NVMulator: A Configurable Open-Source Non-volatile Memory Emulator for FPGAs

Sajjad Tamimi[1](✉), Arthur Bernhardt[2], Florian Stock[1], Ilia Petrov[2], and Andreas Koch[1]

[1] Embedded Systems and Applications Group, Technical University of Darmstadt, Hochschulstr. 10, 64289 Darmstadt, Germany
{tamimi,stock,koch}@esa.tu-darmstadt.de

[2] Data Management Lab, Reutlingen University, Alteburgstr. 150, 72762 Reutlingen, Germany
{arthur.bernhardt,ilia.petrov}@reutlingen-university.de

Abstract. Near-Data Processing (NDP) is a key computing paradigm for reducing the ever growing time and energy costs of data transport versus computations. With their flexibility, FPGAs are an especially suitable compute element for NDP scenarios. Even more promising is the exploitation of novel and future *non-volatile memory* (NVM) technologies for NDP, which aim to achieve DRAM-like latencies and throughputs, while providing large capacity non-volatile storage.

Experimentation in using FPGAs in such NVM-NDP scenarios has been hindered, though, by the fact that the NVM devices/FPGA boards are still very rare and/or expensive. It thus becomes useful to *emulate* the access characteristics of current and future NVMs using off-the-shelf DRAMs. If such emulation is sufficiently accurate, the resulting FPGA-based NDP computing elements can be used for actual full-stack hardware/software benchmarking, e.g., when employed to accelerate a database.

For this use, we present *NVMulator* [7], an open-source easy-to-use hardware emulation module that can be seamlessly inserted between the NDP processing elements on the FPGA and a conventional DRAM-based memory system. We demonstrate that, with suitable parametrization, the emulated NVM can come very close to the performance characteristics of *actual* NVM technologies, specifically Intel Optane. We achieve 0.62% and 1.7% accuracy for cache line sized accesses for read and write operations, while utilizing only 0.54% of LUT logic resources on a Xilinx/AMD AU280 UltraScale+ FPGA board. We consider both file-system as well as database access patterns, examining the operation of the RocksDB database when running on real or emulated Optane-technology memories.

Keywords: FPGA · Non-volatile memory · Emulator

1 Introduction

With the ever-growing volume of data generated due to the success of the Internet, Near-Data Processing (NDP) offers a promising solution to improve overall system performance and optimize bandwidth usage. NDP achieves this by facilitating more direct access to the actual storage devices for suitable compute elements, enabling both greater throughput as well as shorter latencies. FPGAs serve as an ideal compute element for NDP applications for two primary factors. First, they can offer wide I/Os for direct connections to DRAM banks and maintain a high level of fine-grained parallelism to manage multiple in-flight requests from various Flash banks. Second, their inherent flexibility allows users to readily implement their ideas and adapt to evolving requirements. Moreover, the efficacy of NDP is more promising by the potential integration of cutting-edge non-volatile memory (NVM) technologies. These innovative solutions offer high-density storage, DRAM-comparable performance, and significant capacity for persistent storage. As a result, the utilization of NDP in conjunction with advanced NVM technologies holds significant promise for addressing the challenges posed by ever-increasing data generation.

Utilizing FPGAs in NDP-capable systems based on NVM remains a challenge, though, as NVM devices themselves or suitably equipped FPGA boards are still very rare and/or expensive. To address this challenge, prior research has proposed either software simulation or hardware emulation environments. Software-based approaches utilize standalone simulators or those integrated with DDR systems to model NVM behavior, facilitating the assessment of software-based solutions [3,19]. On the other hand, hardware-based techniques propose the emulation of NVM characteristics by leveraging commercially available DDR on FPGAs. These methods introduce additional latency between read and write accesses during the request handshake process [14] or alter the memory parameters of the memory controller [16]. Despite their potential, these solutions necessitate specialized devices that are also rare or expensive. Moreover, existing research lacks a user-friendly NVM emulation environment that is compatible with various FPGA platforms.

To allow research to push ahead here, we propose an open-source, user-friendly Non-Volatile-Memory emulator (referred to as NVMulator) that can accurately replicate the access characteristics of present and future NVM technologies utilizing commercially available DRAM components. The NVMulator is designed as a hardware emulation module that can be seamlessly integrated between the NDP processing unit on an FPGA and conventional DRAM-based memory system. We demonstrate the application of the NVMulator within the context of the reconfigurable computing framework Task Parallel System Composer (TaPaSCo), which is an open-source platform offering both hardware and software stacks for FPGAs for users, aimed at users without specialized knowledge in the field. The objective of this work is to provide NVM access latencies by creating a fully-functional and *easily usable* emulator.

