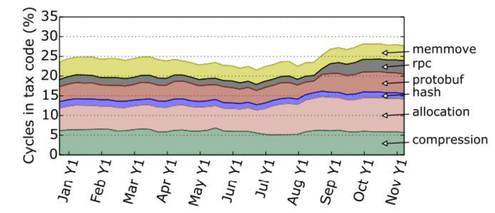
Toward Hardware-Assisted Kernel-Bypass Data Movement and Transfer

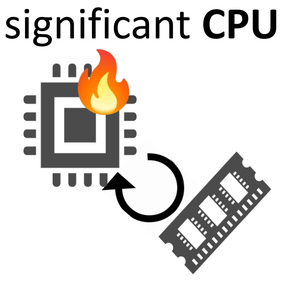
Keisuke Iida
The University of Tokyo
iida@os.is.s.u-tokyo.ac.jp

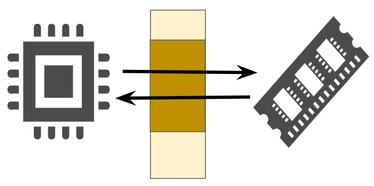
Takahiro Shinagawa
The University of Tokyo
shina@is.s.u-tokyo.ac.jp

1. Background: Datacenter Tax

- Frequent and repetitive data movement and transfer
  - *memset()* and *memcpy()*
  - Inter-process communication (IPC)
  - Data compression etc...
- Consume 22-27% of CPU time in Google DC [1]
  - Cause significant **CPU overhead** and **cache pollution**

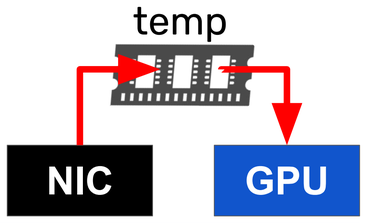






2. Previous Work: On-chip DMA

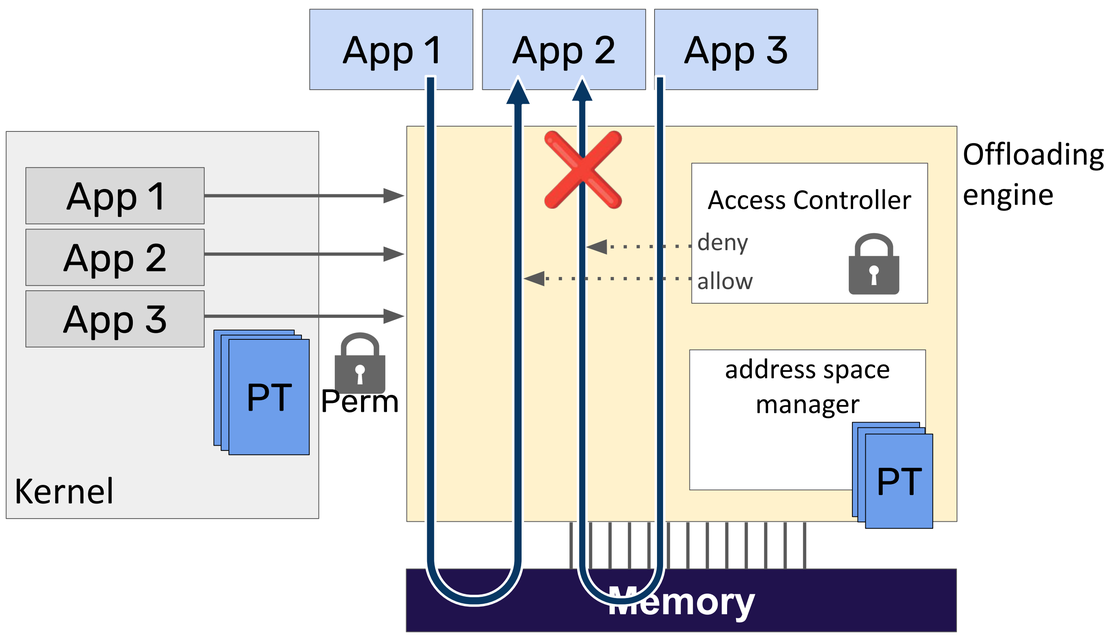
- Intel Data Streaming Accelerator (DSA) [2]
  - An on-chip memory operation offloading engine
    - Supports memory move/fill and simple computations
    - Supports per-process virtual addresses via using PASID
  - Limitations
    - Only supports simple memory-to-{device|memory} transfers
      - No support for device-to-device transfers
    - Restricted to vendor-defined functions



