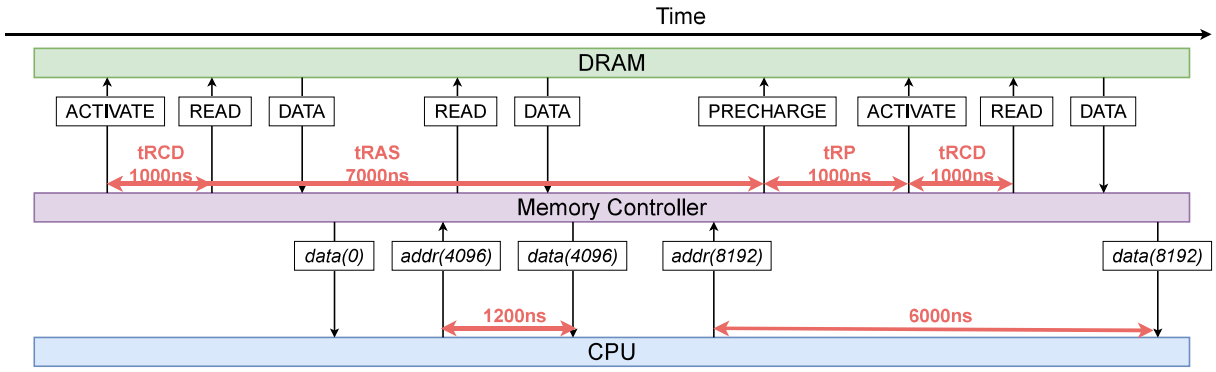


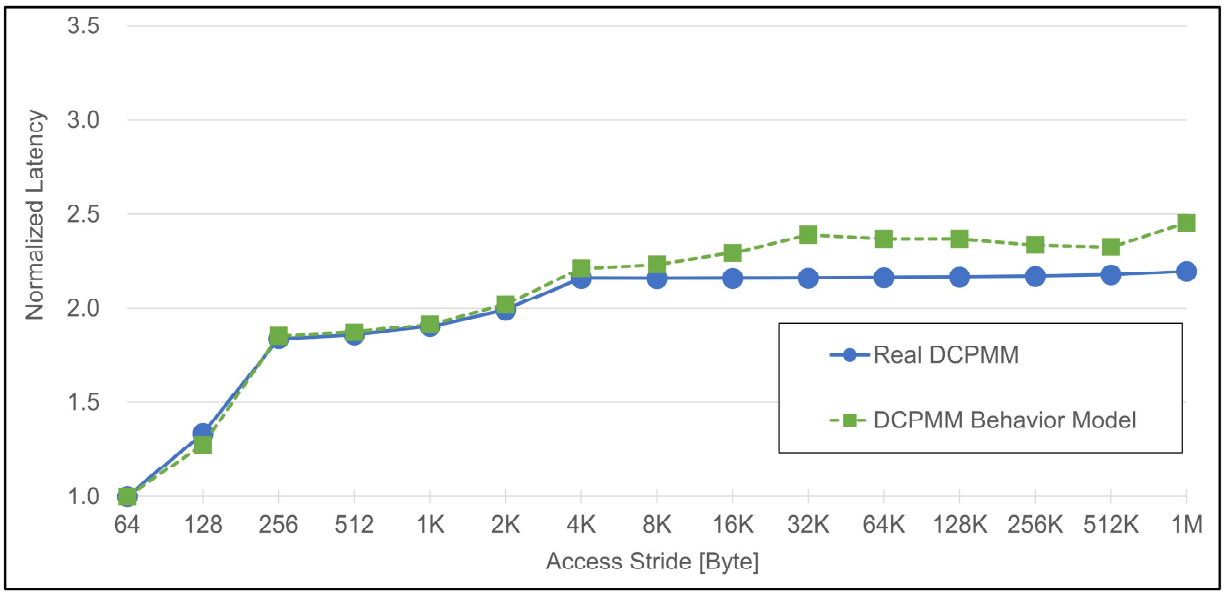
Figure 3.4: Detailed Timings of the Extended Fine-Grain Behavior Model [OK22b]

3.4.2 DCPMM Behavior Behavior Model

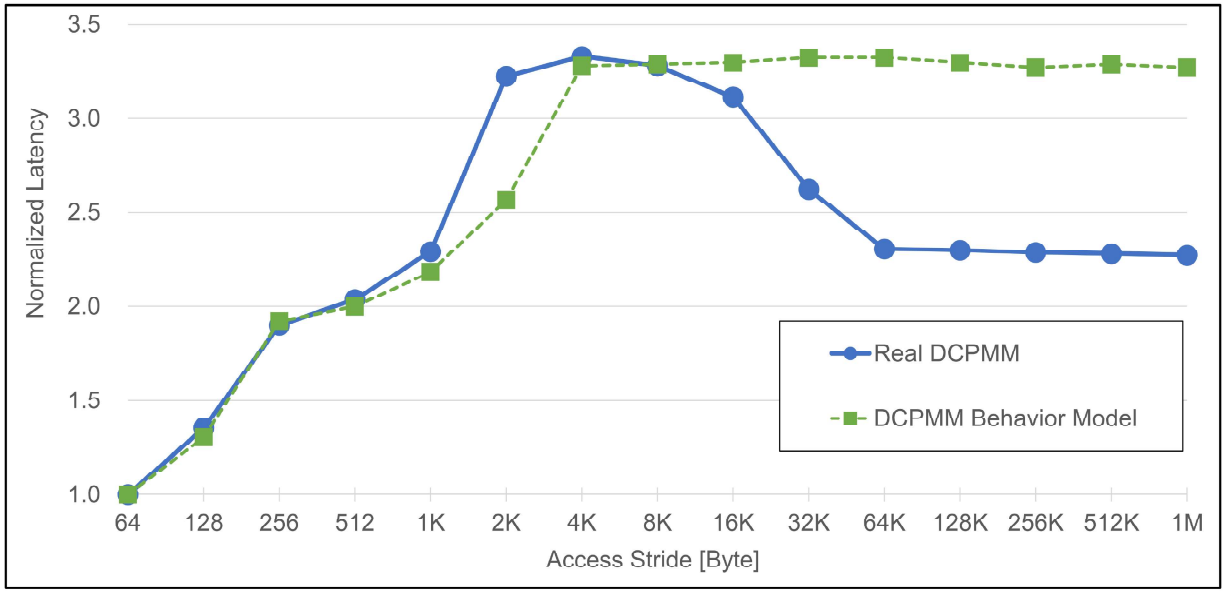
The DCPMM behavior model simulates the DCPMM behavior by relatively injecting additional latency. This section confirms the DCPMM behavior model on the proposed simulator. Average latency is measured using the same benchmark as Section 2.4.3 (Algorithm 1). The access stride, *STRIDE*, is set from 64 to 1-MiB. Additional latency is configured as follows according to the actual DCPMM behavior Fig. 2.3. Read latency increases to 1.84×/2.16× when crossing 256-Byte/4,096-Byte boundaries, respectively. Write latency increases to 1.90×/3.32× when crossing 256-Byte/4,096-Byte boundaries, respectively.

Fig. 3.5a and Fig. 3.5b show the results. Average latency is normalized against latency when *STRIDE* is set to 64. The blue lines plotted with circles and the green broken lines plotted with squares show the normalized average latency of a real DCPMM, and the DCPMM behavior model on the simulator, respectively. These two lines show almost the same behavior except for writes above 4-KByte. This difference is allowable as described in Section 2.4.3.

This section confirms that the DCPMM behavior model can simulate the real DCPMM behavior even on slow soft processor systems.



(a) Read Latency



(b) Write Latency

Figure 3.5: Average Latency while Changing *STRIDE* [OK22b]

3.5 Experimental Evaluation of Extended Fine-Grain Behavior Model with SPEC CPU 2017 Benchmark

This section confirms the effectiveness of the extended fine-grain behavior model by comparing it with the coarse-grain and the original fine-grain behavior models using SPEC CPU 2017 benchmarks [Sta]. The coarse-grain behavior model is used on behalf of the DCPMM behavior model as described in Section 2.5.1. Fourteen benchmarks are chosen from SPEC CPU 2017 rate benchmark programs. They can be compiled and executed on the proposed simulator. All memory allocations in the programs are replaced with the modified jemalloc [Eva06] to allocate heap objects on the NVMM region. Additional latency is configured as follows:

- Coarse-Grain: 1,000ns for both read and write requests
- Fine-Grain: 1,000ns for both *tRCD* and *tRP*
- Extended Fine-Grain: 7,000ns for *tRAS*, in addition to the Fine-Grain

Fig. 3.6 shows the results. Bar graphs are normalized execution time. The bars filled with dots (left), diagonal lines (middle), and crossed stripes (right) correspond to the coarse-grain, the fine-grain, and the extended fine-grain, respectively. They are normalized against the execution time on system RAM (without any additional latency). The line graph shows the memory access frequency measured on the memory bus on the proposed simulator. The programs are sorted in ascending order of normalized execution time of the extended fine-grain behavior model from left to right. Naturally, with memory access frequency increasing, execution time should be heavily affected by NVMM latency. Thus, with the crossed-stripe bar (the extended fine-grain behavior model) increasing from left to right, the line graph should also increase from left to right. Most of the benchmarks follow this expectation, however, a few of them marked with squares are contrary to the expectation. Two factors cause this behavior.