In order to assess the efficiency of the proposed approach, we conducted a comparative analysis with existing NVM technologies, specifically Intel Optane, utilizing a Xilinx/AMD AU280 UltraScale+ FPGA board. Our evaluation was carried out through two distinct methods. Initially, we employed a file-system benchmarking tool (i.e., the FIO tool) to demonstrate the accuracy of the emulator in the context of real-world workloads. Subsequently, we implemented on the FPGAs an emulated non-volatile persistent storage for a host running RocksDB. The findings indicate that not only can the emulator successfully emulate Intel Optane within real-world applications, but it can also be seamlessly deployed as emulated persistent storage with minimal effort. Note that our emulation only considers timing behavior. Lower level characteristics such as reliability (wear) and error rates are not covered, as they were out-of-scope for our use-cases.

In the remaining of the paper, first we give an overview of off-the-shelf NVM technology as well as a discussion of related work in Sect. 2. Then, we present NVMulator in Sect. 3.1 and we show how it is integrated in the TaPaSCo platform in Sect. 3.2. Afterwards we evaluate the accuracy of the proposed approach by comparison to OptaneDC memory in Sect. 4. We close with a conclusion in Sect. 5.

2 Storage Technologies and Related Work

In this section we begin with an overview of NVM storage technologies and the proposed NDP compute units. Afterwards, we discuss previous studies that have aimed to emulate the characteristics of NVM.

2.1 NVM Storage Technologies

Due to their inherent characteristic, exploiting NVM as a storage system has been often proposed by previous research in different domains, ranging from embedded applications to data-centric applications. The properties of newer NVM differ from those of conventional storage technologies such as Flash or DRAM in the following ways:

Byte-Addressability. Newer NVM technologies are byte-addressable, similar to DRAM, and do not rely on page-block accesses like Flash. This characteristic presents new opportunities for NDP approaches that wish to utilize it, however, current algorithmic approaches have not yet succeeded in fully utilizing byte-level reads/writes on NVM.

Read/Write Latency. NVM technologies have higher access latency compared to DRAM. For instance, Phase-Change Memory (PCM) has higher read/write latency than DDR (though still comparable) but significantly lower latency than NAND [6]. Additionally, read and write latency in NVM exhibit asymmetry, characterized by low-latency read operations and slower write operations.

Non-volatile. Modern NVM technologies such as PCM and Spin-Transfer Torque Memory (STT-RAM) are non-volatile. This means that data is preserved intact even when the memory is powered off or in the event of memory or power failure.

Endurance. NVM technologies are wear-prone and face endurance challenges. While NVMs have higher endurance compared to NAND Flash [6,9], implementing wear-leveling strategies is essential to enhance their longevity.

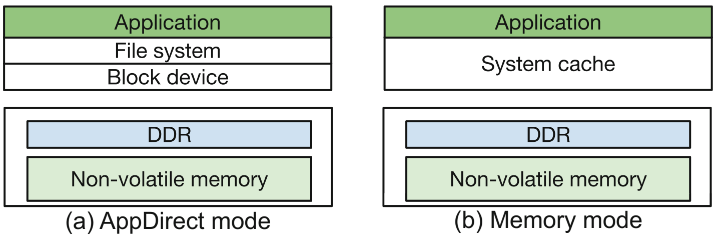

Fig. 1. OptaneDC in (a) AppDirect and (b) Memory modes [18].

Intel Optane DC. Intel® Optane™ DC Persistent Memory module (short: OptaneDC) is an innovative and widely recognized technology developed by Intel Corporation that has captured the attention of users. The OptaneDC technology offers two configuration modes: Memory mode (cached) and AppDirect mode (uncached) [18]. In Memory mode, users can extend the system memory capacity, effectively increasing the overall performance and efficiency of the system. On the other hand, AppDirect mode enables users to leverage the module as persistent storage. Figure 1 shows the integration process of the OptaneDC memory module into a system in these two modes. The primary objective of this study is to examine and replicate the access latency of NVM when employed as persistent storage.

NDP Compute Units. Prior research has explored various types of compute units for Near-Data Processing (NDP) that can be classified into several categories. In the first category, studies such as [4,12] have proposed using existing embedded hard-core processors, such as multi-core ARM processors, for query processing on Samsung's smart-SSD devices. These processors are typically employed for controlling storage modules and managing wear-leveling to enhance flash endurance.

In the second category, researchers have suggested FPGAs as NDP compute units. [20] introduced an approach that connects an FPGA to the NVMe flash via a PCIe interface, allowing for data processing in close proximity to the storage. However, this method is limited by the bandwidth of the PCIe connection and only enables processing close to, rather than near, the storage. [22] proposed offloading operations from the query engine in the host to an embedded ARM core and FPGA on the COSMOS+ board [17]. However, the underlying storage in these approaches relies on NAND Flash. Generally, FPGA boards with NVM devices scarce (often just prototypes) and/or expensive, posing challenges for wider use.

