FPGA-Accelerated Non-Volatile Memory Access

Yussuf Khalil
uyebg@student.kit.edu
Karlsruhe Institute of Technology

Thorsten Gröninger
groeninger@kit.edu
Karlsruhe Institute of Technology

Lukas Werling
lukas.werling@kit.edu
Karlsruhe Institute of Technology

Frank Bellosa
bellosa@kit.edu
Karlsruhe Institute of Technology

ABSTRACT

Intel recently introcuded Optane Persistent Memory (PMem), a byte-addressable non-volatile memory that is attached directly to the processor's memory bus and thereby allows access with traditional load/store instructions. This model enables very low latency, but can have severe performance deficits in practice. In particular, synchronous write accesses can lead to CPU stalls [5] and leave less CPU time for other storage processing tasks (e.g., file systems, compression, encryption) and even completely unrelated CPU-bound processes [4].

We propose mediating access to PMem using an FPGA-based PCIe device. This way, we can preserve low latencies while also freeing the CPU from certain storage processing tasks. Notably, in our system setup, the Optane memory is connected directly to the FPGA instead of the CPU's memory controller. In the future, we plan to move away from PCIe in favor of the upcoming CXL.mem protocol [3].

As a first step, we implement *asynchronous copy offloading*, which is a mechanism for overcoming performance issues from write stalls [4]. Our design features SR-IOV support and a custom low-latency MMU tailored for the typical large sizes of Optane modules. The accompanying Linux driver provides user applications with a `memcpy()`-style interface that allows for parallelized submission of copy tasks without runtime overhead for locks or system calls.

1 ASYNCHRONOUS COPY OFFLOADING

The idea behind *asynchronous copy offloading* is to have the actual copy operations done by periphery hardware, i.e., not by the CPU itself. This concept is asynchronous by nature: if it were synchronous, there would be no benefit as the CPU would have to wait for the periphery hardware (which, in turn, waits for the memory) instead of waiting for the memory directly. By being asynchronous it is possible to let the CPU do other work after it has submitted a copy command. For this approach to be successful, the command submission must be faster than the time that would be spent waiting. In turn, it is imperative to make the submission process as quick as possible. At the same time, in order to be sensible for practical applications, the implementation must be able to saturate the memory's bandwidth to its fullest, while also providing short latencies. These goals guide the design of our approach.

2 DESIGN OVERVIEW

In essence, we aim to co-design specialized hardware and software to implement asynchronous copy offloading for PMem. To this end, we employ an FPGA and connect it to the base system via PCI Express. Now, if the PMem modules were to be part of the system's main memory, all copied data would need to travel through the PCIe bus twice, once for reading and a second time for writing. As we aim for minimal latency, we connect the Optane memory to the FPGA directly instead. Via SR-IOV, our design allows to configure a pair of ring buffers to be used as command queues for read (FPGA $\rightarrow$ system memory) and write (system memory $\rightarrow$ FPGA) operations per each virtual function (VF). User processes may allocate an arbitrary number of VFs from our kernel driver, and thereby, an arbitrary amount of command queues to enable lock-free parallel submission. This further allows processes to directly notify the hardware about new commands by writing to a *bell register* specific to each VF. Otherwise, we would have to employ polling (costly in terms of latency and wasted bandwidth) or system calls (that cause further overhead) for proper isolation.

Given that Optane modules are sold in sizes of up to 512 GiB [1] and that PMem is typically used by only very few applications in a system, we opted for a MMU design with linear page tables and 16 GiB pages. One page table per VF is stored in local SRAM on the FPGA. Our implementation guarantees address translations to be completed within a single clock cycle without having a TLB. Page tables are managed by our kernel driver and processes may allocate pages via a system call.

We offer a library that encapsulates all necessary functionality to user applications and provides them with what is essentially an asynchronous `memcpy()`. A single system call is still necessary the first time a page in system memory is referenced in order to establish an IOMMU mapping. This issue may be alleviated in the future when CPUs supporting *Shared Virtual Memory* become available [2].

REFERENCES

[1] Intel Corporation. 2022. Intel® optane™ persistent memory (pmem). (Aug. 27, 2022). https://www.intel.com/content/www/us/en/products/details/memory-storage/optane-dc-persistent-memory.html.

[2] Intel Corporation. 2022. Intel® virtualization technology for directed i/o architecture specification. (June 2022). https://www.intel.com/content/www/us/en/content-details/671081/intel-virtualization-technology-for-directed-i-o-architecture-specification.html.

[3] Debendra Das Sharma and Siamak Tavallaei. 2020. Compute express link 2.0 white paper. *Tech. Rep.*

[4] Lukas Werling, Christian Schwarz, and Frank Bellosa. 2021. Towards less cpu-intensive pmem file systems. (Sept. 21, 2021). https://www.betriebssysteme.org/wp-content/uploads/2021/09/FGBS_Herbst2021_Folien_Werling.pdf.

[5] Jian Yang, Juno Kim, Morteza Hoseinzadeh, Joseph Izraelevitz, and Steve Swanson. 2020. An empirical guide to the behavior and use of scalable persistent memory. In *18th USENIX Conference on File and Storage Technologies (FAST 20)*, 169–182.