First, the programs marked with rounded squares (511.povray_r, 523.xalancbmk_r, and 505.mcf_r) have high access locality. Access locality is the row-buffer hit ratio, defined as the ratio of the number of memory requests without ACTIVATE to all memory requests issued from CPUs. While the average access locality between 14 benchmarks is 0.18, the exceptions have 0.52, 0.50, and 0.48, respectively. About half of memory requests hit row-buffers. This behavior is shown in only the extended fine-grain behavior model. The extended fine-grain behavior model can exploit the impact of access locality that is ignored by the existing coarse-grain and fine-grain behavior models. It proves the discussion in Section 3.4.1.

Second, the programs marked with sharp squares (525.x264_r, 503.bwaves_r, and 508.namd_r) have quite high read/write ratio. In this case, read/write ratio is defined as the ratio of read requests on a memory bus to write requests on a memory bus. The exceptions show 13.22, 5.44, and 5.40,

respectively. The (Extended) Fine-Grain Model has asymmetric read/write performance. The write latency is longer than read latency as real NVMM cells. Even though the same number of memory requests are issued to NVMM, read-intensive programs show smaller average latency. Thus, the extended fine-grain behavior model can exploit the impact of programs' read/write ratio.

This section confirms that the extended fine-grain behavior model can capture the impacts of memory access characteristics that are ignored in the existing coarse-grain and fine-grain behavior models. The factors are access locality and read/write ratio. These factors are important to reduce the impact of NVMM latency on system performance.

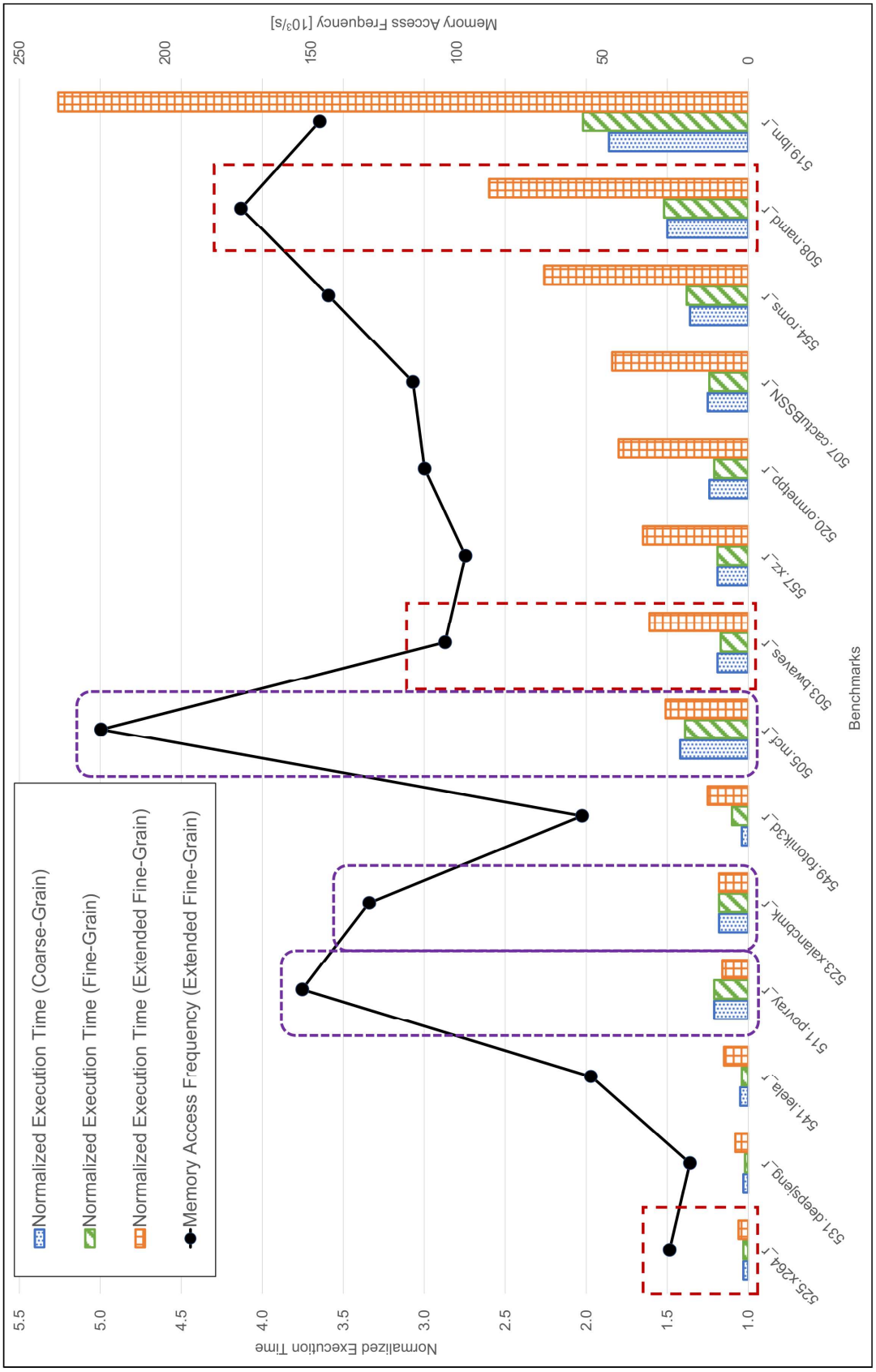


Figure 3.6: Normalized Execution Time of SPEC CPU 2017 Programs [OK22b]

3.6 Conclusion

This chapter proposed the hardware DRAM/NVMM heterogeneous memory simulator on the Xilinx VC707 FPGA. The simulator provides the new NVMM simulation: Extended Fine-Grain. It is the extension of the existing Fine-Grain to capture the impact of access locality even on soft processor systems. Three existing NVMM simulations are also supported: Coarse-Grain, Fine-Grain, and DCPMM. The proposed simulator provides the whole evaluation platform for DRAM/NVMM heterogeneous memory systems. Debian runs on RISC-V CPUs. The RISC-V CPUs support cache flush instructions available regardless of program privilege levels.

The extended fine-grain behavior model was validated against the existing NVMM behavior models. The validation showed that the extended fine-grain behavior model works as expected. Then, the experimental evaluation using SPEC CPU 2017 benchmarks revealed three essential factors in optimization for NVMM: memory access frequency, access locality, and read/write ratio. The last two factors can alleviate the first factor's impact; however, frequent memory accesses spoil them and severely degrade system performance.

Chapter 4

Secure Edge Computing Simulator Employing DRAM/NVMM Heterogeneous Memory Systems *

*This chapter is based on the paper "Open-Source Hardware Memory Protection Engine Integrated With NVMM Simulator", IEEE Computer Architecture Letters, vol.21 , no.2, pp.77-80, Aug. 2022.

4.1 Preface

As described in Section 1.4, edge devices have become important in smart IoT systems. A secure computing framework must be utilized to protect confidential information (trained AI models) and privacy information (input data) from software and hardware adversarial attackers. On IoT edge devices, not only privileged software but also off-chip hardware modules are untrusted because they may be tampered with by attackers. The expected framework, RISC-V Keystone TEE (Trusted Execution Environment), has two significant issues with memory systems: auxiliary devices and off-chip memory protection. These issues can be solved by NVMM-based non-volatile auxiliary devices as described in Section 1.4, and the MPE (Memory Protection Engine) as described in Section 1.5. Despite the demand, the combination of TEE, DRAM/NVMM heterogeneous memory systems, and MPE cannot be explored on existing platforms (Section 1.6).

This chapter proposes a secure edge computing simulator employing the combination. The simulator is based on the hardware DRAM/NVMM heterogeneous simulator proposed in Chapter 3. It has Keystone-compatible RISC-V CPUs. This chapter newly implements MPE based on SGX-style Integrity Tree, then integrated it into the simulator. It is designed to cover large memory regions with limited hardware resources. The MPE on the simulator is validated by micro benchmarks. Besides, this chapter discusses the proposed simulator's role in secure computing research.

This chapter is organized as follows: Section 4.2 discusses the threat model of the simulator. It represents the scope of predicted attacks and defenses. Section 4.3 introduces SGX-style Integrity Tree (SIT). Section 4.4 describes the MPE architecture based on SIT. Section 4.5 describes the whole simulator implementation. Section 4.6 measures MPE overheads on memory latency and validates the MPE implementation, then discusses the simulator's role by comparing it with the existing simulators. Section 4.7 concludes this chapter.

4.2 Threat Model

In secure computing systems, the threat model must be discussed. It represents the scope of predicted attacks and defense techniques for them; what attacks are defended by techniques, and what attacks are out-of-scope. Adversarial attackers are categorized into two groups: software attackers and hardware attackers [LKS+20].

Software attackers are capable of accessing a system running on the device with user level privilege at least. They can collect unprotected information about the system: kernel and OS versions, hardware modules, running processes, installed packages, users, and so on. This information may be used to crack/invade the whole system. In the worst case, attackers can control the whole system and modify the system behavior as they prefer, such as injecting loggers into system calls. Hardware attackers are capable of physically accessing a device. They can steal the device, connect arbitrary peripheral devices, replace existing modules, and tap signals on off-chip buses. In comparison to software attackers, hardware attackers can do almost anything. They can analyze the whole system quite easier than software attackers. All data on memory devices may be tampered with, not only off-chip DRAM but also on-chip flash devices. Even if memory devices are tamper-resistant, attackers can steal data on devices. Hardware attackers can also obtain memory information by tapping memory buses without any modification on system behaviors.

This dissertation focuses on edge devices. The devices are exposed to software attackers like traditional servers due to their internet connections. The devices are also severely exposed to hardware attackers. They are located near endpoints to gather data and/or near users to increase user experiences. Adversarial attackers can access the devices quite easier than servers securely located in data centers. Additional security modules are less effective because attackers can also touch them. Portable edge devices may be stolen. Therefore, to realize secure computing on edge devices, both software and hardware attackers must be considered.