2.2 Related Work

Prior research efforts to emulate NVM behavior can be classified into two categories: software-based and hardware-based solutions. In the following, we will examine both of these categories.

Software-Based Solution. In the first category of research, prior studies have explored the utilization of software to emulate the behavior of NVM as either a standalone simulator or in conjunction with DDR systems. NVMain [19] is a cycle-accurate main memory software simulator that effectively models NVM architectural behavior, including aspects such as power consumption and performance. The simulator facilitates seamless integration with the gem5 simulator, providing a familiar user interface and simplifying the overall experimentation process. HMMSim [5] is an NVM software simulator that incorporates hybrid main memory, comprising both NVM and DDR, to emulate various memory architectures, such as NVM. By offering an Application Programming Interface (API), this work facilitates the evaluation of software solutions for managing hybrid main memory systems. HME [5] is an NVM emulator that simulates remote NVM on Non-Uniform Memory Access (NUMA) nodes by introducing software delays to remote memory accesses. However, these solutions are solely software-based and do not provide users with the ability to employ them as persistent storage. Moreover, the execution time of these software emulations is considerably longer compared to actual hardware implementations.

Hardware-Based Solution. In the second category of related research, prior studies have focused on emulating NVM on FPGAs. TUNA [14] proposed adding a module between the memory controller and processing element within the design, which would enable NVM behavior emulation over existing off-chip DDR memory on FPGA devices. This proposed module introduces additional latency to handshake signals by applying a fixed delay for read and writes accesses. While this approach is suitable for handling large data chunks, it lacks accuracy due to its disregard for memory bank parallelism in the overlaid DDR and NVM systems. To address this limitation, authors in [15] [16] proposed introducing fine-grained latency to memory parameters, such as tRCD or tRP, to more accurately mimic the underlying memory controller. However, modifying the memory controller proves to be a challenging undertaking, particularly as it often involves working with proprietary technology like the Memory Interface Generator (MIG) IP provided by AMD/Xilinx.

While the existing body of research offers a foundation for further exploration, to the best of our knowledge, no current solutions enable designers to easily utilize NVM emulators across various FPGA platforms without the need for specialized hardware. While internal FPGA prototype boards using actual NVMs, such as PCM coupled to an AMD/Xilinx Virtex 7 FPGA, do exist, they are not available to most researchers and often have hardware limitations that make them unsuitable as a general-purpose platform.

3 Proposed Approach

We start by introducing the NVMulator micro-architecture and discussing the proposed approach for emulating NVM characteristics. Then, we show the integration process of the NVMulator within the TaPaSCo FPGA computing framework. By integrating the NVMulator into a framework, instead of just providing

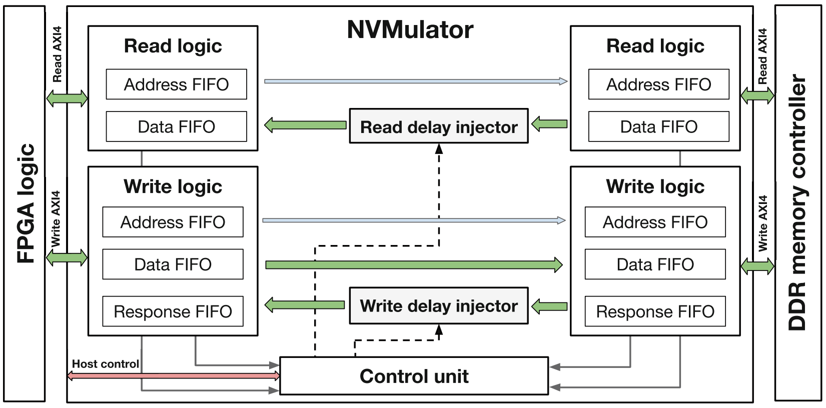

Fig. 2. NVMulator!t micro-architecture.

a stand-alone hardware module as has been done in prior work, we can leverage all of the existing TaPaSCo functionality and design automation to enable productive research into NDP with NVM more quickly.

3.1 NVMulator Micro-Architecture

Figure 2 shows the micro-architecture of NVMulator, which is designed to emulate read/write access latencies. This module is equipped with an AXI4-Slave interface that receives read/write requests from the FPGA logic, an AXI4-Master interface that forwards these requests to the DDR memory controller, and a controlling interface. For read requests, the module injects latency into the read response, in accordance with the host-specified latency. Similarly, for write requests, the module adds an additional delay to the write response, as determined by the host. The controlling interface allows the host to manage the NVMulator functionalities during operation without requiring the stop of execution or re-synthesis of the design for simple timing parameter changes. The NVMulator enables the emulation of access latency for emerging NVM technologies, such as PCM and Spin-Transfer Torque Memory (STT-RAM).

