Open-Source RISC-V Linux-Compatible NVMM Emulator

Yu Omori
oy@kasahara.cs.waseda.ac.jp
Waseda University
Shinjuku-ku, Tokyo, Japan

Keiji Kimura
keiji@waseda.jp
Waseda University
Shinjuku-ku, Tokyo, Japan

ABSTRACT

Emerging byte-addressable Non-Volatile Main Memory (NVMM) is expected as a new class of memory device in computer architecture. A CPU access persistent data on NVMM simply by load, store, and cache eviction instructions without a rich file system and costly system calls for it. One of expected NVMM use cases is secure non-volatile storage for IoT edge devices using RISC-V Keystone TEE. For this purpose, optimization techniques for TEE with NVMM should be fully explored in terms of system software as well as hardware. However, NVMM simulators have a difficulty in system-wide performance analysis despite their flexibility. NVMM emulators are also difficult to evaluate a system considering NVMM access characteristics such as access locality with a soft RISC-V CPU. In this paper, we build a Keystone-compatible RISC-V NVMM emulator. We extend the existing NVMM emulation model to exploit access locality even on a slow soft CPU. Besides, we modify the cache flush instruction to allow user applications. The proposed emulation model is validated by a micro benchmark. Then, they are compared by using SPEC CPU 2017 benchmark. The result shows that only the proposed emulation model can capture the impact of access locality and r/w ratio, which are important factors to reduce NVMM latency.

CCS CONCEPTS

• **Hardware** → *Reconfigurable logic and FPGAs*; **Non-volatile memory**.

KEYWORDS

Non-Volatile Main Memory, Emulator, RISC-V

ACM Reference Format:
Yu Omori and Keiji Kimura. 2022. Open-Source RISC-V Linux-Compatible NVMM Emulator. In *Proceedings of Sixth Workshop on Computer Architecture Research with RISC-V (CARRV'22).* ACM, New York, NY, USA, 7 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Emerging Non-Volatile Main Memory (NVMM) is expected as a byte-addressable and non-volatile memory device. NVMM can realize higher density, and lower standby power consumption than DRAM. Besides, a CPU can access it like DRAM without a rich file system. Data persistency can be ensured by cache eviction operations instead of costly system calls. On the other hand, NVMM has longer latency than DRAM, especially for writes. For data persistency, additional cache evictions and memory barriers are required [17].

To fully exploit NVMM performance, whole system including hardware and software should be optimized for NVMM. This demands has not been satisfied due to the lack of a real NVMM. NVMM simulators [4, 15, 19, 20, 27] provide high flexibility and detailed performance analysis. On the other hand, they usually take a long time for system-wide evaluation. NVMM emulators [9, 10, 12, 13, 26] can evaluate whole system much faster than simulators at the sacrifice of flexibility. However, existing NVMM emulators cannot consider NVMM architecture. NVMM usually has an internal buffer to reduce latency. It must be considered in optimization for NVMM. TUNA v2.1 [13] solved it by the new NVMM emulation model. Despite the contribution, it was unclear which factors are important in optimization for NVMM. In addition, existing NVMM emulators did not focus on Intel Optane DC Persistent Memory (DCPMM) [14]. We developed the NVMM emulator that can emulate existing models and DCPMM, then revealed the important factors for NVMM [17].

One of expected NVMM usages is secure non-volatile storage in Trusted Execution Environment. RISC-V Keystone [11] is a promising solution for secure IoT edge devices. Despite high security, Keystone does not provide an access to auxiliary storage devices by itself due to its design. It instead uses a file system on an untrusted OS, for instance Linux, for persistent data accesses. On the other hand, NVMM enables persistent data accesses in Keystone. However, existing works are difficult to explore this possiblity. A real DCPMM is limited to specific Intel server CPUs. Existing NVMM emulation models assume that a CPU is as fast as memory system. They cannot be directly ported to RISC-V SoCs on an FPGA.

In this paper, we propose an open-source RISC-V NVMM emulator on an FPGA. The emulator is based on Freedom SoC U500 Dev Kit [23]. It is compatible with Linux and Keystone. We extended existing NVMM models [13, 17] to exploit access locality even on a slow soft CPU. The emulator has heterogeneous memory consisting of DRAM and NVMM. Researchers can investigate optimization techniques for Keystone TEE with NVMM. In addition, a cache flush instruction is required to ensure data persistency on NVMM [18]. The flush instruction in Rocket on Freedom SoC is available only in M-mode. We slightly modified the instruction so that a user application can call it directly. Then, we validated the proposed NVMM emulation models by using the micro benchmark, and confirmed the effectiveness of them by using SPEC CPU 2017 benchmark.