Keystone TEE and MPE have different roles in secure computing. Keystone TEE mainly prevents software attacks by CPU-enforced memory isolation. Its strict memory isolation is always applied to all programs regardless of their privileged levels, except for system firmware. The isolated memory region (Enclave) is protected from adversarial programs during Keystone running. In contrast, MPE mainly prevents hardware attacks by memory encryption and tamper detection. All cachelines are encrypted by using securely stored keys and unique nonces before they leave secure on-chip modules. All cachelines are verified using integrity trees before they are loaded into on-chip modules. The complementary combination of TEE and MPE can prevent various attacks. The trusted computing base (TCB) of the combination should be also discussed. TCB is the "trusted" hardware/software modules. Hardware TCB contains only on-chip modules. On-chip modules always work as expected. Attackers cannot tamper with the modules [LKS+20]. All off-chip modules are untrusted. Software TCB contains the Keystone framework including the system firmware, TEE

runtime, and the program binary itself. They are tamper-resistant. In other words, even if they are tampered with, Keystone can detect the tampering by the attestation flow. All other modules are untrusted such as an OS, a kernel, other user processes, and other enclaves. The threat model in this dissertation is the extension of these TCBs with MPE.

From the hardware view, all on-chip modules are trusted. All securely stored data are allowed for specific on-chip modules. Malicious attackers cannot steal and/or tampered with description/encryption keys, hash keys, and integrity tree roots even if they have the same privilege as system softwares (OS, kernel). Off-chip memory is partially trusted. In other words, only the region covered by MPE is trusted because the region is protected by encryption and tamper detection. All other off-chip modules are untrusted. All data passed to them may be stolen and/or tampered with. When TEE uses them, data must be always encrypted and/or verified.

From the software view, only the Keystone TEE framework is trusted. The system firmware, TEE runtime, and the initialized enclave are trusted only after attestation. From the trusted on-chip root of trust, they are hierarchically proven to be trusted. All other software modules are untrusted. An adversarial attacker can pollute the whole system, launch adversarial enclaves, and modify data on unprotected memory regions.

Other typical attacks are assumed as follows. Cache and timing side-channel attacks are out of scope as the same as existing TEEs [LKS+20, CD16]. Controlled channel attack is difficult since Keystone TEE does not share any state with a host by a complete context switch [LKS+20]. Denial-of-Service by OS and memory tampering is out of scope as same as [LKS+20].

4.3 Overview of SGX-style Integrity Tree

The proposed MPE is based on SGX-style Integrity Tree. SGX-style Integrity Tree (SIT) is a kind of hashed octree (8-ary tree) [Gue16]. Fig. 4.1 shows the overview of a 4-level SIT. Each SIT node has one MAC (Message Authentication Code) and eight counters except for PD_Tag nodes. Each counter dominates one subtree and represents the number of updates on the subtree. The MAC is computed over its own eight counters and the corresponding counter in the parent node. The protected memory region is split into "CL"s by cacheline size. PD_Tag nodes store MACs of CLs. Thus, N-level SIT having R roots can cover $R \times 8^N$ CLs.

Each CL is protected by encryption and verification using two tweaks and on-chip keys. Two tweaks are used to prevent replay attacks: node/CL physical address and counters in SIT nodes. The physical address is a spatial unique tweak. Even if the same data is written into multiple lines, the hashed/encrypted results differ from each other. The counter is a temporal unique tweak. It is incremented whenever the CL is updated. Even if the same data is written into the same CL, the hashed/encrypted results differ every time. Therefore, replay attacks using a pair of known plain and cipher/hash is difficult for SIT.

The encryption scheme is AES-128 counter mode encryption [Nat01] using on-chip key K_E and tweaks. The verification scheme is hash comparison using MACs stored on SIT nodes computed by CWMAC [WC81] with on-chip two keys: K_P K_M, and tweaks. The keys K_E, K_P, and K_M are securely stored on trusted on-chip memory. They are configured by user defined values or randomly generated at the system boot.

When a memory controller accepts a memory read request to the protected region, the request is passed to MPE. MPE traverses SIT and fetches all corresponding nodes with the target CL. Then, MPE computes MACs for all fetched nodes using counters on them. If a SIT node is not tampered with, the computed MAC matches the stored MAC on the fetched node. If any pair of the MACs does not match, the read request is rejected and an error is thrown. Otherwise, SIT decrypts the fetched CL and returns it. When a memory controller accepts a write memory request to the protected region, the request is passed to MPE. First, MPE verifies the current CL as the same as read requests to prevent writing data to the region controlled by attackers. Any tampering on the protected region indicates that an attacker can control data on the region. If the verification succeeds, MPE encrypts the given data, increments SIT node counters, and re-computes MACs. Then, they will be written back to memory.

SIT-based memory protection requires securely stored tweaks. In other words, SIT nodes must be tamper-resistant. If an attacker can tamper with tweaks easily, all nodes also can be modified easily keeping the tree integrity. It is not realistic to store all SIT nodes on trusted on-chip memory. One 4-level SIT can protect a 256-KiB region using 68.5-KiB SIT nodes (64-Byte aligned 1096 nodes). As a result, 274-KiB on-chip memory is required per 1-MiB protected region. SIT utilizes

dependency between nodes described above to reduce on-chip area overhead. Only roots are stored on trusted on-chip memory, and other nodes are stored on untrusted off-chip memory. Even if ALL SIT nodes on off-chip memory are tampered with, SIT verification works as expected unless roots are stolen and/or tampered with. On-chip memory size can be reduced to 28-Byte per 1-MiB protected region.

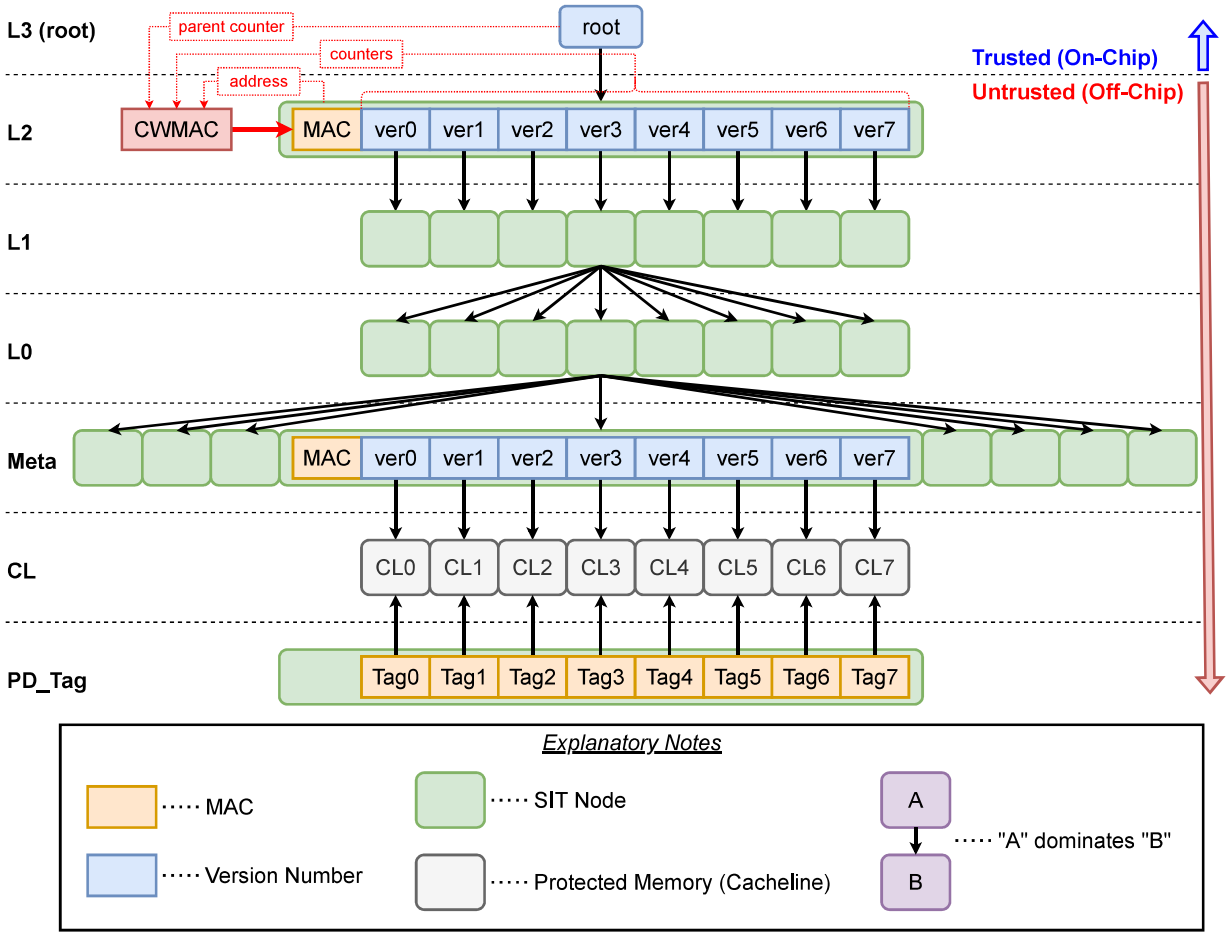


Figure 4.1: Overview of the SGX-style Integrity Tree

4.4 Overview of Memory Protection Engine

The proposed MPE uses 4-level SIT with 384 roots. Thus, the MPE can protect a 96-MiB memory region with a 32-MiB metadata region. Its configuration is the same as Intel SGX1. Fig. 4.2 depicts the MPE architecture. In Fig. 4.2, "trusted" modules are on-chip modules. A malicious attacker cannot observe their behaviors and steal and/or tampered with "trusted" data. The MPE is implemented as a module between the LLC and the MIG. It mainly consists of three modules: Frontend, Tree, and Backend.