Timing Model. Equation 1 shows the timing model employed in the NVMulator. Upon receiving a read/write request (r/w) on the address channels, the NVMulator module forwards this request to the DDR memory controller and records the request receipt ($\text{Time}_{\text{received}}$). Once the response from the DDR is obtained, NVMulator logs the completion time of the request($\text{Time}_{\text{completed}}$). Subsequently, the control unit computes the expected latency for the request, taking into consideration the burst size (Burst_size) and the host-specified delays per beat for read and write requests ($\text{Delay}_{\text{beat}}$). Then, the control unit determines the delay to inject ($\text{Delay}_{\text{inject}}$), which represents the differ-

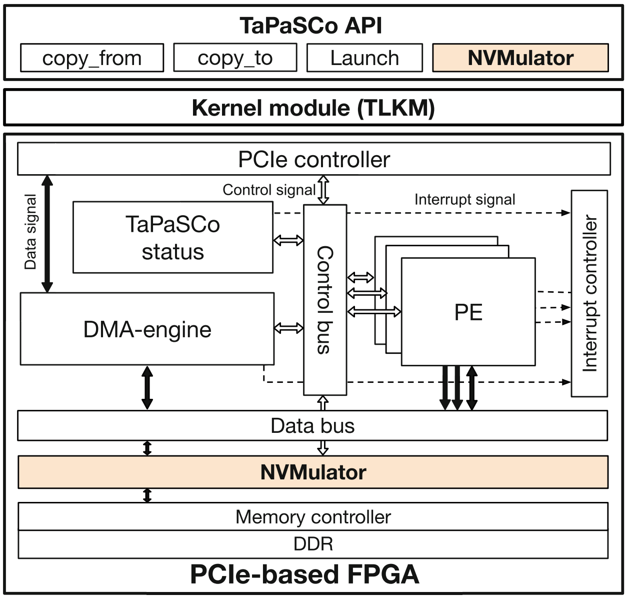

Fig. 3. Integration of NVMulator module into the software and hardware stacks of the TaPaSCo framework

ence between the expected time for the request and the actual response time from the DDR memory.

$$\begin{aligned}
\mathrm{Delay}_{\mathrm{NVM}}(\mathrm{r/w}) &= \mathrm{Burst\_size}(\mathrm{r/w}) + \mathrm{Delay}_{\mathrm{beat}}(\mathrm{r/w}) \\
\mathrm{Delay}_{\mathrm{DDR}}(\mathrm{r/w}) &= \mathrm{Time}_{\mathrm{completed}}(\mathrm{r/w}) - \mathrm{Time}_{\mathrm{received}}(\mathrm{r/w}) \\
\mathrm{Delay}_{\mathrm{inject}}(\mathrm{r/w}) &= \mathrm{Delay}_{\mathrm{NVM}}(\mathrm{r/w}) - \mathrm{Delay}_{\mathrm{DDR}}(\mathrm{r/w})
\end{aligned} \tag{1}$$

3.2 TaPaSCo Integration

TaPaSCo is an open-source hardware/software framework [13] that facilitates the integration of FPGA-based accelerators into heterogeneous computing systems. The goal of TaPaSCo is to provide a flexible and scalable framework for both expert and non-expert FPGA users. It supports a wide range of FPGA platforms, including high-performance PCIe-based platforms with large FPGA devices, such as data center AU280 and AU250 FPGA or Versal cards, small embedded Zynq platforms like Xilinx ZC702 and ZC706, and Amazon AWS F1 instances with FPGA accelerators.

TaPaSCo comprises an automated System-on-Chip (SoC) design generator and an Application Programming Interface (API). The SoC design generator enables the construction of a pool of Processing Elements (PE) with the peripheral module to interface with user applications. The API provides a software interface for easily controlling the implemented accelerators. Users can implement their custom hardware accelerators (i.e., PEs) and seamlessly use them across various supported platforms. TaPaSCo also provides optional functionality through plugins, referred to as *features*, which may not be supported by all platforms. These features are configurations that must be set during the design-building process. For example, users can easily enable the use of High Bandwidth Memory (HBM) RAM interfaces or network interfaces on supported devices. In this work, we aim to extend TaPaSCo with an NVMulator as a feature in PCIe-based platforms. In the following, we will explain the integration of the emulator into the TaPaSCo framework in both the hardware and software stack.

