A direct I/O framework for SoC-based accelerators

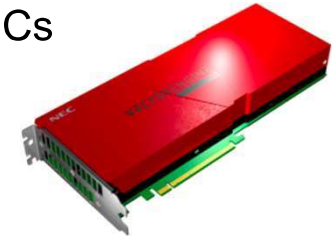
Shinichi Awamoto, Antonio Barbalace, Michio Honda
University of Edinburgh

SoC-based Accelerators

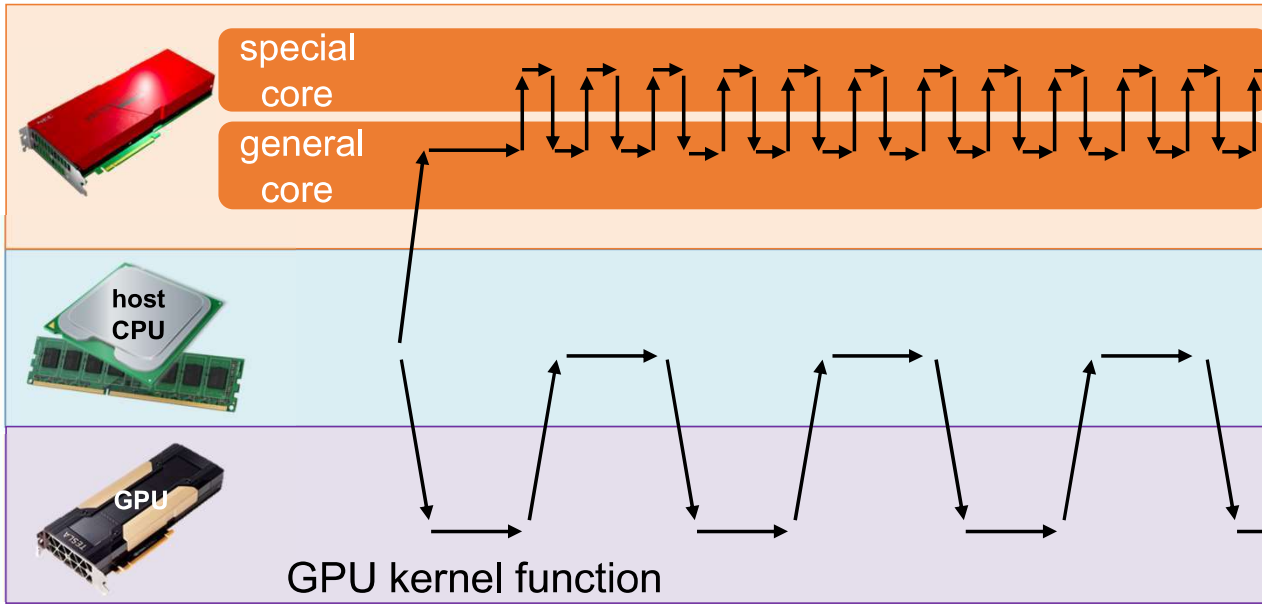
They can execute the entire application code on its general-purpose cores.