Our contributions are summarized as follows:

CARRV'22, June 18–22,2022, Co-located with ISCA 2022

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

- We built a RISC-V NVMM emulator [1]. It has the extended NVMM emulation model for a soft CPU. It also has the DCPMM emulation model.
- We made the cache flush instruction available in U-mode. The original implementation allows only M-mode.
- We validated the models by using the micro benchmark. The result showed that only the proposed model can capture the impact of access locality. Besides, the DCPMM model can emulate behavior of a real DCPMM.
- We confirmed the effectiveness of the proposed model. The result showed that only the proposed model can capture the important factors for NVMM: access locality and r/w ratio.

2 RELATED WORKS

2.1 NVMM Simulators and Emulators

Gem5, NVMain, PCMSim, HMMSim [4, 15, 19, 20, 27] are software simulators that enables system simulation having NVMM. They have been widely used thanks to their detailed performance analysis and high flexibility, while suffering from about three to five orders of magnitude longer execution time than a real hardware. Thus, the system-wide performance analysis including OS on them is difficult task.

TUNA [12, 13], Quartz [26], and write-back aware Quartz [9, 10] are NVMM emulators. They emulate NVMM performance by injecting additional latency to DRAM accesses. Emulators can evaluate large workloads on whole system at the speed of a base hardware, however, existing emulators have issues of NVMM emulation. Quartz based emulators [9, 10, 26] cannot consider NVMM micro architectures due to their design. TUNA [12, 13] is the hardware emulator on an FPGA. TUNA v2.1 [13] solved the issue by introducing the new NVMM emulation model. Despite their contributions, effectiveness of the model on a system with NVMM was not fully discussed. Therefore, we implemented the NVMM emulator [17] based on TUNA v2.1 [13], and confirmed the effectiveness of the emulation model. Then, we revealed the important factors for a system with NVMM by analyzing an impact of memory access characteristics on application performance. Besides, we introduced the new NVMM emulation technique for DCPMM (Intel Optane DC Persistent Memory) [14] and validated its behavior.

Despite existing emulators' contributions, they do not target to RISC-V. Quartz-based emulators require specific Intel CPUs. TUNA-based emulators assumes that a hardware implemented CPU is sufficiently faster than its memory subsystem. To investigate a RISC-V system with NVMM, CPU and memory system should be customizable. A soft RISC-V CPU on an FPGA is usually slower than its memory system. An existing RISC-V NVMM emulator [21] uses the TUNA-based emulation model. In the paper, we propose a new NVMM emulation model that can be applied to a RISC-V SoC on an FPGA. It can consider NVMM micro architecture even on a soft CPU.

2.2 RISC-V Boards and SoCs

NVMM emulation technique proposed in [17] requires modification of a memory controller to inject additional latency. It cannot be ported to RISC-V boards: [3, 16, 24], Some customizable SoC have been also released: Rocket/BOOM [1], Freedom SoC [23], Shakti [22] and Ariane [29]. They have a RISC-V core IP and some peripheral IPs. The SoCs can be implemented on an FPGA according various demands. In the paper, we used Freedom SoC as a base RISC-V SoC considering customizability and Keystone requirements [7]. We implemented NVMM performance emulation models on the SoC.

[1] https://github.com/uyiromo/freedom

3 NVMM EMULATION TECHNIQUES

This section introduces overviews of four NVMM emulation behavior models: coarse-grain [12], fine-grain [13, 17], new extended fine-grain, and DCPMM [17]. The former three models are based on NVMM which architecture is similar to DRAM. The last one is based on a real DCPMM behavior.

3.1 Coarse-Grain Behavior Model

The coarse-grain behavior model is a simple NVMM model originally proposed in TUNA v1 [12]. It injects additional latency into DRAM accesses at a module on a memory bus. More specifically, it delays a handshake between an LLC and a memory controller. When a CPU misses LLC and issues a memory request, the module interrupt the request and start the timer. The handshake is blocked until the timer reaches the configured latency.