Hardware-Side. Figure 3 shows the SoC architecture of TaPaSCo on a PCIe-based FPGA platform that includes a PCIe controller, memory controller, TaPaSCo status module, DMA-engine, control and data buses, interrupt controller, and pool of PEs. The PCIe controller provides communication between the hardware and host via the PCIe bus. The memory controller, which utilizes the AMD/Xilinx Memory Interface Generator (MIG) IP core, provides an interface to the off-chip device memory, such as DDR. The TaPaSCo status module is responsible for storing hardware information, including mapped addresses, and the DMA-engine manages PCIe-DMA transfers between the host memory and device memory. The control bus enables the host to control FPGA modules and the data bus enables PEs and DMA-engine to access device memory. The interrupt handler collects signals raised by the hardware modules and forwards them to the host API of TaPaSCo. PEs are custom hardware accelerators, which can be implemented in either Hardware Description Language (HDL) or by High-Level Synthesis (HLS), and are responsible for executing applications on the hardware. It is essential to design and implement PEs with a so-called *T-shape* architecture compatible with TaPaSCo, featuring three interfaces: control, interrupt, and data interfaces. The control and interrupt interfaces enable the host to manage the PE, while the data interface provides PE access to off-chip memory. As shown in Fig. 3, the NVMulator has been integrated into the TaPaSCo SoC. To provide runtime control of the NVMulator by the host, the module's controlling interface is connected to the control bus, which allows the host to manage it via the software interface of TaPaSCo. To enable the module within the design, it is necessary to include --features 'NVMmulator enable: true' flag during the TaPaSCo building process as follows:

```
tapasco compose [PE x 1] @ 100MHz
                        --platforms AU280
                        --features "NVMulator {enable: true}"
```

Software-Side. The TaPaSCo software interface consists of an API and kernel module. The API is based on a task-parallel model, which involves decomposing computations into discrete tasks that can be independently executed on hardware accelerators. The kernel module, known as the TaPaSCo Loadable Kernel Module (TLKM), communicates with the device using ioctl commands. As shown in Fig. 3, the nvMulator function has been integrated into the software layer hierarchy. When the API call invokes this function, it triggers the corresponding functions in the kernel, allowing users to modify the control registers of the NVMulator in the hardware. This approach enables on-the-fly configuration of the NVMulator, eliminating the need for design alterations to accommodate various types of NVM during experimentation. Figure 4 presents a C++ example utilizing the nvMulator() function call within the user program. This function call requires three parameters: read_latency and write_latency in a number of clock cycles, and an NVM_mode that accepts values of either 0 or 1 to disable or enable the NVMulator functionality.

```cpp
#include <tapasco.hpp>

#define PE_ID       7
#define READ_DELAY  100  // in clock cycles
#define WRITE_DELAY 200  // in clock cycles
#define NVM_MODE    1    // enable NVM emulation mode

int main() {
    tapasco::Tapasco tapasco;

    tapasco.nvMulator(READ_DELAY, WRITE_DELAY, NVM_MODE);

    auto job = tapasco.launch(PE_ID, reg1, reg2, ...);
    job();

    return 0;
}
```

Fig. 4. Example of using the nvMulator() function call in the C++ API of TaPaSCo

4 Experimental Setup and Evaluation

To evaluate the accuracy of the emulator, we built a TaPaSCo design on an AMD/Xilinx Alveo U280 FPGA card, which is connected to a host through a PCIe Gen3 interface with 16x lanes. The host is an ARM Neoverse N1 System Development Platform (N1-SDP). Furthermore, we have developed a rudimentary block device driver to facilitate the utilization of the FPGA as a storage device. Throughout the experiments conducted in this study, we have compared the measured latency and throughput values to the reported values for OptaneDC, as presented in [11].

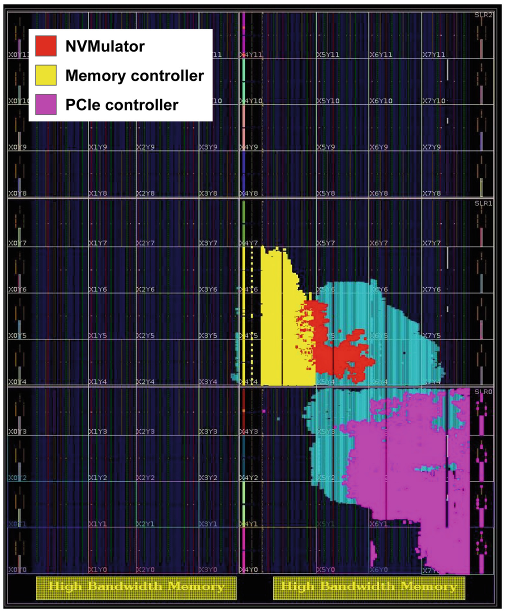

Fig. 5. Layout of the NVM storage built on the AU280.

Figure 5 shows the floorplan of the AU280, which includes the PCIe controller, MIG, TaPaSCo related modules, and NVMulator. This design is synthesized using Xilinx Vitis 2022.2. As reported in Table 1, the NVMulator occupies less than 0.6% of the available resources and logic resources. Thus, the remaining portion of the design provides ample space for the integration of the actual NDP processing units.