SmartNICs

Xilinx Zynq



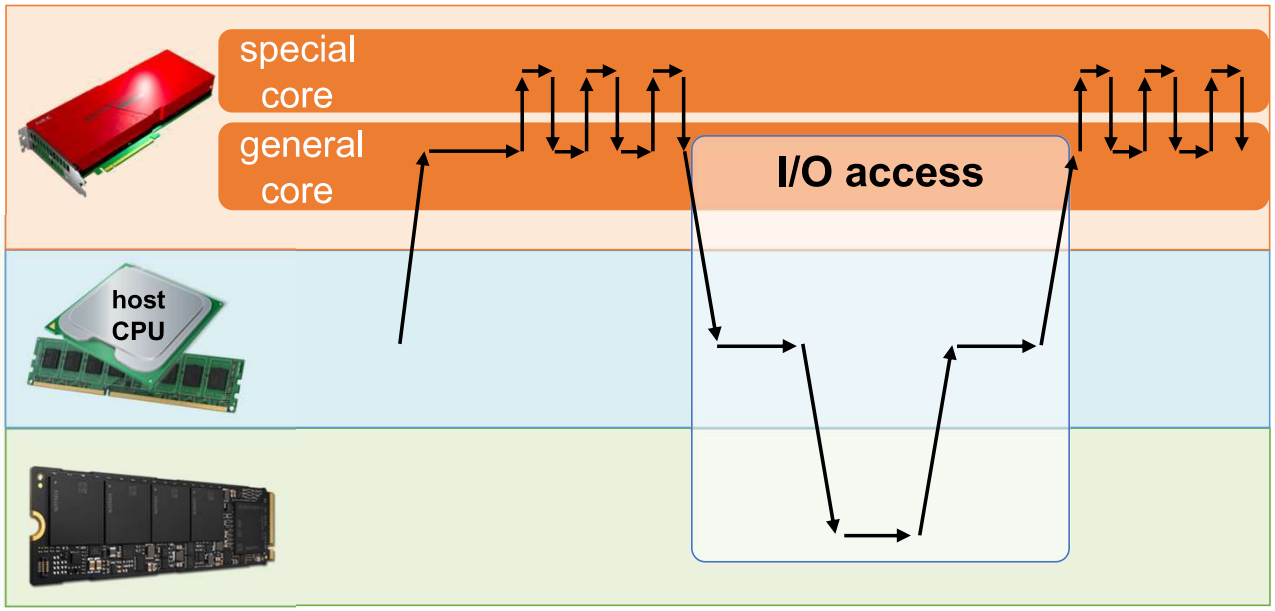
NEC SX-Aurora TSUBASA
Vector Engine



photos from: xilinx.com, www.mellanox.com, uk.nec.com, nvidia.com

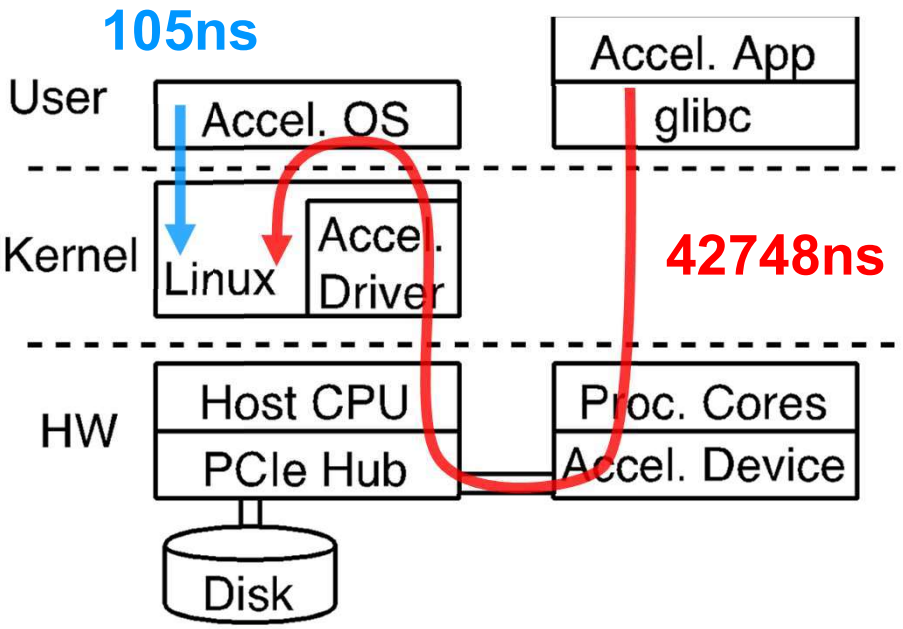
The I/O path in the accelerator

Yet, the host system mediates data access, incurring overhead.