This model injects the same latency into every memory requests. Its simple emulation ignores NVMM micro architecture. For instance, DRAM latency depends on row buffer hit ratio, or access locality, at memory bus. Intel DCPMM also has an internal buffer [14]. The impact of access locality should be bigger on NVMM than DRAM. It must be considered in optimization for NVMM.

3.2 Fine-Grain Behavior Model

The fine-grain behavior model is a detailed NVMM model originally proposed in TUNA v2.1 [13]. It injects additional latency into timing parameters in a memory controller. It assumes that NVMM architecture and protocol are similar to those of DRAM. NVMM consists of banks, rows, and columns. Each bank has a row buffer which works as a write-back cache. NVMM module is mainly operated by three commands: Activate (ACT), Read/Write (R/W), and Precharge (PRE). For latency and endurance, PRE write back only dirty row buffers. *tRCD* is the minimal interval between ACT and R/W, and *tRP* is the minimal interval between PRE and ACT.

- ACT: Read data from memory cells to a row buffer
- R/W: Read data from or Write data into a row buffer
- PRE: Write back a row buffer to memory cells

According to the protocol, memory cells are read by only ACT, and written by only PRE. NVMM latency depends on memory cells, thus, read latency depends on *tRCD*, and write latency depands on *tRP*. The fine-grain behavior model injects additional latency into these parameters. When a memory request hits a row buffer, memory latency will be reduced since ACT is omitted. Latency of an internal buffer on NVMM can be adjusted by the coarse-grain model. This model injects different latency depending on memory requests. We confirmed that the fine-grain model can consider access locality on the NVMM emulator [17].

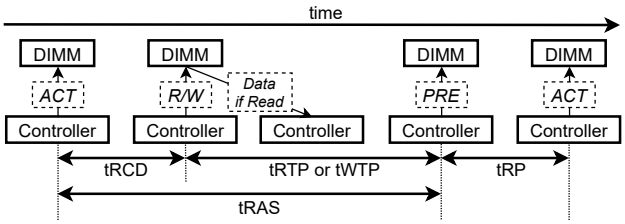


Figure 1: DDR3 Timing Parameters

3.3 Extended Fine-Grain Behavior Model

The extended fine-grain behavior model is an extension of the fine-grain one. The motivation of introducing it is the porting difficulty of the original fine-grain model to a system with a slow CPU. A memory controller issues implicit PRE if it receives no command within *tRTP*/*tWTP* from the preceding R/W. On 200MHz memory controller, they are 2CLK(10ns) and 3CLK(15ns) respectively by rounding up from the protocol values (Table 1). While existing emulators [12, 13, 17] use 667MHz hard CPU, the proposed emulator on an FPGA uses 50MHz soft Rocket CPU. The default *tRTP* and *tWTP* are less than 1 CPU-clock on the slow CPU and implicit PRE is issued. Thus, the buffer locality is spoiled.

Figure 1 depicts the DDR3 timing parameters [2] used in our emulator. PRE can be issued after *tRAS* from the preceding ACT. Similarly, PRE can be issued after *tRTP* or *tWTP* from the preceding READ or WRITE. The extended fine-grain model injects additional latency into these parameters. By setting *tRAS* long enough, successive commands can be issued before the implicit PRE. Figure 2 depicts an example. The original fine-grain model requires additional ACT for the request #2 due to the implicit PRE. In contrast, the extended fine-grain model can omit ACT for the request #2. It also needs to adjust *tRTP* and *tWTP*. Before issuing PRE, all timing constraints of *tRAS*, *tRTP*, and *tWTP* must be satisfied. In some cases, if $tRCD + tRTP/tWTP$ is grater than *tRAS*, a memory controller may ignore *tRAS* when issuing PRE. To ensure the timing constraint of *tRAS* strictly, we set *tRTP* and *tWTP* as below. Here, "(spec)" are specified values in the DDR3 protocol, and "(rest)" are the rest of configured *tRAS* when a R/W command is issued. By choosing a maximum of them, both the extended fine-grain model and the protocol requirement can be satisfied.

$$tRTP = max\{\, tRTP(spec),\ tRAS(rest)\,\}$$
$$tWTP = max\{\, tWTP(spec),\ tRAS(rest)\,\}$$

As described above, the extended fine-grain model adjusts *tRAS*, *tRTP* and *tWTP* in addition to *tRCD* and *tRP*. Table 1 shows their specified values in the DDR3 protocol [2]. Maximum values are not defined except for *tRAS*. The extended fine-grain model does not violate the protocol unless *tRAS* is set to extremely big value.