In the following, we evaluate the effectiveness of the proposed methodology by measuring the on-device latency and its performance as a persistent storage when executing real-world applications. As actual Optane devices are no longer commercially available, we gauge the accuracy of our emulation by using the measurements reported in [11], on an Intel server platform, as a reference baseline. For simple I/O-dominant benchmarks (latency, throughput), our emulation can be quite accurate. For more complex benchmarks (e.g., database workloads), the differences in the underlying host machines become more apparent, but our emulator stills stays within 2x of the original.

4.1 Latency

In the first experiment, our objective was to assess the random read and write latency of single cache-line sized (64B) accesses to the emulated NVM. To this end, we employed a custom hardware module to generate such accesses and an Integrated Logic Analyzer (ILA) [10] to accurately measure each access latency. The measured read and write latencies for the NVM emulator, and

Table 1. Resource utilization of the NVM storage on AU280

Module name	LUTs	Registers	BRAM
Available resources	1303680	2607360	2016
NVMulator	0.54%	0.19%	0.42%
Memory controller	1.54%	0.93%	1.26%
DMA-engine	0.88%	0.68%	0.74%
PCIe controller	3.09%	1.7%	3.13%

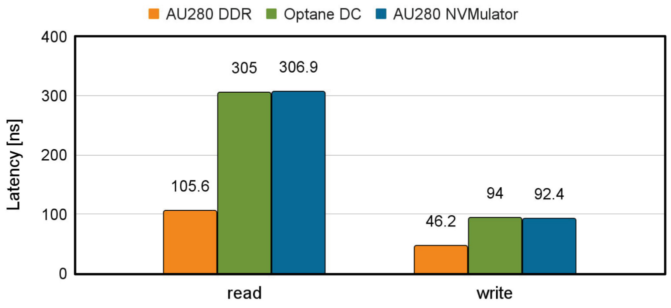

Fig. 6. Cache line sized (64B) read and write latency comparison between the OptaneDC [11] and NVMulator

the OptaneDC are illustrated in Fig. 6. The results reveal that the NVMulator is capable of emulating the read access latency of the OptaneDC with a precision of 0.62% of the target latency (i.e., 305 ns [11]). Similarly, the NVMulator emulates the write access latency with an accuracy of 1.7% of the target latency (i.e., 94 ns [11]). These findings indicate that our proposed approach effectively emulates access latency in close proximity to the intended objective.

4.2 FIO Bandwidth

In this experiment, we aimed to assess NVMulator when used as storage for filesystem-managed data by using the well-known Flexible I/O (FIO) tool [1] for benchmarking. This tool enables us to generate practical I/O traffic on the storage device. To achieve this, we configured the FIO tool to create workloads using the sync ioengine for random read and write accesses. The generated workload comprised a 512 MB file size per thread and a block size of 4KB. We ran this experiment by varying the number of active threads between one and four, and executed fsync() following each write request to the storage, ensuring that the written data was not delayed. For the underlying file system, we employed Ext4, which is a widely used Linux file system.

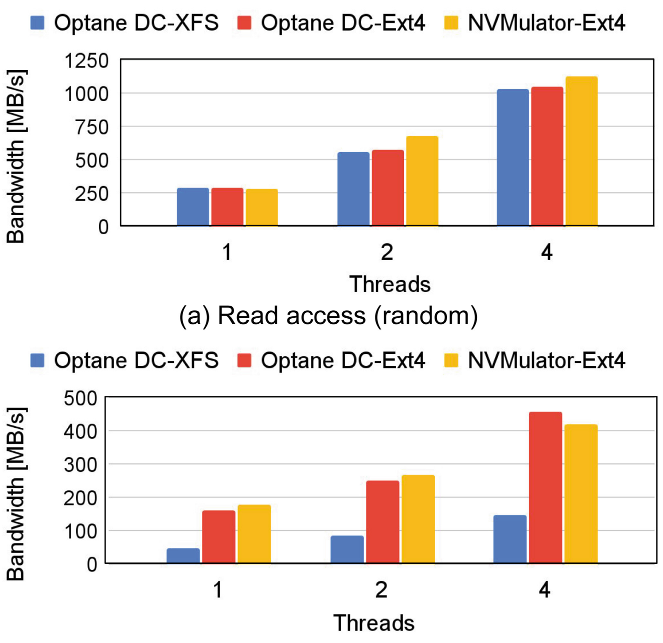

Fig. 7. A FIO 4 KB random read and write bandwidth comparison between the OptaneDC [11] and NVMulator

Figure 7 shows the results of this experiment in comparison to the OptaneDC for random read and write operations. As expected, the NVMulator effectively emulates the OptaneDC under practical I/O traffic generation conditions. We further extended this experiment for sequential read and write operations. To this end, we expanded our block device driver to support *memory coalescing*, enabling the combination of sequential accesses into a single, larger transaction. The outcomes of this experiment are shown in Fig. 8. As expected, the NVMulator effectively emulates both sequential and random access operations.