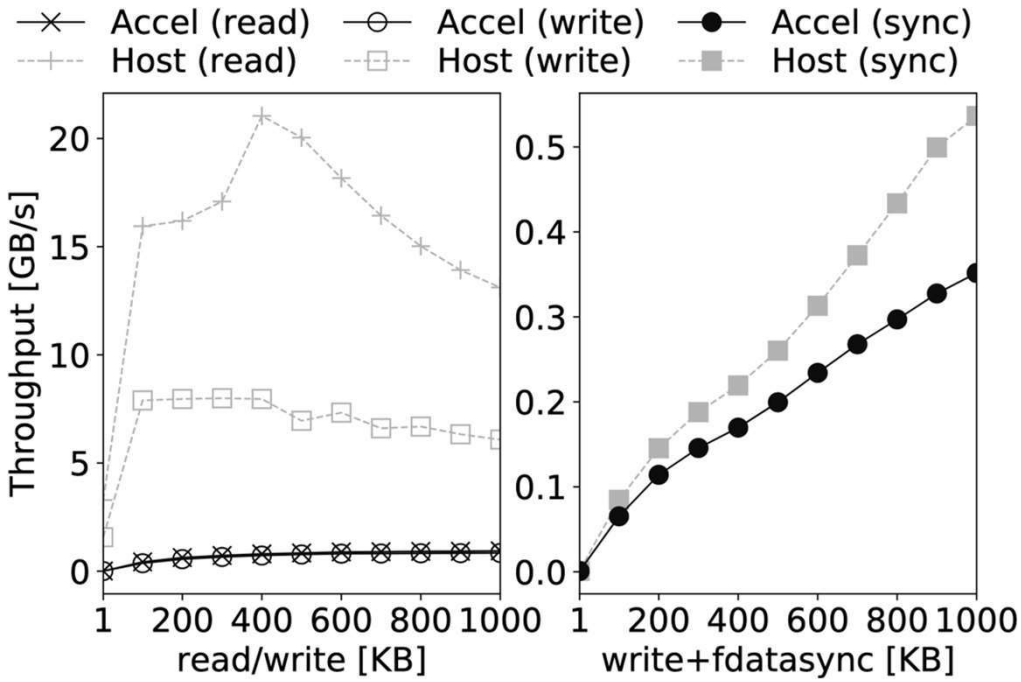


I/O latency analysis

Multiple data copies and dispatch inside of redirected system calls increase the latency.

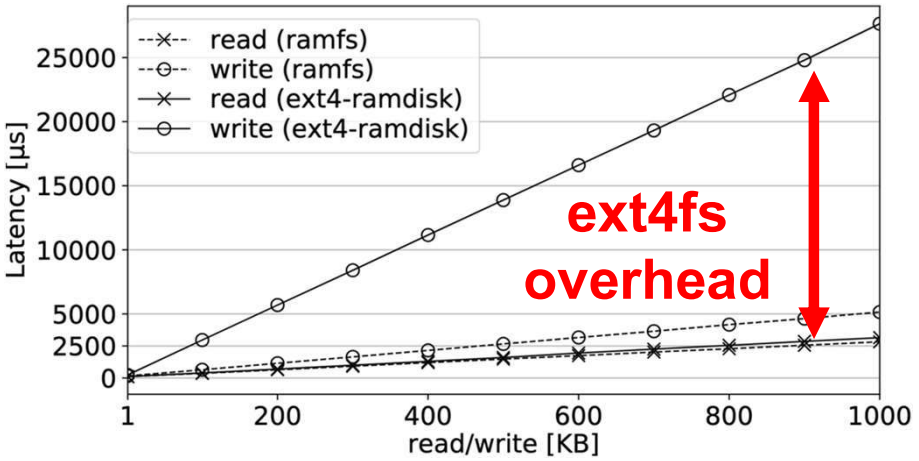


Even on microbenchmarks, the overhead is obvious.