3.4 DCPMM Behavior Model

The DCPMM behavior model emulates a real Intel Optane DC Persistent Memory (DCPMM) [14]. DCPMM shows inconsistent performance characteristics [17, 28]. We analyzed DCPMM behavior in the literature [17], then modeled it as the DCPMM behavior model with some abstraction. On the model, memory read latency increases to 1.84×/2.16× when crossing 256-byte/4,096-byte boundaries, respectively. Memory write latency also increases to 1.90×/3.32× when crossing 256-byte/4,096-byte boundary, respectively.

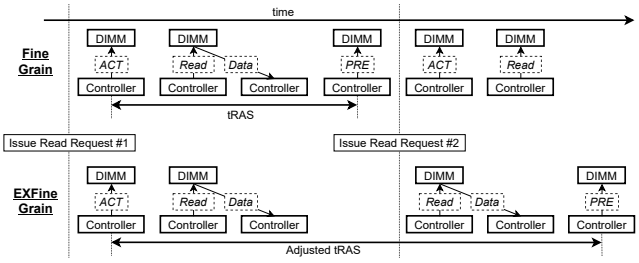


Figure 2: Behavior Comparison of the Fine-Grain Model and the Extended Fine-Grain Model

Table 1: DDR3-1600 Timing Parameters

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

Table 2: Specification of the Proposed NVMM Emulator

FPGA	Xilinx Virtex-7 FPGA VC707
Device	Virtex-7 XC7VX485T-2FFG1761
Rocket Core Spec.	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
Kernel/OS	GNU/Linux riscv64 5.6.0-dirty Debian GNU/Linux bullseye/sid

4 EMULATOR IMPLEMENTATION

4.1 Overview of NVMM Emulator

The proposed NVMM emulator is implemented on Freedom U500 VC707 FPGA Dev Kit [23] provided by SiFive. Table 2 shows the detailed specification. Freedom SoC has a Rocket core [1] employing RV64GC ISA. It has single-issue, 5-stage pipeline and in-order architecture. To build a Keystone-compatible NVMM emulator, we chose Freedom SoC following Keystone requirements [7].

We built the Linux kernel on the e448fa3 commit in the Keystone repository [6]. It is the extended Linux 5.6.0 for Keystone. We selected Debian bullseye/sid from Debian RISC-V Ports [5].

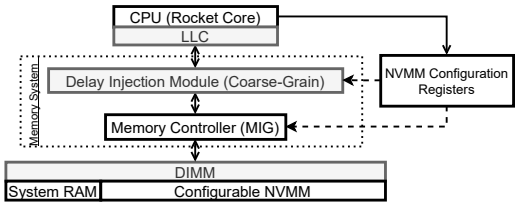


Figure 3: A Block Diagram of Memory System

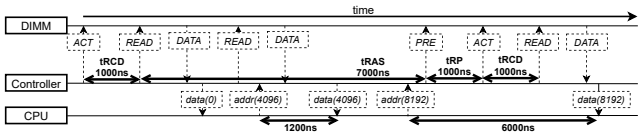


Figure 4: Detailed Behavior of the Extended Fine-Grain Model.
"addr(X)" is a read request to address X. Some latency are simplified for descriptions.

4.2 NVMM Emulation

Freedom SoC uses the MIG (Memory Interface Generator) IP provided by Xilinx as a memory controller. As shown in Table 2, CPUs are slow relative to memory system. Thus, the fine-grain model described in Section 3.2 cannot exploit access locality on this emulator. We implemented the coarse-grain, fine-grain, extended fine-grain, and DCPMM models described in Section 3 in the emulator for the comparison. They can be dynamically switched. The amount of additional latency can be configured via the MMIO registers.

Freedom SoC on VC707 FPGA has only one DIMM. We logically divided it to realize heterogeneous memory consisting of DRAM and NVMM. 1-GiB of DIMM is reserved as system RAM for Linux and Debian OS. The rest is configurable NVMM. The configurable NVMM must be explicitly and manually allocated by a dedicated API, like mmap. This can avoid the kernel's implicit memory allocation from NVMM area, which causes unintentional system performance degradation.