4.3 Database Application

One of the key advantages of the proposed methodology is its capacity to facilitate easy and seamless integration of emulated NVM into a given system, thereby allowing users to execute applications on it. NVM has demonstrated benefits in database applications, serving as both persistent and computational storage capable of managing database operations through NDP [2,21,22]. To demonstrate the ease of integrating NVMulator for this use-case, we will examine the usage of the emulated NVM as persistent storage for a popular key-value database system. The database system employed here is RocksDB [8], which is an embedded key-value store designed by Facebook/Meta. In order to assess the

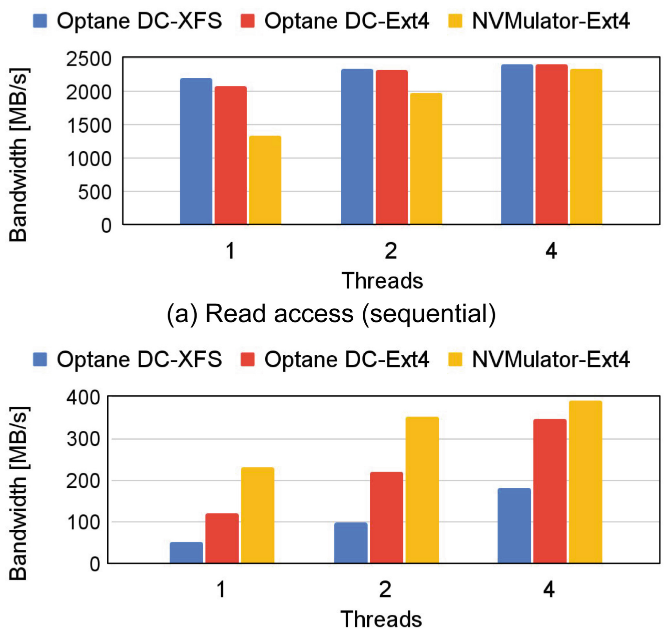

Fig. 8. A FIO 4 KB sequential read and write bandwidth comparison between the OptaneDC [11] and NVMulator

performance of the suggested methodology, we conduct an experiment utilizing the db_bench tool. This involves executing the fillrandom command with a key and value size of 20B and 100M, respectively, while processing 10 million records on RocksDB V5.4.

Figure 9 shows the measured throughput of the number of operations while executing db_bench benchmark for the Ext4 file system. The NVMulator, as demonstrated in the figure, effectively manages read-and-write accesses to the storage device with a similar behaviour as the OptaneDC modules. It is important to acknowledge that the block device driver employed in this research is a simplified version, lacking the comprehensive buffering and caching functionalities present in existing drivers, such as the NVMe block device driver and that the server used for the AU280 and the ones used by the authors of [11] are quite different (e.g. 24 cores compared to 4). However, the raw performance of the block driver is only secondary for this experiment, which aims at accurate emulation of NVM timing behavior, specifically for OptaneDC. As shown here, that can be achieved even without optimizing the block driver further.

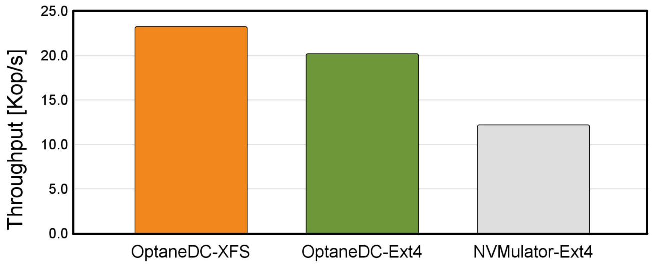

Fig. 9. A comparison of RocksDB throughput while executing db_bench on emulated NVM and OptaneDC. Note that the Optane and AU280 measurements were performed on two different host machines. For this more complex workload, the differences in host machines become more dominant, but the emulation still stays within 2x of the measurements reported in [11].

5 Conclusion

We introduce NVMulator, an open-source easy-to-use NVM emulator for FPGAs, which has been integrated into the TaPaSCo framework to enable the emulation of NVM access latencies using off-the-shelf DRAM memories. Our approach enables users to effortlessly emulate various types of NVM, including PCM and STT, while also incorporating their accelerators into FPGA designs. Furthermore, the NVMulator module can be easily configured through the TaPaSCo API, providing users with the flexibility to modify the emulator's configuration and switch between different NVM types without requiring a complete redesign and re-synthesis of the system. Our evaluation examines both file-system and database access patterns and shows tight accuracy of our emulation for I/O-dominant benchmarks, with the emulator staying within 2x of accuracy even for more complex database workloads. It is thus suitable to support further work on NVM-based Near Data Processing architectures.