Design options

- Direct buffer cache access
  - e.g., GPUfs (ASPLOS '13), SPIN (ATC '17)
  - Only DMAs are mitigated.
- Heterogeneous-arch kernels
  - e.g., *Multikernel (SOSP '09)*, *Popcorn Linux (ASPLOS '17)*
  - No kernel context in some accelerators
- **Linux kernel library** *(RoEduNet '10)*
  - General file system overheads

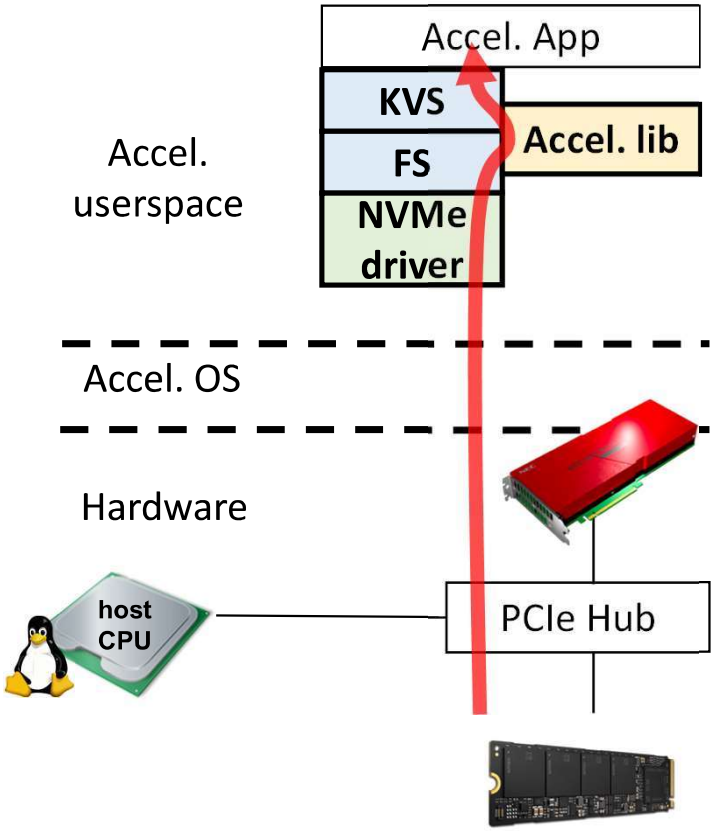


Conventional system software does not perform well on wimpy accelerator cores.

Design decisions

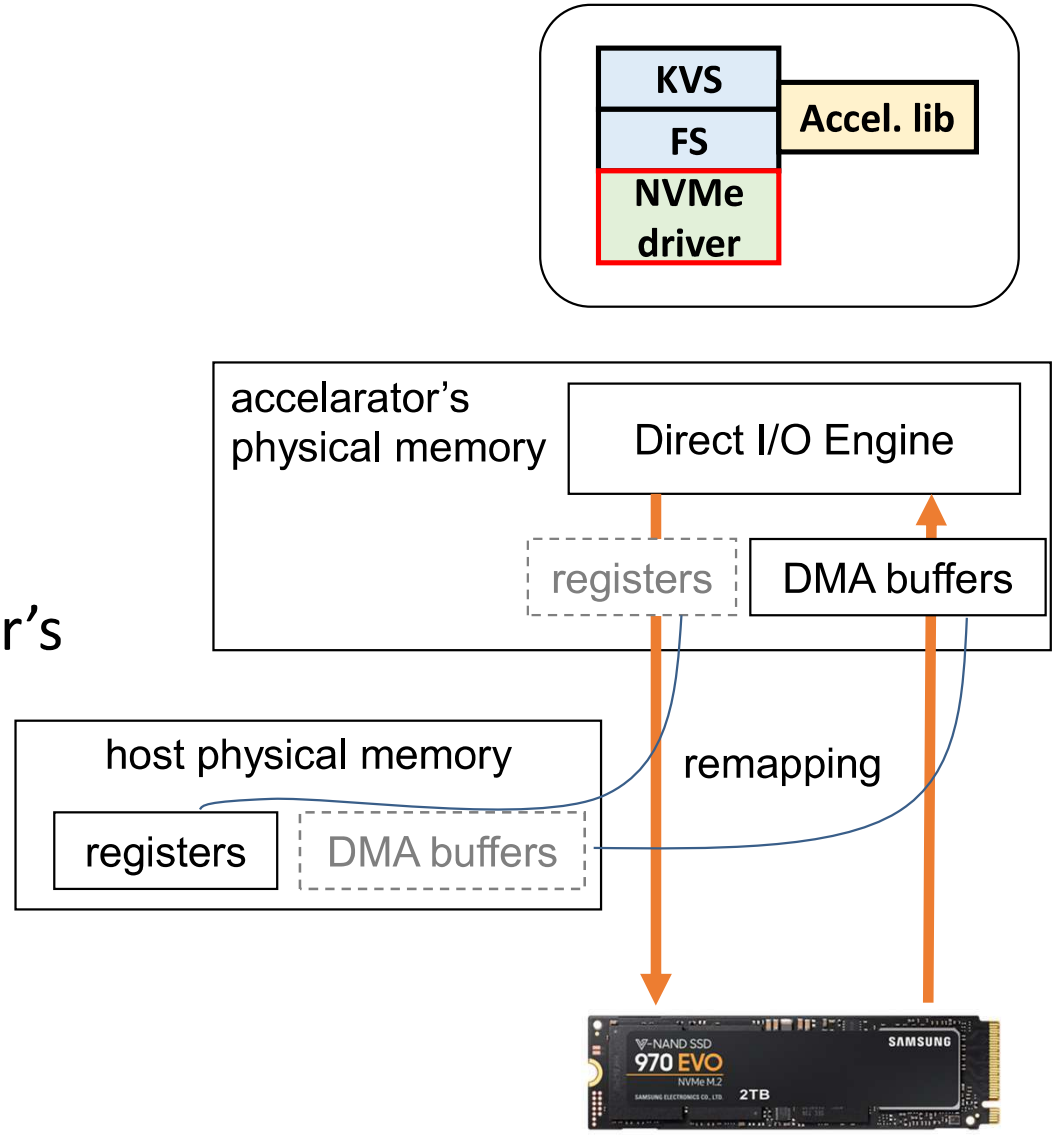
Full stack design of device driver, file system and app is needed.