4.3 Cache Flush Operation

Cache eviction is required to ensure data persistency on NVMM. As of May 2022, RISC-V ISA standard does not define a cache flush instruction. On the other hand, the SiFive custom L1D$ flush instruction, CFLUSH.D.L1, is ported into the Rocket core. While it is also available on Freedom SoC, it can be used only in M-mode. A dedicated API to call the instruction from S/U-mode should cause large overhead.

We modified the implementation of CFLUSH.D.L1 to allow S/U-mode. This modification does not interfere with other instructions. An inline assembly of CFLUSH.D.L1 is shown below. If the zero register is specified, whole L1 D$ is flushed. Otherwise, only one line containing the specified virtual address by `reg` is flushed.

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

5 EXPERIMENTAL EVALUATION

This section validates the NVMM emulation models and confirms their effectiveness. We used the micro benchmark and SPEC CPU 2017 benchmark [25]. Through this section, we configure the NVMM emulation as below:

- coarse-grain: read+1,000ns and write+1,000ns
- fine-grain: *tRCP* = *tRP* = 1,000ns
- extended fine-grain: fine-grain + *tRAS* = 7,000ns

5.1 Validation of the Extended Fine-Grain model

This section validates the extended fine-grain model by comparing with the coarse- and the fine-grain models. The viewpoint is that three NVMM models can consider access locality or not. We used the micro benchmark shown in Algorithm 1. The *STRIDE* is set to 4,096 or 8,192. If *STRIDE* is smaller than row buffer size (8,192), average access latency will be reduced by locality.

Table 3a shows the result of read latency. Only the extended fine-grain model can capture the impact of *STRIDE*. However, the extended fine-grain model is only reduced to 67% when *STRIDE* is 4,096. It decreases to 50% in ideal, because row buffer hit ratio will be 50% when *STRIDE* is one half of row buffer size. Figure 4 depicts detailed analysis of the extended fine-grain model. A read request to address 0 activates the row buffer for address 0-8191. Then, the read request to address 4096 hits the row buffer. The latency is 1,200 ns when viewed from a CPU ("addr(4096)" to "data"). The next read request to address 8192 misses the row buffer. Although PRE and ACT are required to activate the new row buffer, *tRAS* must be satisfied. Thus, the latency of 8,192 becomes about 6,000 ns ("addr(8192)" to "data"). When *STRIDE* is 4,096, memory requests hit or miss a row buffer in turn. Average latency should be about $(1,200 + 6,000)/2 = 3,600$ ns. The result in Table 3a follows the expected behavior.

Table 3b shows the result of write latency. Unlike Table 3a, the fine-grain model and the extended fine-grain model show same trends. When a CPU has a write-back cache, the order of write requests viewed at the memory bus is not the same as what is viewed at the CPU. Access locality is to be lower than the expected. We measured that row buffer hit ratio is lower than 5% when $STRIDE = 4,096$. In addition, the fine-grain model shows longer latency than the coarse-grain one. This is due to total injected latency into a write request. The coarse-grain injects additional 1,000ns into a write request on the memory bus. In contrast, the fine-grain injects additional *tRCD (1,000ns)* and *tRP (1,000ns)* into a write request on the memory controller.

The experiment result above showed that the extended fine-grain model can exploit the access locality that are ignored on the fine-grain model. The extended fine-grain model cannot fully exploit access locality on a pure write benchmark as shown in Table 3b, however, real applications do not usually show such extreme high write/read ratio. Therefore, the extended fine-grain model can be thought to exploit access locality on real applications. Section 5.3 confirms the effectiveness of the extended fine-grain model using real benchmarks.

Algorithm 1 A Micro Benchmark for Access Locality

Parameters *STRIDE*: access stride in bytes
Input p: a pointer to the allocated NVMM region
Input *SIZE*: allocated size to p in bytes
  warm up TLB, then evict all cachelines
  start timer
  $addr \leftarrow p$
  while $addr < (p + SIZE)$ **do**
    read from / write into Mem[$addr$]
    $addr \leftarrow addr + STRIDE$
  end while
  stop timer
  divide "elapsed time" by "number of iterations"

Table 3: Average Latency while Changing *STRIDE*. (×N) is the normalized latency against 8192

(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

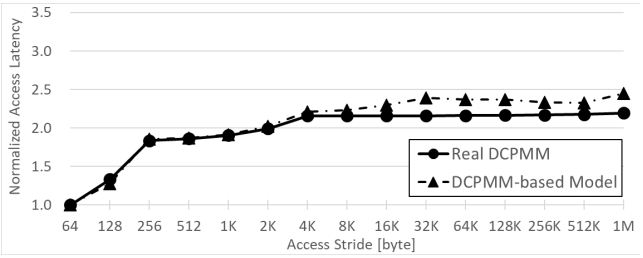
5.2 Validation of the DCPMM-based model

This section validates the DCPMM-based model by comparing it with a real DCPMM [17]. The viewpoint is that the DCPMM-based model shows the same trend of access latency as a real DCPMM while changing *STRIDE*. We also used the micro benchmark shown in Algorithm 1. The *STRIDE* was set from 64 to 1-MiB.

