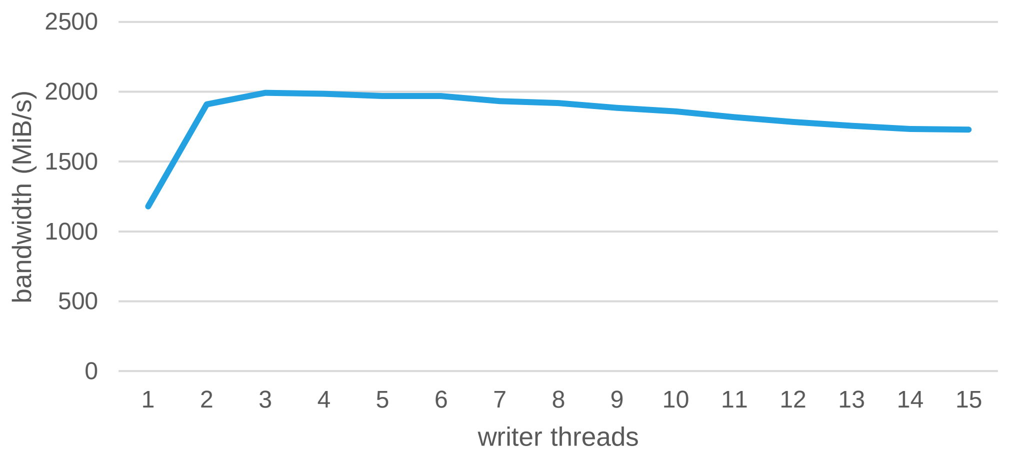
FPGA-Accelerated Non-Volatile Memory Access

Yussuf Khalil, Thorsten Gröninger, Lukas Werling, Frank Bellosa

Motivation

- Optane PMem is a recent non-volatile byte-addressable memory
- Bandwidth drops with increasing parallelism

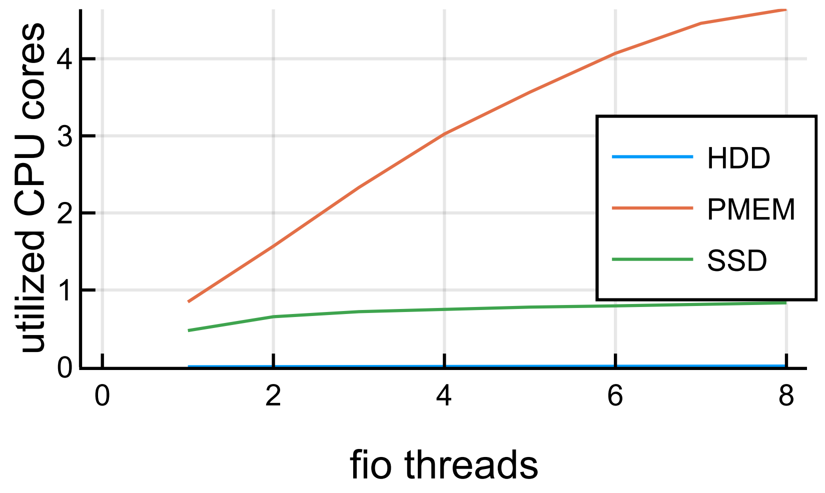


Motivation

- CPU utilization scales linearly with parallel writer threads

- Cores stall while waiting for write operations to complete
  - e.g., file system implementations become inefficient

- Hurts overall system performance



Asynchronous Copy Offloading to the Rescue

Idea: Make copy operations asynchronous

- Let external hardware perform the actual copy operations
  - Previous approaches included Intel I/OAT and GPUs