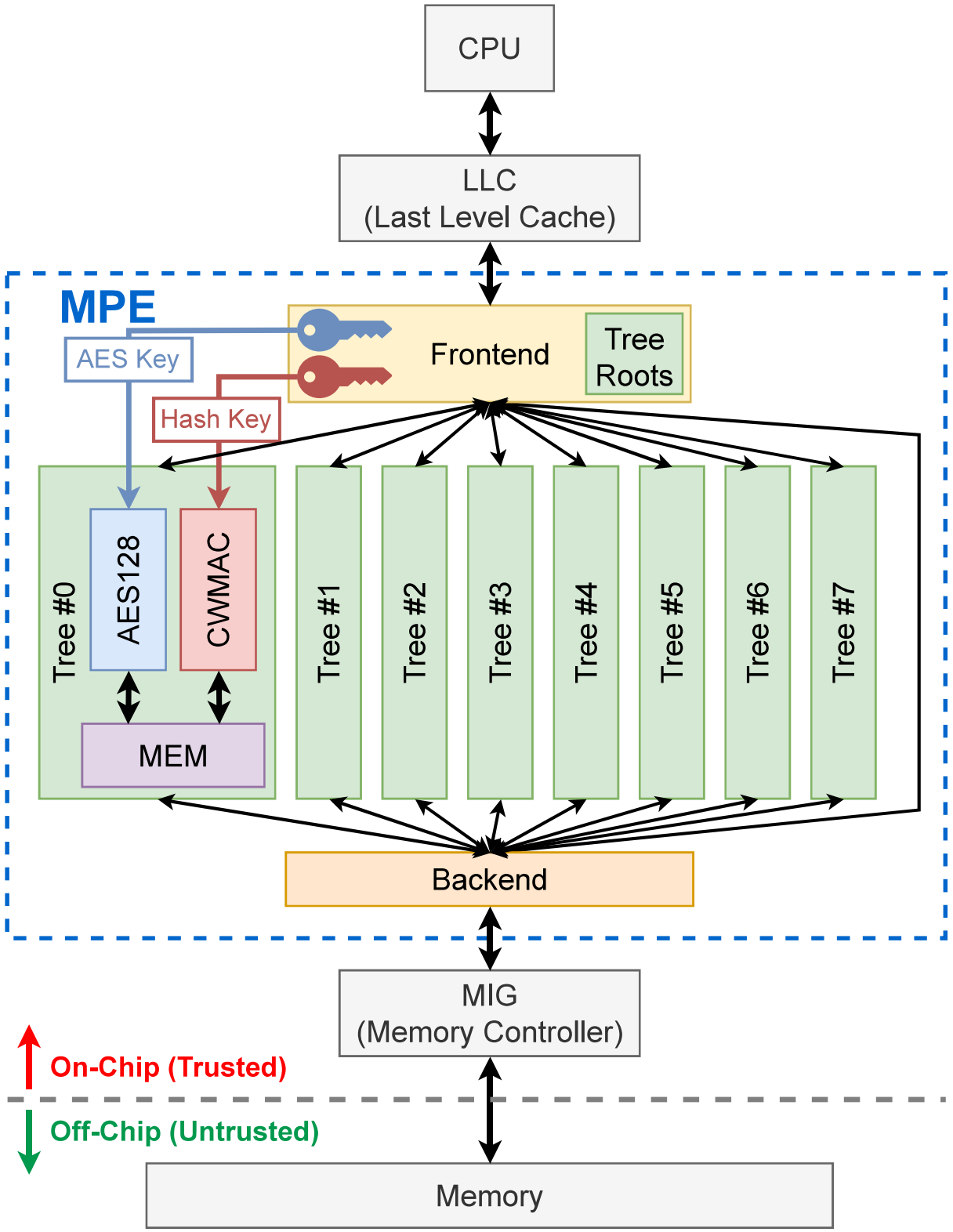


Figure 4.2: Memory Protection Engine Architecture [OK22a]

4.4.1 Frontend

The Frontend module is an arbiter for memory requests between the LLC and the Tree/Backend modules. It accepts a memory request from the LLC, then decides the appropriate destination module depending on the request address. If the address is a part of the protected region, the request is passed to Tree modules (Section 4.4.2). Otherwise, it is passed to the Backend module (Section 4.4.3). When the Tree/Backend modules become ready to return some acknowledgments, the Frontend module picks and returns it to the LLC. The Frontend module allows out-of-order memory request processing to reduce memory and SIT latency. It does not affect the behavior of the LLC and CPUs because randomly ordered return values are correctly re-ordered by them.

The Frontend module is also responsible for managing tree roots ("Tree Roots" in Fig. 4.2). In ideal, each root is statically assigned to one Tree module. However, 384 Tree modules for 384 roots require infeasible on-chip hardware resources. The proposed MPE dynamically assigns a tree root along with a memory request. Section 4.4.4 describes the detailed implementation. Tree roots are stored on trusted on-chip non-volatile registers and never leave on-chip registers. Thus, SIT can verify the protected NVMM region across power failure.

4.4.2 Tree

The Tree module processes a given memory request using 4-level SIT. When a pair of (memory request, root) is assigned from the Frontend module, at first, the Tree module traverses SIT to verify the target CL (Section 4.3). For a read request, the MPE returns a decrypted CL. For a write request, the MPE encrypts the given CL, and updates SIT nodes. If *any* MAC mismatch is found during the verification, the request is rejected and the MPE returns an error to the LLC (Section 4.4.5). Write requests with verification errors do not update the protected and SIT metadata region (Section 4.3).

The Tree module has three submodules: AES, CWMAC, and MEM. AES encrypts and decrypts CLs using AES128. CWMAC computes SIT node MACs using Carter-Wegman MAC [WC81]. They access the protected or SIT region via the MEM module. These three modules cooperatively work to minimize SIT overhead as much as possible. Fig. 4.3 is a gantt chart on a verilog simulation. One cell in the graph corresponds to one clock cycle. "Comp" represents the computed and the fetched MAC comparison. "Inc" represents a counter increment. All corresponding counters are incremented at a time in one cycle. "$\oplus$" represents AES XOR operations. The AES one-time pad (OTP) is speculatively computed over fetched "Meta node". For a read request, OTP uses the current (fetched) counter. For a write request, OTP uses the incremented counter. $\oplus$ in Decryption decrypts the fetched CL using the OTP, and $\oplus$ in Encryption encrypts the given CL using the OTP. All speculative data are discarded if a verification error occurs. According to Fig. 4.3, the verification finishes in 109 cycles, and update finishes in 73 cycles. A response to a read memory

request will be returned after $\oplus$ in the verification pipeline. On the other hand, a response to write memory request will be returned after $\oplus$ in the update pipeline. Thus, SIT takes 109 cycles per read requests, and 182 cycles per write requests. Memory requests without SIT can be returned in 18 read cycles ("Load CL") or 12 write cycles ("Store CL"). As a result, SIT overhead on a read request is 91 cycles, and that on a write request is 170 cycles

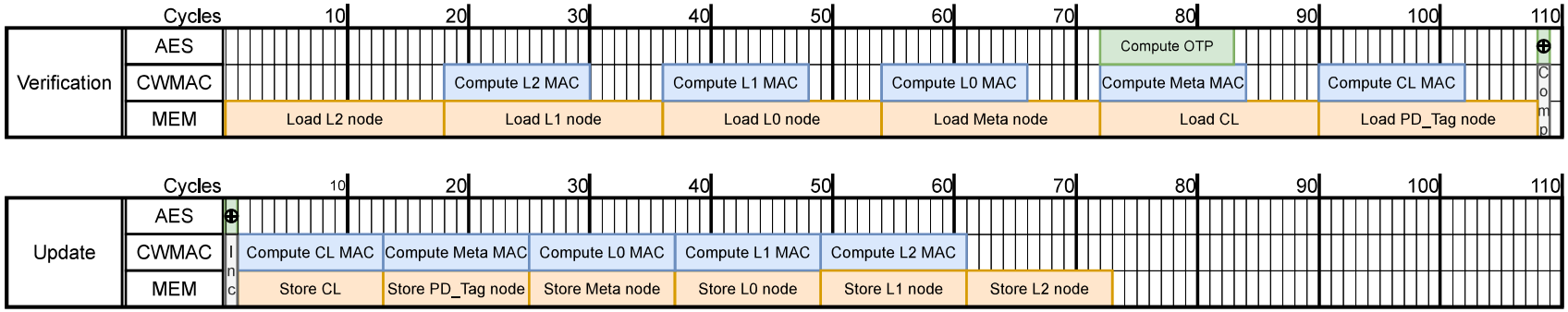


Figure 4.3: Gantt Chart of the Tree Module [OK22a]

4.4.3 Backend

The Backend module is an N-to-1 arbiter for memory requests between the Frontend/Tree modules and the MIG. The arbitration scheme is a round-robin FCFS (First-Come First-Served). One memory request can be issued to the MIG at a time. The Backend module does not support asynchronous or out-of-order execution to guarantee expected behaviors. Therefore, when multiple modules issue memory requests in parallel, they are pending and processed in-turn.

4.4.4 Efficient Large Memory Protection

As described in Section 4.4.1, static tree root assignment incurs infeasible hardware resources. The Frontend module manages tree roots ("Tree Roots" in Fig. 4.2) and dynamically assigns a corresponding root to a Tree module.