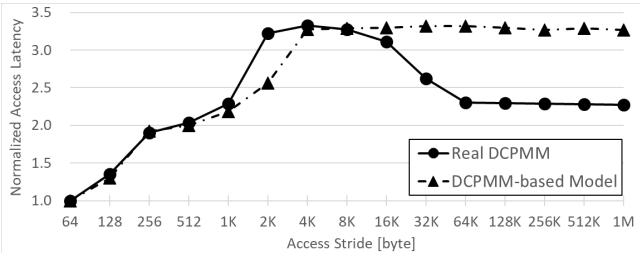
Figure 5 shows the result. The DCPMM-based model on the emulator shows the similar trend to a real DCPMM. However, in Figure 5b, the trend of the DCPMM-based model from 4K- is different from that of a real DCPMM. Its behavior should be caused by the DCPMM advanced controller [17]. Such advanced controllers are not expected for an edge-device in terms of cost, area, power, and so on. Thus, the write behavior from 4K- is abstracted on the DCPMM-based model.

5.3 Effectiveness of the Extended Fine-Grain Model on SPEC CPU 2017 Benchmark

This section confirms the effectiveness of the extended fine-grain model by using SPEC CPU 2017 benchmark [25]. We used 14 of 24 programs, which can be successfully compiled and executed on the emulator. All dynamic memory allocation in them were replaced with the modified jemalloc [8] to allocate memory from



(a) Read Latency



(b) Write Latency

Figure 5: Normalized Average Latency of a Real DCPMM and the DCPMM-based Model on the Emulator

configurable NVMM (Table 2). Additional latency is configured as described at the beginning of Section 5.

Figure 6 shows the results. Bar graphs are normalized execution time of the coarse-, fine-, and extended fine-grain against execution time. The line graph is memory access frequency measured on the memory bus in the emulator when executed on the system RAM. The programs are sorted in ascending order of the extended fine-grain model from left to right. If a program issues memory requests frequently, its execution time should be heavily affected by memory latency. Thus, the line graph is expected to increase from left to right following bars of the extended fine-grain model. Most of benchmarks follow this expectation, however, a few of them marked with squares are contrary to the expectation.

First, we focused on the programs marked with rounded squares (511.povray_r, 523.xalancbmk_r and 505.mcf_r). We measured access locality (i.e. row buffer hit ratio) in the memory controller of each program for detailed investigation. It was calculated from the number of issued *ACT*, and the number of accepted memory requests. The three programs showed high access locality. While the average was 0.18, their hit ratio were 0.52, 0.50, and 0.48, respectively. These exception are shown in only the extended fine-grain model. This result confirms the discussion in Section 5.1.

Second, we focused on the benchmarks marked with sharp squares (525.x264_r, 503.bwaves_r, and 508.namd_r). We measured r/w ratio of memory requests for detailed investigation. It is calculated by dividing the number of read requests issued to configurable NVMM on the memory bus by that of write requests. The three programs showed high r/w ratio (13.22, 5.44, and 5.40, respectively). According to Table 3a and Table 3b, emulated NVMM write latency is longer than read latency. If a benchmark is read intensive, total