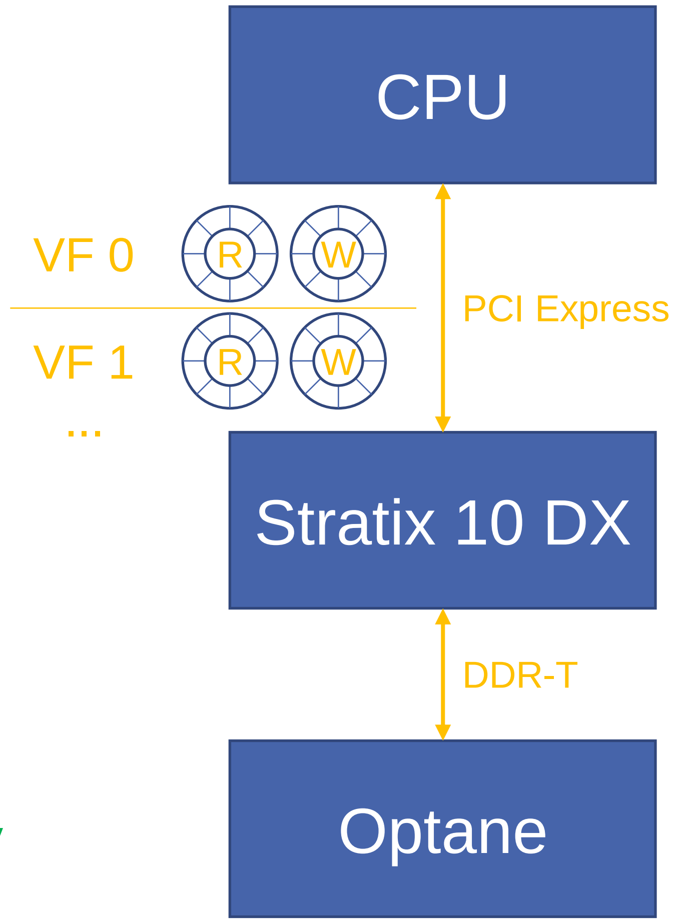
- CPU only submits copy commands

- Must provide:
  - ✓ Quick command submission
  - ✓ Short latencies
  - ✓ Saturated Optane bandwidth

Why not use an FPGA for that?

Design: Hardware

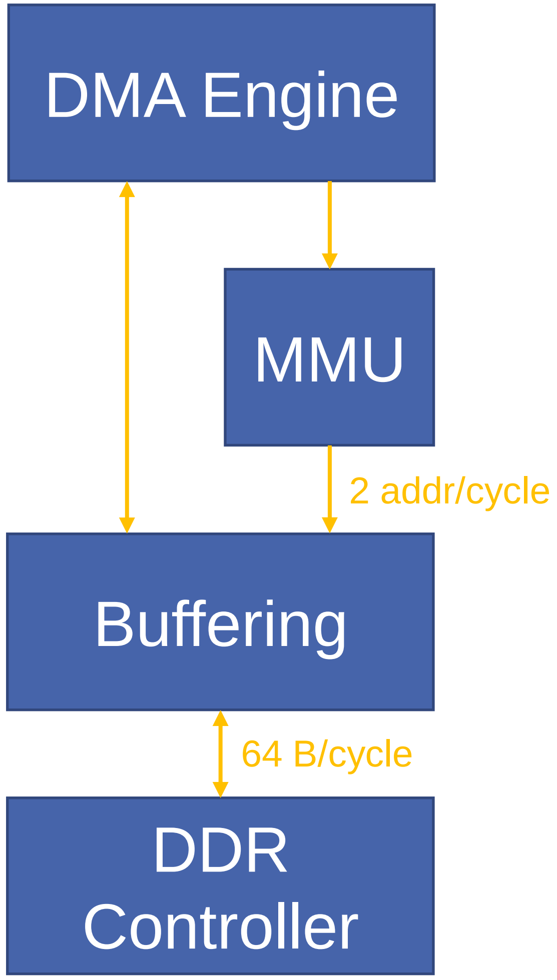
- Based on Intel Stratix 10 DX
  - Connected to base system via PCIe
- SR-IOV for multiplexing
  - Each virtual function (VF) provides
    - Read command queue
    - Write command queue
    - Bell register for command notification ✓ Latency
- Optane connected to FPGA instead of CPU ✓ Latency



Design: Hardware Core Logic

- Specialized MMU
  - 16 GiB pages
  - One linear page table per VF stored in local SRAM
  - Two address translations simultaneously
  - Guaranteed completion within single clock cycle ✓ Latency
- Buffering logic for 64 read + 64 write operations ✓ Bandwidth
- 64 B read or write per clock cycle
  - 250 MHz ⇒ up to 15 GiB/s