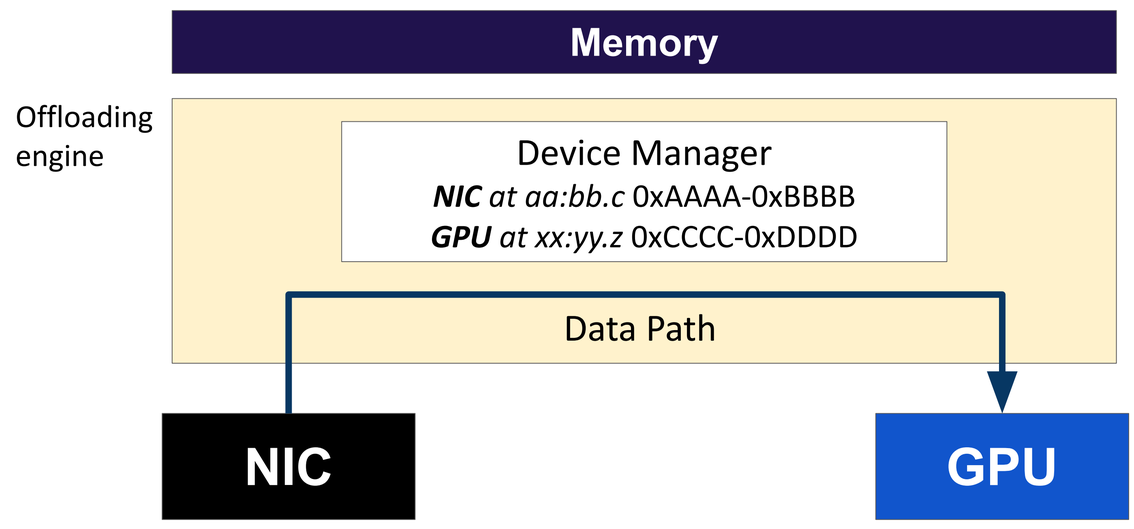
3. Proposal: Context-aware Direct Memory Access (DMA) Engine

GOAL: Empower users and kernels to offload more advanced data operations to hardware

① Inter-address-space data copy
- DMA engine understands virtual address spaces
  - Integrated address-space manager and MMU/TLB
  - Capability-based access control mechainsm
- OS kernel interfaces with the DMA engine for software context
  - Provides page tables and access permissions

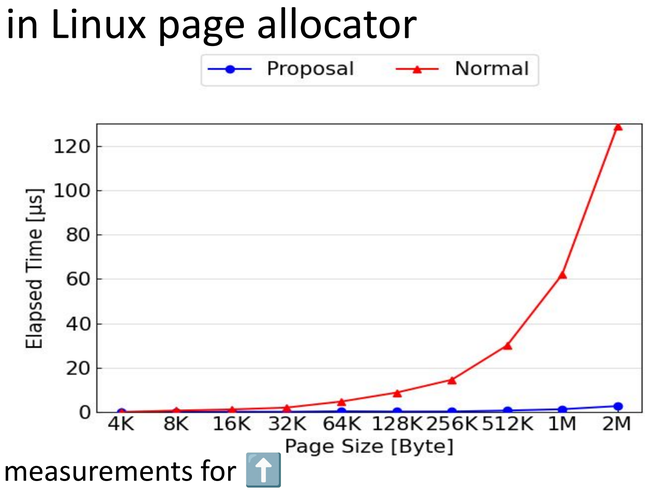


② Transparent Device-to-Device data movement
- Performs peer-to-peer DMA without device support
  - E.g., send packets directly from NIC to GPU
- Maintains device topology and I/O memory map
- Require TLP-transparent bridging machnism



4. Prototype Implementation

- Implemented a prototype PCIe hardware on FPGA
  - Alveo U50 board
  - using verilog-pcie library [3]
- Successfully offloaded memset() and memcpy() in the Linux kernel
  - Offloaded zero-fill operation in Linux page allocator
    - Improved performance of alloc_pages()/free_pages()
- Next step: implement address space manager



CPU-based vs. **Offloaded** approach elapsed time measurements for ⬆️ alloc_pages() and __free_pages() for each page size.

5. Future Extensions

① More OS function offloading support
- Copy (on-write, between NUMA nodes, huge/nomal pages)
- Lock and transaction operations, device I/O protection, etc.

② User-defined memory operation support
- Provide an execution environment for user-defined code
  - E.g., run eBPF inside the DMA engine

③ Confidential virtual machine (CVM) support
- Challenge: How to perform key exchange?
  - Between the DMA engine and CVMs
- Idea: Leverage CVM's built-in key exchange mechanism
  - Designed for live migration

References
[1] S. Kanev et al. Profiling a warehouse-scale computer. In Proc. ISCA 2015, page 158–169, 2015.
[2] R. Kuper et al. A quantitative analysis and guidelines of data streaming accelerator in modern intel xeon scalable processors. In Proc. ASPLOS 2024, pages 37–54, 2024.
[3] Forencich, A.: alexforencich/verilog-pcie: Verilog PCI express components, (online), available from ⟨https://github.com/alexforencich/verilog-pcie⟩ (accessed 2025-07-04)