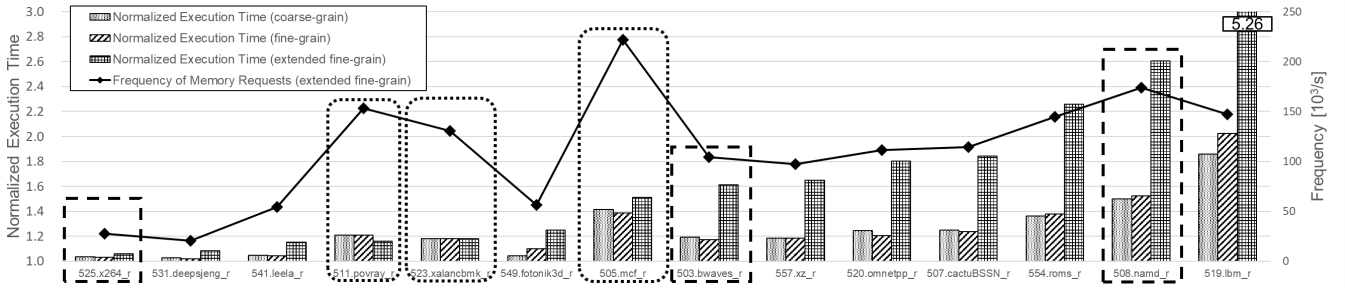


Figure 6: Normalized Execution Time of SPEC CPU 2017 Benchmark Programs. All results are normalized against the execution time when they are executed on DRAM. The left side vertical axis is for normalized execution time (bar graph). The right side vertical axis is for memory access frequency (line graph).

NVMM latency will be smaller. It is confirmed that the extended fine-grain model can capture the impact of r/w ratio.

The discussion above confirmed that only the extended fine-grain model can exploit memory access characteristics of programs. In some cases, access locality and r/w ratio exceed the impact of access frequency. They are important factors to reduce NVMM latency and performance degradation. Only the proposed extended fine-grain model can enables optimization by utilizing them.

6 CONCLUSION

In this paper, we implemented a RISC-V NVMM emulator on an FPGA using the Freedom U500 VC707 FPGA Dev Kit. We extended the fine-grain model as the extended fine-grain model to exploit access locality even on a slow soft CPU on an FPGA. In addition, we modified the Rocket core so that programs in S/U-mode can directly issue the cache flush instruction. It enables low overhead cache eviction from an user application. We also confirm that a Linux kernel and Debian OS works well on the emulator.

We validated the extended fine-grain model using the micro benchmark. The result showed that the extended fine-grain model can capture the impact of access locality that is ignored on the coarse- and fine-grain models. We also confirmed the effectiveness of the extended fine-grain model on real applications by using the SPEC CPU 2017 benchmark. It revealed that the impact of NVMM latency on execution time is mainly affected by memory access frequency, however, it can be reduced by access locality and r/w ratio. Only the extended fine-grain model can consider these two factors. Besides, we validated the DCPMM-based model by comparing with a real DCPMM.

We confirmed that the proposed NVMM emulator enables to fully investigate optimization techniques for NVMM even on a system with a slow CPU. The NVMM emulator has an important role as Keystone-compatible NVMM emulator.

ACKNOWLEDGMENTS

This work was partly executed under the cooperation of organization between Kioxia Corporation and Waseda University.

REFERENCES

[1] Asanović et al. 2016. *The Rocket Chip Generator.* Technical Report UCB/EECS-2016-17. EECS Department, University of California, Berkeley. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.html

[2] JEDEC SOLID STATE TECHNOLOGY ASSOCIATION. 2012. *DDR3 SDRAM Standard (Revision of JESD79-3E, July 2010).* Technical Report. JEDEC.

[3] beagleboard.org. 2021. Beagle V. https://beaglev.org/

[4] Bock et al. 2015. HMMSim: a simulator for hardware-software co-design of hybrid main memory. In *NVMSA'15.* IEEE, Hong Kong, China, 1–6. https://doi.org/10.1109/NVMSA.2015.7304374

[5] Debian.org. 2022. debian-ports. https://snapshot.debian.org/archive/debian-ports/

[6] Keystone Enclave. 2019. Keystone Enclave (QEMU + HiFive Unleashed). https://github.com/keystone-enclave/keystone

[7] Keystone Enclave. 2019. RISC-V Background. http://docs.keystone-enclave.org/en/latest/Getting-Started/How-Keystone-Works/RISC-V-Background.html

[8] jemalloc.net. 2015. jemalloc. https://github.com/jemalloc/jemalloc

[9] Koshiba et al. 2017. Towards write-back aware software emulator for non-volatile memory. In *NVMSA '17.* IEEE, Hsinchu, Taiwan, 1–6. https://doi.org/10.1109/NVMSA.2017.8064479

[10] Koshiba et al. 2019. A Software-based NVM Emulator Supporting Read/Write Asymmetric Latencies. *IEICE Transactions* 102-D, 12 (2019), 2377–2388. http://search.ieice.org/bin/summary.php?id=e102-d_12_2377