Future optimization: parallel read and write

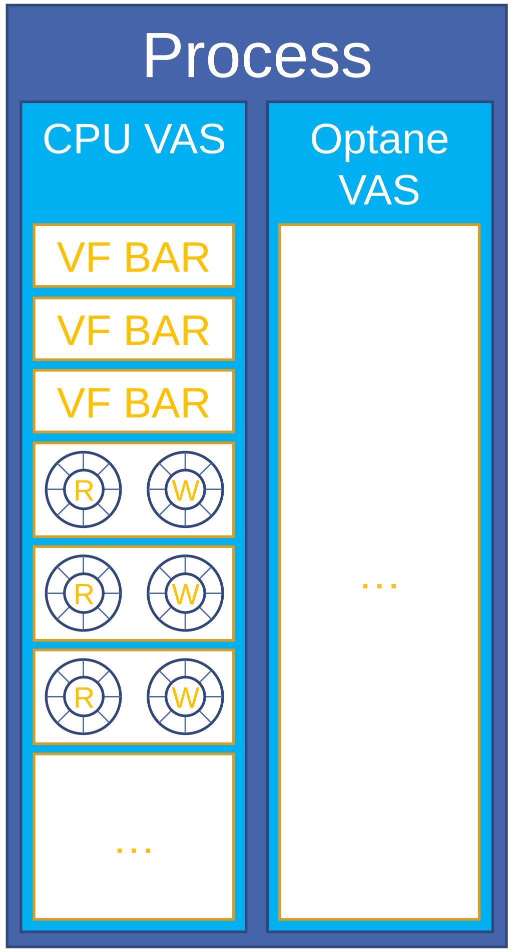


Design: Linux Driver

- Provides device file with:
  - open() – allocates a VF to the process
    - Multiple allocations

      ⇒ lock-free parallel command queues ✓ Quick submission
  - mmap() – maps VF's BAR into CPU VAS
    - Processes can directly write to bell register ✓ Quick submission
  - ioctl(MMAP) – maps page in Optane VAS

- Driver shares page tables between VFs of same process
  - Single Optane VAS per Linux process, not per VF