Acknowledgment. The authors would like to thank the anonymous reviewers for their valuable comments. This research was funded by the German Research Foundation (DFG) as project #419942270 *neoDBMS*.

References

1. Axboe, J.: Fio tool source code. https://github.com/axboe/fio. Accessed 16 Dec 2021
2. Bernhardt, A., Tamimi, S., Stock, F., Vinçon, T., Koch, A., Petrov, I.: Cache-coherent shared locking for transactionally consistent updates in near-data processing DBMS on smart storage. In: EDBT, pp. 2–424 (2022)

3. Bock, S., Childers, B.R., Melhem, R., Mosse, D.: Hmmsim: a simulator for hardware-software co-design of hybrid main memory. In: 2015 IEEE Non-Volatile Memory System and Applications Symposium (NVMSA), pp. 1–6 (2015)
4. Do, J., Kee, Y.S., Patel, J.M., Park, C., Park, K., DeWitt, D.J.: Query processing on smart SSDs: opportunities and challenges. In: Proceedings of SIGMOD, p. 1221 (2013)
5. Duan, Z., Liu, H., Liao, X., Jin, H.: HME: a lightweight emulator for hybrid memory. In: 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 1375–1380. IEEE (2018)
6. Eilert, S., Leinwander, M., Crisenza, G.: Phase change memory (PCM): a new memory technology to enable new memory usage models (2011)
7. Embedded Systems and Applications Group, TU Darmstadt: Tapasco on Github. https://github.com/esa-tu-darmstadt/tapasco
8. Facebook: Rocksdb (2017). http://rocksdb.org
9. Hoefflinger, B.: ITRS: the international technology roadmap for semiconductors. In: Hoefflinger, B. (ed.) Chips 2020, pp. 161–174. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23096-7_7
10. Xilinx Inc.: Vivado design suite user guide: programming and debugging (UG908). Technical report, Xilinx Inc. (2021). https://docs.xilinx.com/r/en-US/ug908-vivado-programming-debugging/ILA
11. Izraelevitz, J., et al.: Basic performance measurements of the intel optane DC persistent memory module. arXiv preprint arXiv:1903.05714 (2019)
12. Kim, S., Oh, H., Park, C., Cho, S., Lee, S.W., Moon, B.: In-storage processing of database scans and joins. Inf. Sci. **327**, 183–200 (2016)
13. Korinth, J., Hofmann, J., Heinz, C., Koch, A.: The TaPaSCo open-source toolflow for the automated composition of task-based parallel reconfigurable computing systems. In: Applied Reconfigurable Computing (2019)
14. Lee, T., Kim, D., Park, H., Yoo, S., Lee, S.: FPGA-based prototyping systems for emerging memory technologies. In: 2014 25nd IEEE International Symposium on Rapid System Prototyping, pp. 115–120 (2014)
15. Lee, T., Yoo, S.: An FPGA-based platform for non volatile memory emulation. In: 2017 IEEE 6th Non-Volatile Memory Systems and Applications Symposium (NVMSA), pp. 1–4 (2017)
16. Omori, Y., Kimura, K.: Performance evaluation on NVMM emulator employing fine-grain delay injection. In: 2019 IEEE Non-Volatile Memory Systems and Applications Symposium (NVMSA), pp. 1–6 (2019)
17. OpenSSD Project: COSMOS Project Documentation (2019). http://www.openssd-project.org/wiki/Cosmos_OpenSSD_Technical_Resources
18. Peng, I.B., Gokhale, M.B., Green, E.W.: System evaluation of the intel optane byte-addressable NVM. In: Proceedings of the International Symposium on Memory Systems, pp. 304–315 (2019)
19. Poremba, M., Xie, Y.: NVMain: an architectural-level main memory simulator for emerging non-volatile memories. In: 2012 IEEE Computer Society Annual Symposium on VLSI, pp. 392–397 (2012)
20. Salamat, S., Haj Aboutalebi, A., Khaleghi, B., Lee, J.H., Ki, Y.S., Rosing, T.: Nascent: near-storage acceleration of database sort on SmartSSD, FPGA 2021, pp. 262–272. Association for Computing Machinery, New York (2021). https://doi.org/10.1145/3431920.3439298

21. Tamimi, S., Stock, F., Koch, A., Bernhardt, A., Petrov, I.: An evaluation of using CCIX for cache-coherent host-FPGA interfacing. In: 2022 IEEE 30th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), pp. 1–9 (2022)
22. Vinçon, T., et al.: Near-data processing in database systems on native computational storage under HTAP workloads. Proc. VLDB Endow. **15**(10), 1991–2004 (2022). https://doi.org/10.14778/3547305.3547307