- Userspace device drivers
- Lightweight data path abstractions
- Accelerator-specific software specialization

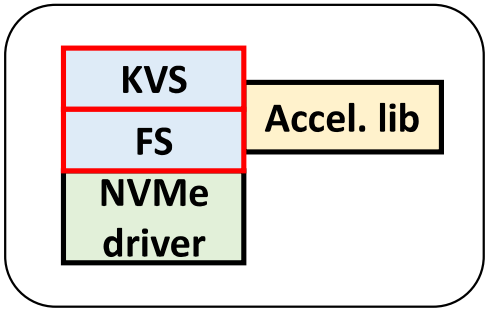


Device driver

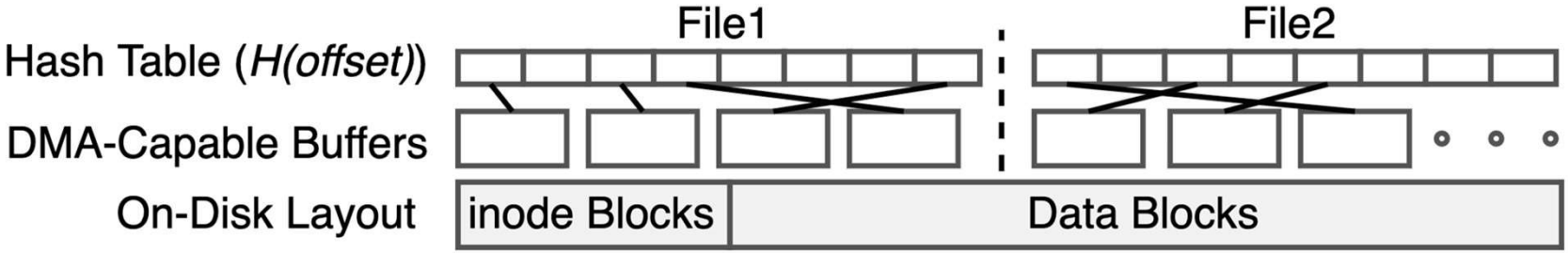
Device registers and DMA buffers are remapped from the host into accelerator's address space.