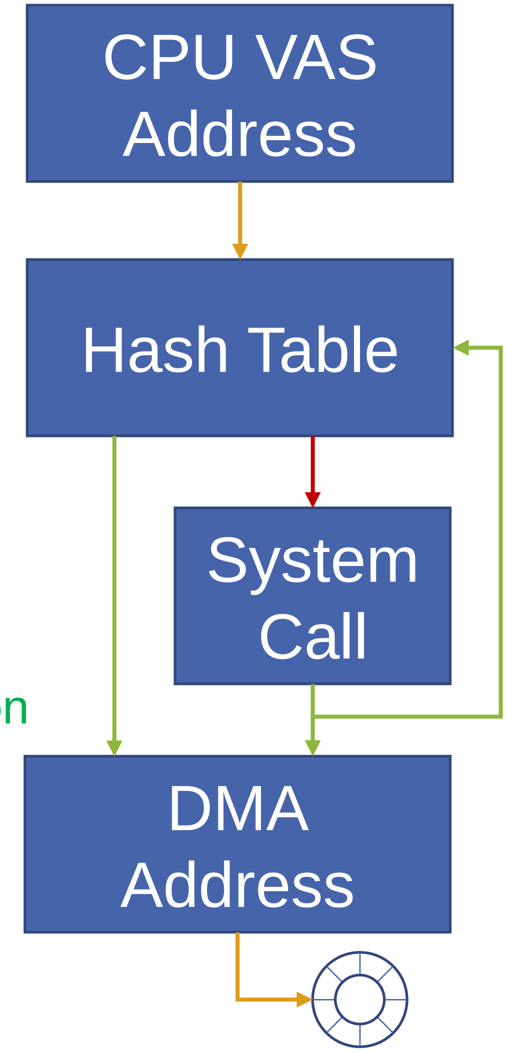


Design: User-Space Library

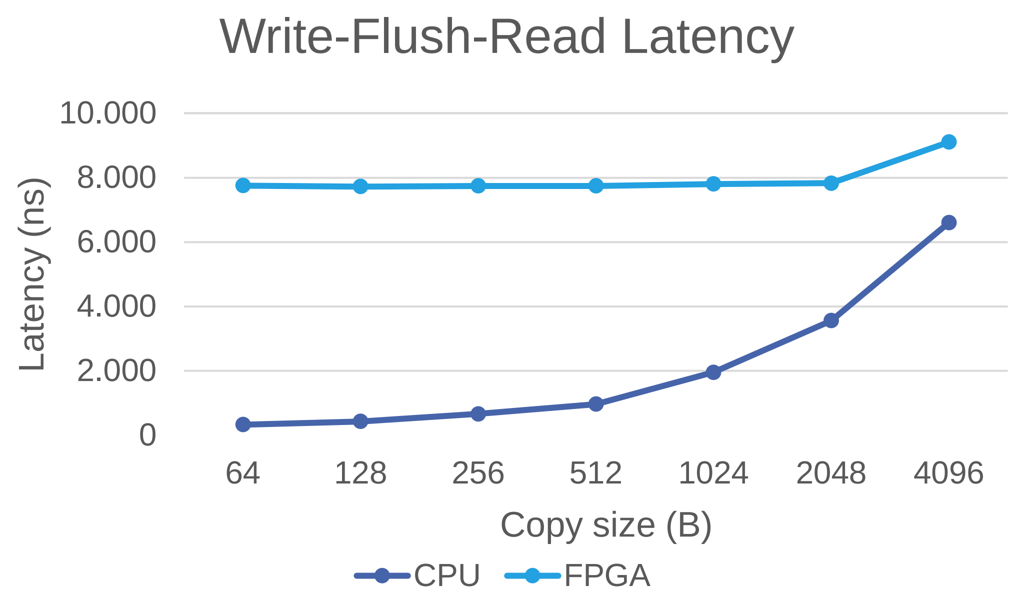
- Encapsulates all initialization and command functionality
- Provides processes with an asynchronous memcpy()
  - _wait() variants with busy waiting available

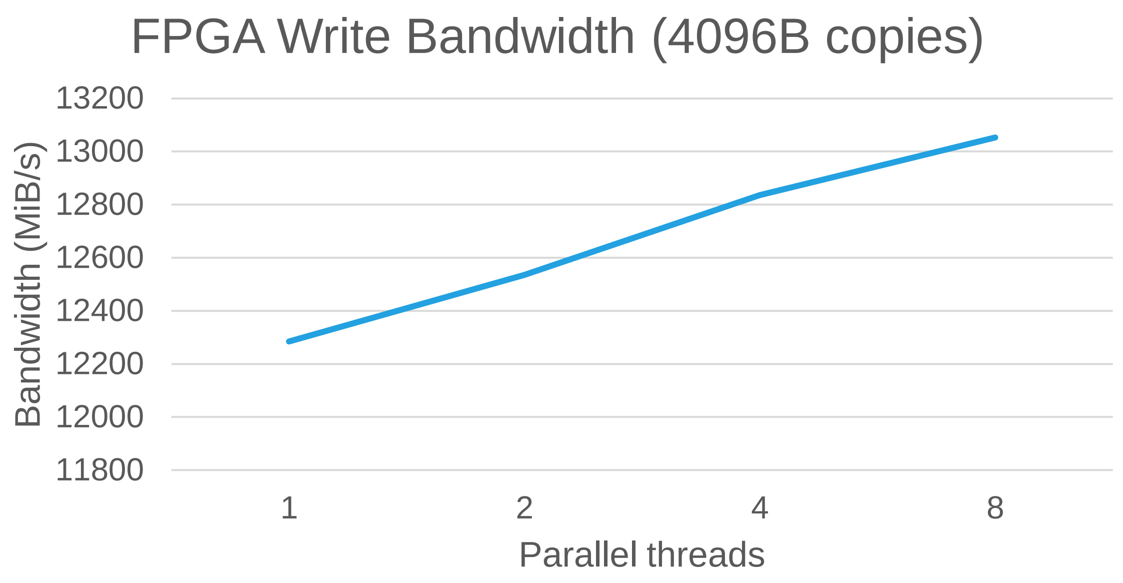
  Future optimization: use UMWAIT
- First reference to page causes syscall for IOMMU mapping
  - Zero system calls otherwise after initialization ✓ Quick submission
  - CPU VAS ↔ IOVAS mappings cached in hash table

  Future optimization: use Shared Virtual Memory



Evaluation: DDR4





Intel Cascade Lake, Single DDR4-2133 DIMM

Implementation would be capable of saturating Optane bandwidth.

Future Work

- Switch to CXL ✓ Latency
  - Mixed sync + async operations may become feasible
- File system acceleration
- Out-of-order processing ✓ Bandwidth
  - Optane performance depends strongly on access patterns
- More primitives
  - memset()
  - memcpy() within Optane
  - Atomics

Conclusion

- System performance suffers with parallel Optane accesses
- Asynchronous copy offloading uses external hardware to perform copy operations
- Initial numbers with our FPGA-based approach look promising
- Interesting starting point for future research
  - CXL, out-of-order, file systems…