When the MPE accepts a memory request to the protected region, the Frontend module looks up a corresponding root, then tries to acquire a lock on the root. If it fails, the module stalls until the lock is released. Then, the Frontend module tries to assign the pair of (memory request, root) to an idle Tree module. If there is no idle Tree module, the Frontend module stalls until at least one Tree module is idle. When Tree modules become ready to return some acknowledgments, the Frontend module returns it to the LLC and releases the lock on the assigned root. The lock is used to prevent tree inconsistency due to simultaneous updates on tree nodes. This lock does not decrease Tree parallelism that can be exploited in static assignment. If successive requests are covered by different roots, they can be assigned to different Tree modules in parallel. More specifically, if a request is 256 KiB distant from the previous one, these two requests can be parallelized in the MPE.

Without the above tree root management, each root corresponds to one Tree module one-by-one ($C_0 \rightarrow Tree_0$, $\cdots$, $C_R \rightarrow Tree_R$). In that case, 384 roots require 384 Tree modules which require infeasible hardware resources. The proposed MPE can cover 384 roots by *only eight* Tree modules by the dynamic tree root assignment. One Tree module consumes 11,717 LUTs on average (Table 4.2). On static assignment, 384 Tree modules consume 4,499,328 LUTs (1,482% of available LUTs on the VC707 board). In contrast, the MPE only consumes 93,734 LUTs to cover 384 roots (98% of resources are saved). The Tree modules can be instantiated as much as hardware resources allow. The MPE design itself does not restrict the number of Tree modules. To maximize the MPE performance, tree modules should be instantiated as much as possible. More Tree parallelism can be exploited by more Tree modules. Insufficient Tree modules cause stalls from module thrashing and affect system performance seriously.

4.4.5 Integrity Error Notification

The SIT verification error must be immediately notified to prevent more malicious attacks. The original SIT uses the drop-and-lock policy [Gue16]. When the SIT finds a tampered node, the

whole memory system is completely locked. System reboot and key regeneration are required to unlock the memory system. This strict fail-secure behavior is quite fragile for memory bit errors. The memory system is locked even if ONLY ONE bit error is found in the protected region. A rowhammer-based attack can flip a bit of the region without bypassing hardware-enforced memory isolation. An adversarial attacker can halt the system by only launching a TEE process at user privilege level. When the attacker crashes the allocated TEE memory region, the whole system is locked. SGX-Bomb [JLLK17] demonstrates that a normal user can halt an SGX-enabled system through an ssh connection.

The proposed MPE uses an interrupt for notification instead of the drop-and-lock policy. A verification result bit (*corrupt* bit) is always returned to the LLC along with acknowledgments. The LLC raises an interrupt when the bit is high. The interrupt is passed to the system firmware (machine-mode SM), then delegated to other components following RISC-V and Keystone frameworks. Users can handle the error as they prefer by catching the interrupt. For example, a device sends an error report, or halts immediately. The MPE provides a way for flexible and efficient device management.

4.5 Simulator Implementation

The proposed simulator is based on the hardware DRAM/NVMM heterogeneous simulator (Chapter 3). Rocket cores on the simulator are compatible with Keystone TEE. Table 4.1 shows the proposed simulator specification. Its NVMM performance simulation is implemented the same as Chapter 3. All Coarse-Grain, Fine-Grain, Extended Fine-Grain, and DCPMM behavior models are implemented.

The MPE is integrated into the simulator to work transparently at the memory bus as depicted in Fig. 4.2. It is 4-level, 384 roots SIT, the same as SGX1. The MPE has one Frontend module, eight Tree modules, and one Backend module (Fig. 4.2). Resource utilization overview is listed in Table 4.2. The simulator has 4-GiB DRAM in total. 1 GiB is used as system RAM managed by the OS, the remaining region can be treated as the NVMM region. 128 MiB of the NVMM region is reserved by MPE (data 96 MiB + metadata 32 MiB). The NVMM region is excluded from system RAM to prevent unintentional allocation by the OS as the same as Section 2.3.4. The Linux kernel is not modified for the region cacheability because memory cacheability on RISC-V is determined by hardware implementation, not by the Linux OS. NVMM performance simulation can be applied to the whole NVMM region. Target regions and additional latency can be dynamically configured via MMIO registers.

The MPE was carefully implemented and tested. The AES module was tested using test vectors. While other modules including CWMAC and Tree itself were not verified due to the lack of reference testcases, they are tested by verilog simulation and benchmarks on Linux on the FPGA. The encryption, hash computation, and tree traversal work as expected.

Table 4.1: The Proposed Hardware Simulator Specification [OK22a]

FPGA	Xilinx Virtex-7 FPGA VC707
Device	Virtex-7 XC7VX485T-2FFG1761
CPU Core	Rocket-Core x4
RISC-V ISA	RV64GC (Unpriv 2.1 / Priv 2.1)
L1 Cache	I=16 KiB/core, D=32 KiB/core
L2 Cache	16-way 512 KiB/SoC
DRAM	4 GiB, DDR3-1600, SO-DIMM
SoC Frequency	50 MHz
Memory System Frequency	200 MHz
Linux Kernel	Linux 5.7.0-dirty / Debian bullseye
Operating System	Debian GNU/Linux bullseye/sid
MPE Modules	Frontend × 1, Tree × 8, Backend × 1
Integrity Tree	SGX-style Integrity Tree
Protected Memory Size	128-MiB (96-MiB data + 32-MiB SIT nodes)

Table 4.2: FPGA Resource Utilization [OK22a]

	LUT	BRAM
Whole SoC	257,265	762
Rocket Tile x4	106,244	168
MPE	99,327	37
Frontend	2,008	1
Tree x8	93,734	36
Backend	3,585	0
Total Available (Used %)	303,600 (84.7%)	1,030 (74.0%)

4.6 Experimental Evaluation

This section measures MPE overheads on memory latency, and validates the MPE implementation by comparing the behavior on an FPGA with that on the verilog simulator (Section 4.4.2). In addition, this section discusses the proposed simulator's role in comparison to the existing cycle-accurate simulator.

4.6.1 MPE Overhead on Memory Latency

Table 4.3 shows the results. Algorithm 2 shows the pseudo code of the micro benchmark. The "DRAM" rows show the results on DRAM (no additional latency). The "NVMM" rows show the results on the simulated DCPMM. DCPMM delay injection is configured the same as Section 3.4.2. "w/o MPE" shows memory latency on the unprotected region. "w/ MPE" shows memory latency on the protected region by the MPE. "w/ MPE" is normalized against the corresponding "w/o MPE". For instance, "DRAM w/ MPE Read" is normalized against "DRAM w/o MPE Read" ($168/66 = 2.55$). The results are measured under the following conditions.

- All cachelines are invalidated in advance.
- All results are the average of ten times.
- Access strides are 64-Bytes (cacheline size). All memory requests miss CPU caches and use different SIT nodes.

The "DRAM" rows indicate that the MPE increases DRAM read latency 2.55×, and DRAM write latency 4.16×, respectively. The MPE overheads are 102 read cycles, and 253 write cycles. Section 4.4.2 shows that the Tree module itself causes 91/170 cycles for read/write requests, respectively. The results on Table 4.3 were measured on the CPU with caches, thus, "Read" requires one Tree's read, and "Write" requires both Tree's read and write. Tree's overhead should be 91 cycles for "Read", and 261 cycles for "Write". While there are slight differences between the result and the expectation (+11 read cycles, -8 write cycles), they are acceptable. The overhead described in Section 4.4.2 includes only Tree module overheads. From a CPU view, memory latency includes other modules' overheads: the MPE (Frontend, Tree, Backend), the memory bus, and CPU caches. CPU memory requests are passed to the next modules after a handshake between modules. Some overheads are introduced to CPU's read requests due to these handshakes. They also cause overheads on write requests, however, CPU caches often alleviate them. The errors caused by these factors are acceptable. The result and discussion above indicate that the MPE on the simulator works as expected.

The "NVMM" rows indicate that the MPE increases NVMM read latency 3.05×, and NVMM write latency 5.40×, respectively. Focusing on the gantt chart (Fig. 4.3), the MPE throughput is

dominated by memory accesses. The gantt chart was measured on DRAM (no additional latency). NVMM longer latency should cause more stall cycles on the AES and the CWMAC modules. To reduce stall cycles, some proposed techniques are required, such as node caches.

Table 4.3: MPE's Performance Impact on Memory Latency [OK22a]

			Memory Latency [cpu cycles]	Normalized Performance
DRAM	w/o MPE	Read	66	-
		Write	80	-
	w/ MPE	Read	168	2.55×
		Write	333	4.16×
NVMM	w/o MPE	Read	80	-
		Write	98	-
	w/ MPE	Read	244	3.05×
		Write	529	5.40×

4.6.2 Evaluation Time Comparison to Cycle Accurate Simulator

Software simulators, gem5 [BBB+11], for secure computing have been proposed [HLC+21, AAPLP21]. They are useful for detailed architecture research, however, the papers also show that simulators take a long evaluation time. This section discusses the proposed simulator's role in secure computing research on edge devices focused on evaluation time.

In general, the proposed simulator can evaluate a program using TEE tens to hundreds of times faster than gem5. The average execution time in ten times of the simple Keystone ocall example [1] on the proposed simulator is 29 seconds, while that on the full-system gem5 is 1,750 seconds. The gem5 21.1 was configured to simulate the proposed simulator (Table 4.1). Specifically, it has one in-order (minor) CPU, 16K L1I$, 32K L1D$, 512K L2$, and non-protected 1-GiB DRAM. Even the simple benchmark causes a 60× performance gap. It increases on large, complex, and detailed TEE applications, and/or hardware modules. For example, the gem5 used in the measurement has only 1 core. Exploring multi-threaded TEE should require quite a long time.