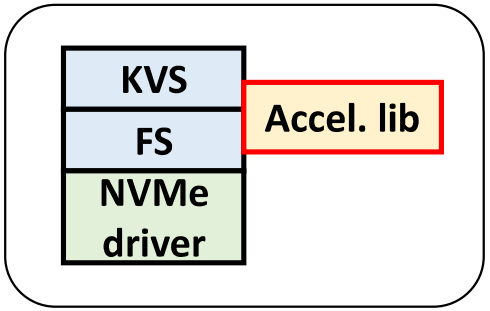
File system & key-value store



- Lightweight ext2-like file system to minimize software overheads
- Ongoing effort of building a KVS from scratch

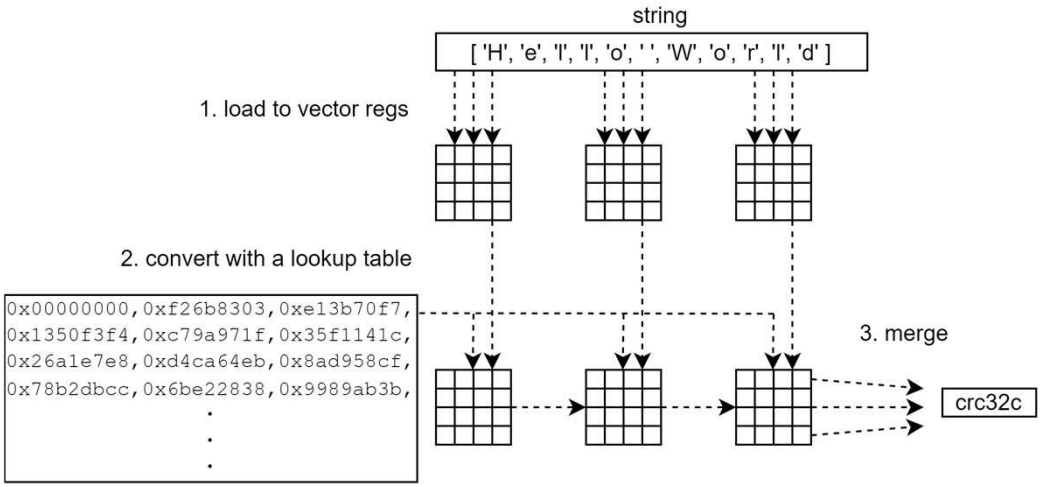


Vectorized CRC32C algorithm



The faster checksum algorithm could speed up file systems, key value stores and network stacks.

- rewritten in vector instructions
- processing 256 32-bit integers at once



Evaluation setup

- Host:

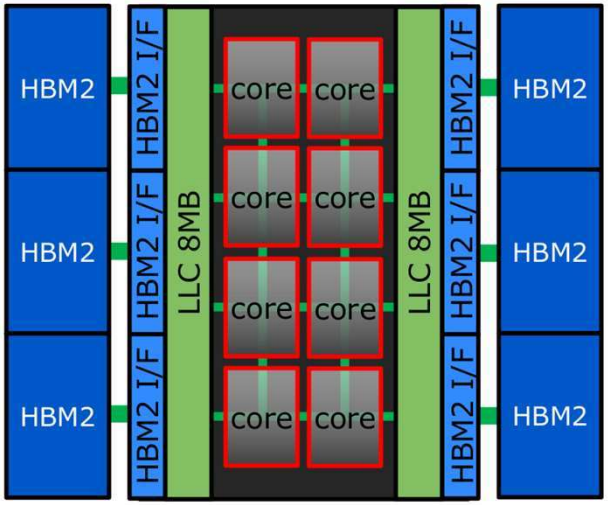
  Intel Xeon Gold 6126 (2.60GHz, 12-cores)