[11] Lee et al. 2020. Keystone: An Open Framework for Architecting Trusted Execution Environments. In *Eurosys '20.* Association for Computing Machinery, New York, NY, USA, Article 38, 16 pages. https://doi.org/10.1145/3342195.3387532

[12] Lee et al. 2014. FPGA-based prototyping systems for emerging memory technologies. In *2014 25nd IEEE International Symposium on Rapid System Prototyping.* IEEE, New Delhi, India, 115–120. https://doi.org/10.1109/RSP.2014.6966901

[13] T. Lee and S. Yoo. 2017. An FPGA-based platform for non volatile memory emulation. In *NVMSA '17.* IEEE, Hsinchu, Taiwan, 1–4. https://doi.org/10.1109/NVMSA.2017.8064466

[14] Lily Looi and Jianping Jane Xu. 2019. INTEL® OPTANE™ DATA CENTER PERSISTENT MEMORY. In *Hot Chips (HC) 31.* IEEE, Cupertino, CA, USA, 1–25. https://doi.org/10.1109/HOTCHIPS.2019.8875668

[15] Lowe-Power et al. 2020. The gem5 Simulator: Version 20.0+. https://doi.org/10.48550/ARXIV.2007.03152

[16] Microsemi. 2020. PolarFire SoC FPGA Icicle Kit. https://www.microsemi.com/existing-parts/parts/152514

[17] Yu Omori and Keiji Kimura. 2021. Non-Volatile Main Memory Emulator for Embedded Systems Employing Three NVMM Behaviour Models. *IEICE Trans. Inf. Syst.* 104-D, 5 (2021), 697–708.

[18] Steven Pelley, Peter M. Chen, and Thomas F. Wenisch. 2014. Memory Persistency. In *Proceeding of the 41st Annual International Symposium on Computer Architecuture* (Minneapolis, Minnesota, USA) *(ISCA '14).* IEEE Press, Piscataway, NJ, USA, 265–276. http://dl.acm.org/citation.cfm?id=2665671.2665712

[19] M. Poremba and Y. Xie. 2012. NVMain: An Architectural-Level Main Memory Simulator for Emerging Non-volatile Memories. In *2012 IEEE Computer Society Annual Symposium on VLSI.* IEEE, Amherst, MA, USA, 392–397. https://doi.org/10.1109/ISVLSI.2012.82

[20] M. Poremba, T. Zhang, and Y. Xie. 2015. NVMain 2.0: A User-Friendly Memory Simulator to Model (Non-)Volatile Memory Systems. *IEEE Computer Architecture Letters* 14, 2 (July 2015), 140–143. https://doi.org/10.1109/LCA.2015.2402435

[21] Ma Ruoheng. 2020. An RISCV Emulation Board for Non-Volatile Memory. (2020). Graduation Thesis at Fakultät für Informatik, Karlsruher Institut für Technologie.

[22] shakti.org. 2020. Shakti Processor. https://shakti.org.in/

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

[24] SiFive. 2020. HiFive Unmatched. https://www.sifive.com/boards/hifive-unmatched

[25] spec.org. 2017. SPEC CPU(R) 2017. https://www.spec.org/cpu2017/

[26] Volos et al. 2015. Quartz: A Lightweight Performance Emulator for Persistent Memory Software. In *Middleware '15*. ACM, New York, NY, USA, 37–49. https://doi.org/10.1145/2814576.2814806

[27] J. Wang and B. Wang. 2017. PCMSim: A Hybrid Memory System Simulator for the Cloud Storage. In *CBD '17*. IEEE, Shanghai, China, 81–86. https://doi.org/10.1109/CBD.2017.22

[28] Wang et al. 2020. Characterizing and Modeling Non-Volatile Memory Systems. In *MICRO '20*. IEEE, Athens, Greece, 496–508. https://doi.org/10.1109/MICRO50266.2020.00049

[29] F. Zaruba and L. Benini. 2019. The Cost of Application-Class Processing: Energy and Performance Analysis of a Linux-Ready 1.7-GHz 64-Bit RISC-V Core in 22-nm FDSOI Technology. *IEEE Trans. on VLSI Systems* 27, 11 (Nov 2019), 2629–2640. https://doi.org/10.1109/TVLSI.2019.2926114