The above discussion uses the result on the unprotected DRAM region. The NVMM region and the MPE incur more overheads. TEE on the protected NVMM cannot be evaluated on the existing Keystone because its memory space must be built on system RAM. The below discussion estimates the impact of the protected NVMM. An example is a context switch between TEE and a host OS. It is one of the inevitable NVMM intensive tasks when using TEE on the protected NVMM. Keystone evicts the whole private caches including L1D$ on every context switch, not only voluntary switches but also implicit switches (timer interrupts etc.). If a TEE uses the protected NVMM, most of the

[1] https://github.com/keystone-enclave/keystone-sdk/tree/master/examples/hello-native

dirty cachelines are fetched from the region and evicted to the region. The above ocall example on the proposed simulator takes 7,910 cycles on average for context switches with 225 dirty L1D$ lines (43.9% L1D$). Cache eviction to the protected NVMM takes 285 cycles per cachelines according to Table 4.3. On the table, "Read" contains one cacheline load, and "Write" contains one cacheline load and eviction. Thus, pure cacheline eviction should take 285 cycles on the protected NVMM. As a result, if 90% of dirty lines are written back to the protected NVMM, the context switch with the protected NVMM is estimated to take $7,910 + 285 \times (225 \times 0.9) = 65,623$ cycles. This is 8.3× larger than that without cache eviction. The protected NVMM latency also affects other memory accesses in the whole program execution. Software simulators may be affected by the protected NVMM, then hundreds to thousands of times slower than the proposed hardware simulator. Therefore, the proposed simulator has an important role in exploring the combination of TEE, MPE, and NVMM.

The above discussions clarify the importance of the proposed simulator in system-wide exploration for secure computing on edge devices. However, software simulators also have important roles. The proposed simulator can evaluate system-wide applications in a realistic time. On the other hand, cycle-level software simulators are suitable for microarchitecture level optimizations with strictly reproducible behavior and detailed statistics. As the evolution from SGX1 to SGX2, micro architecture also should be optimized. The proposed simulator and the software simulators complement each other.

4.7 Conclusion

This chapter proposed the secure computing simulator on a Xilinx VC707 FPGA. The simulator solved two significant issues of Keystone TEE: auxiliary devices and off-chip memory protection, by integrating NVMM simulation and Memory Protection Engine (MPE) into a RISC-V SoC. The MPE is based on SGX-style Integrity Tree. The simulator provides a way to explore the combination of RISC-V Keystone TEE, DRAM/NVMM heterogeneous memory systems, and MPE.

The MPE was validated using verilog simulation and micro benchmarks. Experimental evaluation using the micro benchmark showed that the MPE increases DRAM read/write latency by 2.55×/4.16×, and simulated DCPMM read/write latency by 3.05×/5.40×, respectively. These overheads should prolong the existing software simulators. The proposed simulator can run secure programs with these overheads in a realistic time.

Chapter 5

Conclusion

5.1 Summary of Works

This dissertation proposed hardware simulators for DRAM/NVMM heterogeneous memory systems, and secure computing employing the memory systems. The simulators provide ways to explore system-wide optimization techniques for them, which were challenging on existing works.

This dissertation is summarized as follows:

ARM-based Hardware DRAM/NVMM Heterogeneous Memory Simulator (Chapter 2). Heterogeneous memory systems consisting of DRAM and NVMM are expected as efficient memory systems on IoT edge devices. To fully exploit the memory system advantages, the whole system must be optimized including hardware modules, system software, and user software. Existing works cannot satisfy the demand due to the tradeoffs between evaluation speed, and accuracy of evaluation stats. In addition, existing works do not provide DCPMM simulations. The proposed simulator provides the evaluation platform for DRAM/NVMM heterogeneous memory systems on the ARM-based hard SoC. Three NVMM architectures can be simulated: Coarse-Grain, Fine-Grain, and DCPMM. The Coarse-Grain and Fine-Grain can simulate NVMM behaviors with DRAM-like internal architecture. The DCPMM behavior can simulate the actual DCPMM behavior. These validated NVMM simulations can be flexibly configured to represent various NVMM cell latency. The simulator provides a fast and reliable way to explore system-wide optimization techniques focusing on program characteristics: memory access frequency, access locality, and bank parallelism.

RISC-V-based Hardware DRAM/NVMM Heterogeneous Memory Simulator (Chapter 3). DRAM/NVMM heterogeneous memory systems are expected to be used with soft processor systems in the future. The existing NVMM simulations are not suitable for soft processor systems because they presuppose that a CPU runs sufficiently faster than its memory system. Soft processors are sometimes slower than memory systems. New NVMM simulation techniques are required for soft processor systems. Thus, the proposed simulator provides the evaluation platform for DRAM/NVMM heterogeneous memory systems on the RISC-V-based soft SoC. The new Extended Fine-Grain NVMM simulation technique is introduced. It can capture program characteristics that are ignored in existing NVMM simulations, even on soft processor systems. The simulator provides a fast and reliable way to explore system-wide optimization techniques focusing on program characteristics: memory access frequency, access locality, and read/write ratio.

Secure Computing Simulator Employing DRAM/NVMM Heterogeneous Memory Systems (Chapter 4). With growing smart IoT systems, security on edge devices has been increasingly important to protect confidential information (trained AI models) and privacy information (input data). The devices are always exposed to not only software attackers but also hardware attackers. RISC-V Keystone TEE is the expected secure computing framework on IoT

edge devices. However, Keystone has two significant issues of memory systems: auxiliary devices and off-chip memory protection. They can be solved by NVMM-based devices and MPE (Memory Protection Engine). Despite the demand, there are no existing evaluation platforms that fully support the combination of TEE, DRAM/NVMM heterogeneous memory systems, and MPE. The proposed simulator provides the evaluation platform for the combination. The simulator has all of RISC-V Keystone TEE, DRAM/NVMM heterogeneous memory systems, and SGX-style Integrity Tree based MPE. The MPE can efficiently protect a large memory region with limited hardware resources. Experimental evaluation using the micro benchmark showed that the MPE increases DRAM read/write latency to 2.55×/4.16×, and simulated DCPMM read/write latency to 3.05×/5.40×, respectively. The simulator provides a fast and reliable way to explore system-wide optimization techniques for the combination of TEE, DRAM/NVMM heterogeneous memory systems, and MPE.

5.2 Future Works

Some works can be explored using the proposed evaluation platforms:

Exploration of Real Program Optimization for DRAM/NVMM Heterogeneous Memory Systems. Chapter 2 and Chapter 3 proposed the evaluation platforms for DRAM/NVMM heterogeneous memory systems on edge devices. The platforms realized well-validated NVMM simulations even when lacking real NVMM modules. This dissertation revealed the essential factors for DRAM/NVMM heterogeneous memory systems using SPEC CPU 2017 benchmarks. System-wide optimization for the heterogeneous memory systems can be explored on the platforms. The Linux DAX driver, file systems, and user programs are examples. Their NVMM-friendly data structures can be explored on the proposed platforms focusing on memory access frequency, access locality, bank parallelism, and read/write ratio.

Fully Verifiable Secure Computing Platforms. This dissertation proposed the fully open-source MPE using SGX-style Integrity Tree. All modules on TEEs should be formally verified. They are expected to be bug-free and work as "expected" for any case [LKS+20]. Its behavior cannot be confirmed on closed implementations. The proposed MPE with Keystone TEE can realize fully-verifiable and trustable secure computing platforms. Users can verify whole components on trusted computing as they prefer.

Advanced Tree Module Design. Chapter 4 proposed the basic hardware MPE design using SGX-style Integrity Tree. The MPE increases DRAM read/write latency to 2.55×/4.16×, and simulated DCPMM read/write latency to 3.05×/5.40×, respectively. These overheads are applied to all memory requests to the protected region. Various techniques have been proposed to improve MPE throughput such as tree node caches. Node caches can improve MPE throughput at the sacrifice of on-chip hardware resources, design complexity, and timing constraints. Besides, node caches for covered NVMM regions must be also crash-consistent and recoverable across power failures. Such node caches require more complex designs, performance overheads, and on/off-chip memory spaces. These tradeoffs must be considered in more advanced MPE designs: Osiris [YHA18], Anubis [ZA19], PSIT [LWF+20], CacheTree [CZX21], STAR [HH21], Triad-NVM [AYS+19], Morphable Counters [SNR+18], and Phoenix [AZMA22]. The proposed MPE and evaluation platforms can be the basic hardware design for the exploration.

Keystone on the DRAM/NVMM Heterogeneous Memory Systems. As described in Section 4.6.2, the current Keystone does not support heterogeneous memory systems. Its memory space must be physically sequential, and built on system RAM. The whole Keystone framework must be extended to support DRAM/NVMM heterogeneous memory systems. It consists of security

monitors (firmware), eyrie runtimes, kernel drivers, and user programs. Their extensions must be optimized for heterogeneous memory systems through evaluations. The proposed secure computing simulator provides a way for its exploration.

Acknowledgements

I am grateful for my supervisor Prof. Keiji Kimura and Prof. Hironori Kasahara's invaluable advises through master and doctoral courses. I also appreciate Prof. Nozomu Togawa's constructive and kind suggestions.

I would like to thank Mr. Toshiya Otomo and Mr. Tomokazu Yoshida from Fixstars for valuable discussions.

Part of this dissertation was executed under the cooperation of organizations between Kioxia Corporation and Waseda University.

Bibliography

[AAPLP21] Ayaz Akram, Venkatesh Akella, Sean Peisert, and Jason Lowe-Power. Enabling Design Space Exploration for RISC-V Secure Compute Environments. In *Fifth Workshop on Computer Architecture Research with RISC-V (CARRV 2021)*, 2021.

[Alv04] Tiago Alves. Trustzone: Integrated Hardware and Software Security. *White paper*, 2004.

[AMD15] AMD. High-Bandwidth Memory (HBM). Technical report, AMD, 2015.

[ARM08] ARM®. *ARM® Architecture Reference Manual; ARM™ v7-A and ARM™ v7-R edition*, Apr 2008.

[ARM13] ARM. *AMBA® AXI® and ACE® Protocol Specification*, 2013.

[ATM+] Yuta Aiba, Hitomi Tanaka, Takashi Maeda, Keiichi Sawa, Fumie Kikushima, Masayuki Miura, Toshio Fujisawa, Mie Matsuo, and Tomoya Sanuki. cryogenic operation of 3d flash memory for new applications and bit cost scaling with 6-bit per cell (hlc) and beyond. In *2021 5th IEEE Electron Devices Technology & Manufacturing Conference (EDTM)*.

[AYS+19] Amro Awad, Mao Ye, Yan Solihin, Laurent Njilla, and Kazi Abu Zubair. Triad-NVM: Persistency for Integrity-Protected and Encrypted Non-Volatile Memories. In *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, pages 104–115, 2019.

[AZMA22] Mazen Alwadi, Kazi Abu Zubair, David Mohaisen, and Amro Awad. Phoenix: Towards Ultra-Low Overhead, Recoverable, and Persistently Secure NVM. *IEEE Transactions on Dependable and Secure Computing*, 19(2):1049–1063, 2022.

[BasFC+90] J. Barnaś, A. Fuss, R. E. Camley, P. Grünberg, and W. Zinn. Novel Magnetoresistance Effect in Layered Magnetic Structures: Theory and Experiment. *Phys. Rev. B*, 42:8110–8120, Nov 1990.

[BBB+11] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, August 2011.

[BCMM15] S. Bock, B. R. Childers, R. Melhem, and D. Mosse. HMMSim: A Simulator for Hardware-Software Co-Design of Hybrid Main Memory. In *2015 IEEE Non-Volatile Memory System and Applications Symposium (NVMSA)*, pages 1–6, August 2015.

[CAD+10] E. Chen, D. Apalkov, Z. Diao, A. Driskill-Smith, D. Druist, D. Lottis, V. Nikitin, X. Tang, S. Watts, S. Wang, S. A. Wolf, A. W. Ghosh, J. W. Lu, S. J. Poon, M. Stan, W. H. Butler, S. Gupta, C. K. A. Mewes, Tim Mewes, and P. B. Visscher. Advances and Future Prospects of Spin-Transfer Torque Random Access Memory. *IEEE Transactions on Magnetics*, 46(6):1873–1878, 2010.

[CD16] Victor Costan and Srinivas Devadas. Intel SGX Explained. *IACR Cryptol. ePrint Arch.*, 2016(86):1–118, 2016.

[CZX21] Zhengguo Chen, Youtao Zhang, and Nong Xiao. CacheTree: Reducing Integrity Verification Overhead of Secure Nonvolatile Memories. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40(7):1340–1353, 2021.

[DKK+14] Subramanya R. Dulloor, Sanjay Kumar, Anil Keshavamurthy, Philip Lantz, Dheeraj Reddy, Rajesh Sankaran, and Jeff Jackson. System Software for Persistent Memory. In *Proceedings of the Ninth European Conference on Computer Systems*, EuroSys '14, New York, NY, USA, 2014. Association for Computing Machinery.

[DLLJ18] Zhuohui Duan, Haikun Liu, Xiaofei Liao, and Hai Jin. HME: A lightweight emulator for hybrid memory. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1375–1380, 2018.

[ECC14] F. A. Endo, D. Couroussé, and H. Charles. Micro-architectural Simulation of In-order and Out-of-order ARM Microprocessors with gem5. In *2014 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIV)*, pages 266–273, 2014.

[ECL+07] Reouven Elbaz, David Champagne, Ruby B. Lee, Lionel Torres, Gilles Sassatelli, and Pierre Guillemin. TEC-Tree: A Low-Cost, Parallelizable Tree for Efficient Defense Against Memory Replay Attacks. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007*, pages 289–302, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[Eva06] Jason Evans. A Scalable Concurrent malloc(3) Implementation for FreeBSD. In *BSDcan conference*, April 2006.

[FLD+21] Erhu Feng, Xu Lu, Dong Du, Bicheng Yang, Xueqiang Jiang, Yubin Xia, Binyu Zang, and Haibo Chen. Scalable Memory Protection in the PENGLAI Enclave. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*, pages 275–294. USENIX Association, July 2021.

[GDD+20] Andrade Gui, Lee Dayeol, Kohlbrenner David, Asanović Krste, and Dawn Song. Software-Based Off-Chip Memory Protection for RISC-V Trusted Execution Environments. In *Fourth Workshop on Computer Architecture Research with RISC-V (CARRV 2020)*, 2020.

[GLH+20] Peng Gu, Benjamin S. Lim, Wenqin Huangfu, Krishan T. Malladi, Andrew Chang, and Yuan Xie. NMTSim: Transaction-Command Based Simulator for New Memory Technology Devices. *IEEE Computer Architecture Letters*, 19(1):76–79, 2020.

[Gue16] Shay Gueron. A Memory Encryption Engine Suitable for General Purpose Processors. *IACR Cryptol. ePrint Arch.*, 2016:204, 2016.

[HH21] Jianming Huang and Yu Hua. A Write-Friendly and Fast-Recovery Scheme for Security Metadata in Non-Volatile Memories. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 359–370, 2021.

[HLC+21] Peter Yuen Ho Hin, Xiongfei Liao, Jin Cui, Andrea Mondelli, Thannirmalai Muthukaruppan Somu, and Naxin Zhang. Supporting RISC-V Full System Simulation in gem5. In *Fifth Workshop on Computer Architecture Research with RISC-V (CARRV 2021)*, 2021.

[Int19] Intel. *Intel® Optane™ DC Persistent Memory Product Brief*, Apr 2019.

[Int21] Intel. *Intel® Architecture Memory Encryption Technologies*, Apr 2021.

[Int22a] Intel. *Intel® 64 and IA-32 Architectures Optimization Reference Manual*, February 2022.

[Int22b] Intel. *Intel® 64 and IA-32 Architectures Software Developer's Manual*, April 2022.

[JED12] JEDEC SOLID STATE TECHNOLOGY ASSOCIATION. *DDR3 SDRAM Standard (JESD79-3E)*, July 2012.

[JED22] JEDEC SOLID STATE TECHNOLOGY ASSOCIATION. *JEDEC STANDARD DDR5 SDRAM (JESD79-5B)*, Aug 2022.

[JLLK17] Yeongjin Jang, Jaehyuk Lee, Sangho Lee, and Taesoo Kim. SGX-Bomb: Locking Down the Processor via Rowhammer Attack. In *Proceedings of the 2nd Workshop on System Software for Trusted Execution*, SysTEX'17, New York, NY, USA, 2017. Association for Computing Machinery.

[KHA+17] A. Koshiba, T. Hirofuchi, S. Akiyama, R. Takano, and M. Namiki. Towards Write-back Aware Software Emulator for Non-Volatile Memory. In *2017 IEEE 6th Non-Volatile Memory Systems and Applications Symposium (NVMSA)*, pages 1–6, August 2017.

[KHC18] Asif Ali Khan, Fazal Hameed, and Jeronimo Castrillon. NVMain Extension for Multi-Level Cache Systems. In *Proceedings of the Rapido'18 Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools*, RAPIDO '18, New York, NY, USA, 2018. Association for Computing Machinery.

[KPW16] David Kaplan, Jeremy Powell, and Tom Woller. *AMD memory encryption*, 2016.

[LKP+14] T. Lee, D. Kim, H. Park, S. Yoo, and S. Lee. FPGA-based Prototyping Systems for Emerging Memory Technologies. In *2014 25nd IEEE International Symposium on Rapid System Prototyping*, pages 115–120, October 2014.

[LKS+20] Dayeol Lee, David Kohlbrenner, Shweta Shinde, Krste Asanović, and Dawn Song. Keystone: An Open Framework for Architecting Trusted Execution Environments. In *Eurosys '20*, EuroSys '20, New York, NY, USA, 2020. Association for Computing Machinery.

[LLZ12] Xu Li, Kai Lu, and Xu Zhou. SIM-PCM: A PCM Simulator Based on Simics. In *2012 Fourth International Conference on Computational and Information Sciences*, pages 1236–1239, 2012.

[LWF+20] Mengya Lei, Fang Wang, Dan Feng, Fan Li, and Jie Xu. An Efficient Persistency and Recovery Mechanism for SGX-style Integrity Tree in Secure NVM. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 702–707, 2020.

[LX19] Lily Looi and Jianping Jane Xu. INTEL® OPTANE™ DATA CENTER PERSISTENT MEMORY. In *Hot Chips (HC) 31*, Aug 2019.

[LY17] T. Lee and S. Yoo. An FPGA-based Platform for Non Volatile Memory Emulation. In *2017 IEEE 6th Non-Volatile Memory Systems and Applications Symposium (NVMSA)*, pages 1–4, August 2017.

[Mer80] Ralph Merkle. Protocols for Public Key Cryptosystems. In *IEEE Symposium on Security and Privacy*, pages 122–134, 04 1980.

[MWA+21] Haoyuan Ma, You Wang, Rashid Ali, Zhengyi Hou, Deming Zhang, Erya Deng, Gefei Wang, and Weisheng Zhao. SpinSim: A Computer Architecture-Level Variation Aware STT-MRAM Performance Evaluation Framework. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, 2021.

[Nat01] National Institute of Standards and Technology (NIST). *Advanced Encryption Standard (AES) (FIPS PUB 197)*, November 2001.

[OK21] Yu OMORI and Keiji KIMURA. Non-Volatile Main Memory Emulator for Embedded Systems Employing Three NVMM Behaviour Models. *IEICE Transactions on Information and Systems*, E104.D(5):697–708, 2021.

[OK22a] Yu Omori and Keiji Kimura. Open-Source Hardware Memory Protection Engine Integrated With NVMM Simulator. *IEEE Computer Architecture Letters*, 21(2):77–80, 2022.

[OK22b] Yu Omori and Keiji Kimura. Open-Source RISC-V Linux-Compatible NVMM Emulator. In *Sixth Workshop on Computer Architecture Research with RISC-V (CARRV 2022)*, 2022.

[PA17] Anastasios Petropoulos and Theodore Antonakopoulos. Hardware emulation of phase change memory. In *2017 Panhellenic Conference on Electronics and Telecommunications (PACET)*, pages 1–4, 2017.

[PX12] M. Poremba and Y. Xie. NVMain: An Architectural-Level Main Memory Simulator for Emerging Non-volatile Memories. In *2012 IEEE Computer Society Annual Symposium on VLSI*, pages 392–397, August 2012.

[PZX15] M. Poremba, T. Zhang, and Y. Xie. NVMain 2.0: A User-Friendly Memory Simulator to Model (Non-)Volatile Memory Systems. *IEEE Computer Architecture Letters*, 14(2):140–143, July 2015.

[RCPS07] Brian Rogers, Siddhartha Chhabra, Milos Prvulovic, and Yan Solihin. 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)*, pages 183–196, 2007.

[SAB15] Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. Trusted Execution Environment: What It is, and What It is Not. In *2015 IEEE Trustcom/BigDataSE/ISPA*, volume 1, pages 57–64, 2015.

[SiF19] SiFive. Freedom SoC. https://github.com/sifive/freedom, 2019.

[SJH+96] Tatsumi Sumi, Yuji Judai, Kanji Hirano, Toyoji Ito, Takumi Mikawa, Masato Takeo, Masamichi Azuma, Shin ichiro Hayashi, Yasuhiro Uemoto, Koji Arita, Toru Nasu, Yoshihisa Nagano, Atsuo Inoue, Akihiro Matsuda, Eiji Fuji, Yasuhiro Shimada, and Tatsuo Otsuki. Ferroelectric Nonvolatile Memory Technology and Its Applications. *Japanese Journal of Applied Physics*, 35(Part 1, No. 2B):1516–1520, feb 1996.

[SNR+18] Gururaj Saileshwar, Prashant J. Nair, Prakash Ramrakhyani, Wendy Elsasser, Jose A. Joao, and Moinuddin K. Qureshi. Morphable Counters: Enabling Compact Integrity Trees For Low-Overhead Secure Memories. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 416–427, 2018.

[Sta] Standard Performance Evaluation Corporation. SPEC CPU® 2017. https://www.spec.org/cpu2017/.

[SWW71] Teng Hsu Sheng and Zhuang Wei-Wei. ELECTRICALLY PROGRAMMABLE RESISTANCE CROSS POINT MEMORY, U.S. Patent 6531371.

[VMCL15] Haris Volos, Guilherme Magalhaes, Ludmila Cherkasova, and Jun Li. Quartz: A lightweight performance emulator for persistent memory software. In *Proceedings of the 16th Annual Middleware Conference*, Middleware '15, pages 37–49, New York, NY, USA, 2015. ACM.

[WA19] Andrew Waterman and Krste Asanović. The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Document Version 20191213. Technical report, RISC-V Foundation, dec 2019.

[WAH21] Andrew Waterman, Krste Asanović, and John Hauser. The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Document Version 20211203. Technical report, RISC-V Foundation, dec 2021.

[WC81] MARK N. WEGMAN and J. LAWRENCE CARTER. New Hash Functions and Their Use in Authentication and Set Equality. *Journal of Computer and System Sciences*, 22(3):265–279, 1981.

[Web18] Mark Webb. Markets for 3D-Xpoint. In *Flash Memory Summit 2018*, 2018.

[WLY+20] Zixuan Wang, Xiao Liu, Jian Yang, Theodore Michailidis, Steven Swanson, and Jishen Zhao. Characterizing and Modeling Non-Volatile Memory Systems. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 496–508, 2020.

[WM95] Wm. A. Wulf and Sally A. McKee. Hitting the Memory Wall: Implications of the Obvious. *SIGARCH Comput. Archit. News*, 23(1):20–24, mar 1995.

[WRK+10] H. . P. Wong, S. Raoux, S. Kim, J. Liang, J. P. Reifenberg, B. Rajendran, M. Asheghi, and K. E. Goodson. Phase Change Memory. *Proceedings of the IEEE*, 98(12):2201–2227, Dec 2010.

[WW17] J. Wang and B. Wang. PCMSim: A Hybrid Memory System Simulator for the Cloud Storage. In *2017 Fifth International Conference on Advanced Cloud and Big Data (CBD)*, pages 81–86, August 2017.

[Xil] Xilinx. Xilinx/linux-xlnx: The official Linux kernel from Xilinx. https://github.com/Xilinx/linux-xlnx.

[YHA18] Mao Ye, Clayton Hughes, and Amro Awad. Osiris: A Low-Cost Mechanism to Enable Restoration of Secure Non-Volatile Memories. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 403–415, 2018.

[ZA19] Kazi Abu Zubair and Amro Awad. Anubis: Ultra-Low Overhead and Recovery Time for Secure Non-Volatile Memories. In *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, pages 157–168, 2019.

[ZLWD17] Guoliang Zhu, Kai Lu, Xiaoping Wang, and Yong Dong. Building Emulation Framework for Non-volatile Memory. In *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 330–333, 2017.

Achievement

Academic Papers

◯ Yu Omori and Keiji Kimura, "Open-Source Hardware Memory Protection Engine Integrated With NVMM Simulator", *IEEE Computer Architecture Letters*, Vol.21, No.2, pp.77-80, 2022, Available: https://doi.org/10.1109/LCA.2022.3197777

◯ Yu Omori and Keiji Kimura, "Non-Volatile Main Memory Emulator for Embedded Systems Employing Three NVMM Behaviour Models", *IEICE Transactions on Information and Systems*, Vol.E104-D, No.5, pp.697-708, 2021, Available: https://doi.org/10.1587/transinf.2020EDP7092

International Conferences

◯ Yu Omori and Keiji Kimura, "Open-Source RISC-V Linux-Compatible NVMM Emulator", *Sixth Workshop on Computer Architecture Research with RISC-V (CARRV 2022)*, co-located with ISCA'22, New York City, USA, Jun. 2022.

◯ Yu Omori and Keiji Kimura, "Performance Evaluation on NVMM Emulator Employing Fine-Grain Delay Injection", *2019 IEEE Non-Volatile Memory Systems and Applications Symposium (NVMSA)*, pp.1-6, Hangzhou, China, Aug. 2019, Available: https://doi.org/10.1109/NVMSA.2019.8863522

Domestic Workshop

Lena Yu, Yu Omori and Keiji Kimura, "Prototype Implementation of Non-Volatile Memory Support for RISC-V Keystone Enclave", *SWoPP2021: Parallel, Distributed and Cooperative Processing Systems and Dependable Computing*, vol.121, No.116, pp.7-12, Jul. 2021

大森 侑, 木村 啓二, "Linux が動作可能な RISC-V NVMM エミュレータの実装", 情報処理学会 第 *236* 回システム・アーキテクチャ・第 *194* 回システムと *LSI* の設計技術・第 *56* 回組込みシステム合同研究発表会（*ETNET2021*）, Vol.2021-ARC-244, No.1, pp.1-10, Mar. 2021.

林 知輝, 大森 侑, 木村 啓二, "整合性ツリーおよび暗号化機構を持つ不揮発性メインメモリエミュレータの実装", 情報処理学会 第 *236* 回システム・アーキテクチャ・第 *194* 回システムと *LSI* の設計技術・第 *56* 回組込みシステム合同研究発表会（*ETNET2021*）, Vol.2021-ARC-244, No.2, pp.1-8, Mar. 2021.

大森 侑, 木村 啓二, "不揮発性メインメモリエミュレータの評価", 情報処理学会 第 *227* 回システム・アーキテクチャ・第 *187* 回システムと *LSI* の設計技術・第 *50* 回組込みシステム合同研究発表会（*ETNET2019*）, Vol.2019-ARC-235, No.19, pp.1-8, Mar. 2019.

Award

大森 侑, "電子情報通信学会 CPSY 研究会優秀若手発表賞", 電子情報通信学会（*IEICE*）, May. 2021.

Software Developments During Ph.D

Yu OMORI, "OpenMPE: Open-Source Memory Protection Engine", Apr. 2022, Available: https://github.com/uyiromo/OpenMPE

Yu OMORI, "Freedom: DRAM/NVMM Heterogeneous Memory Simulator on RISC-V SoC (Xilinx VC707 Board)", Apr. 2022, Available: https://github.com/uyiromo/freedom

Yu OMORI, “nvmtest: DRAM/NVMM Heterogeneous Memory Simulator on ARM SoC (Xilinx ZC706 Board)”, Oct. 2019, Available: https://github.com/uyiromo/nvmtest