  192GB RAM

- NVMe SSD: Samsung 960GB PM983
- Accelerator: NEC SX-Aurora TSUBASA

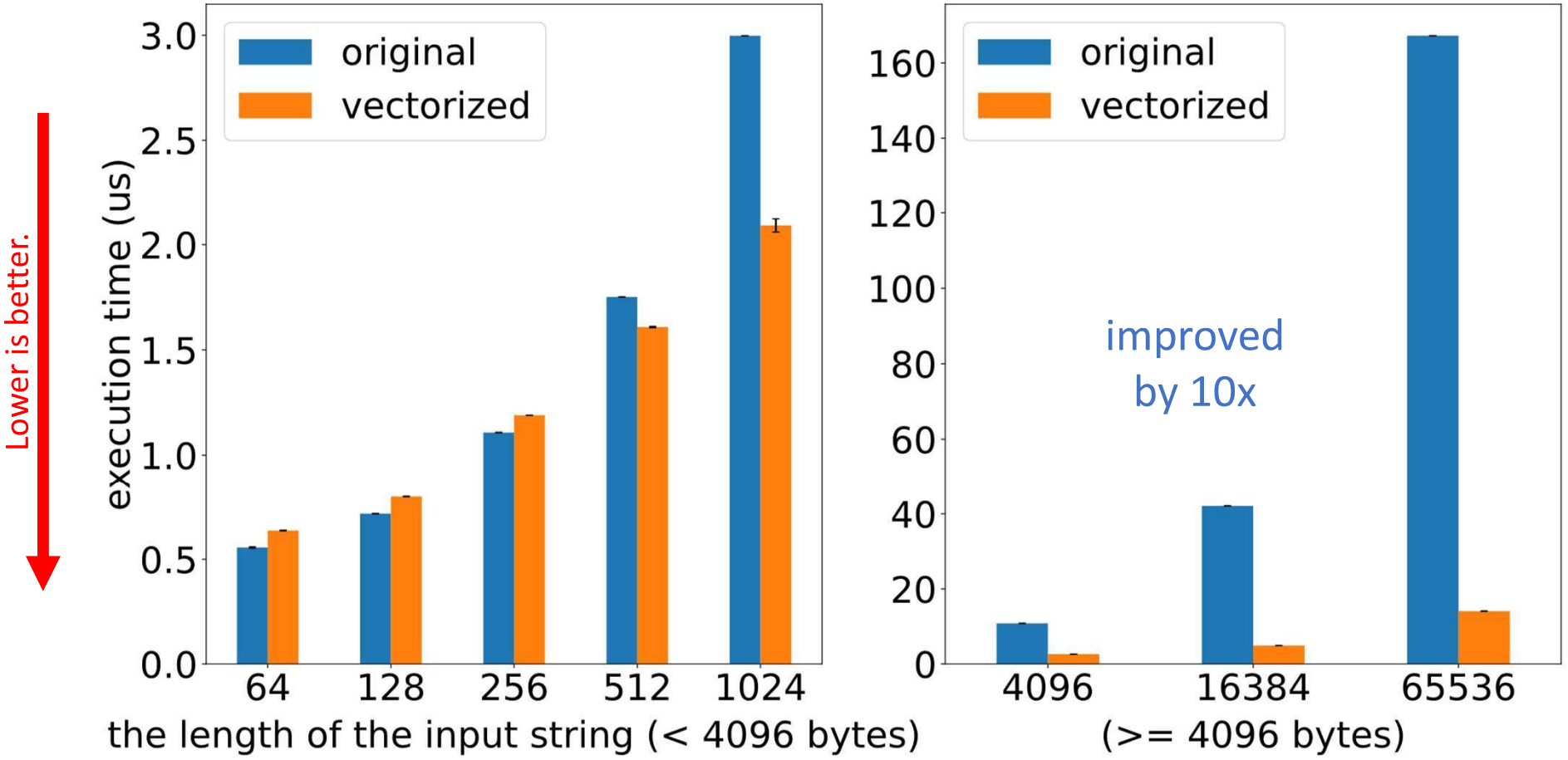
  8 cores @1.4GHz

  48GB RAM



photos from: sx-aurora.github.io

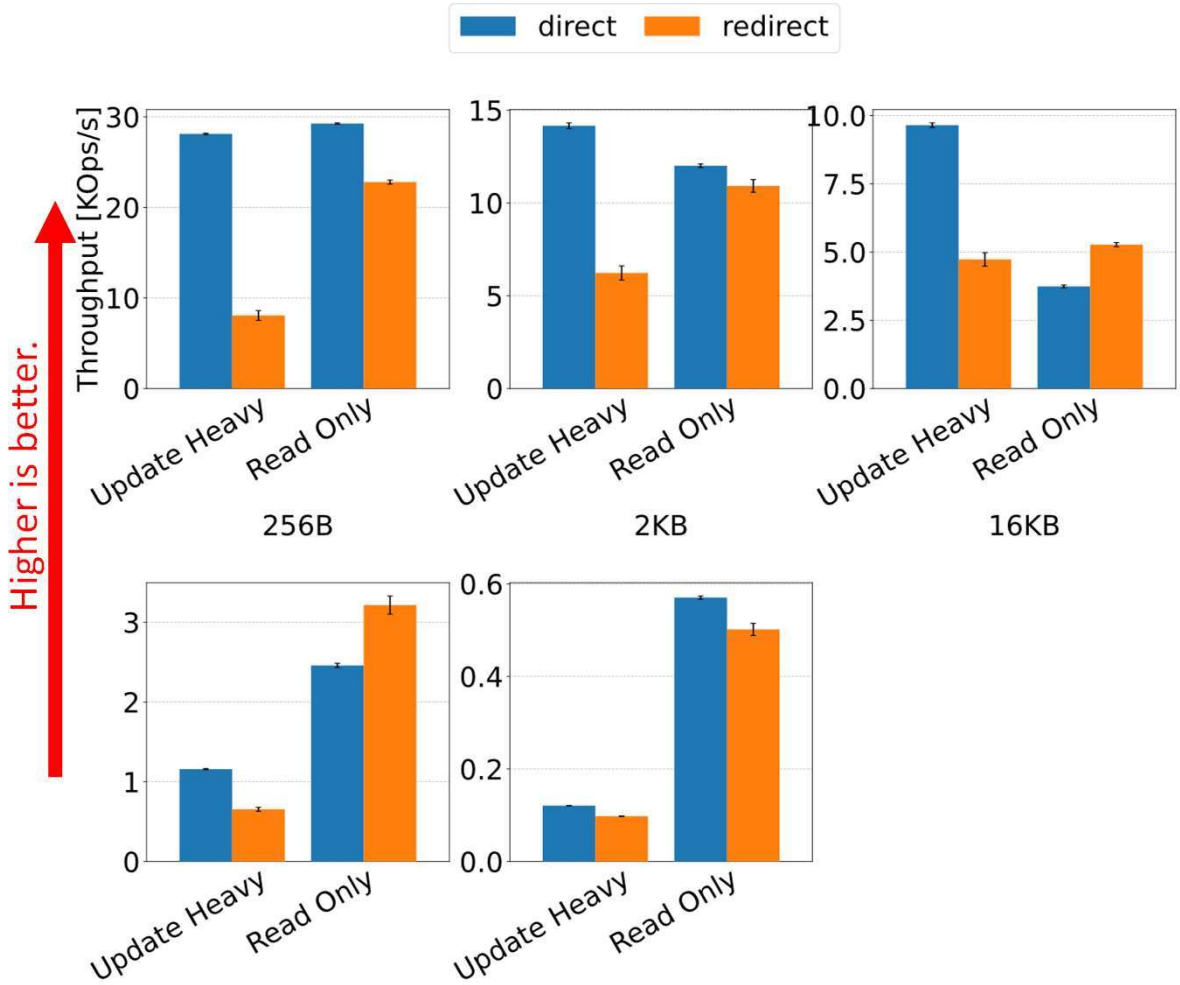
Microbenchmark of CRC32C



Realistic KVS workloads: YCSB benchmark

- LevelDB w/ sw specialization and direct I/O
  - Comparison w/ syscall-based LevelDB
  - Data size varies from 256B to 1MB
- up to 2.5x throughput gain
- 29% performance drop observed

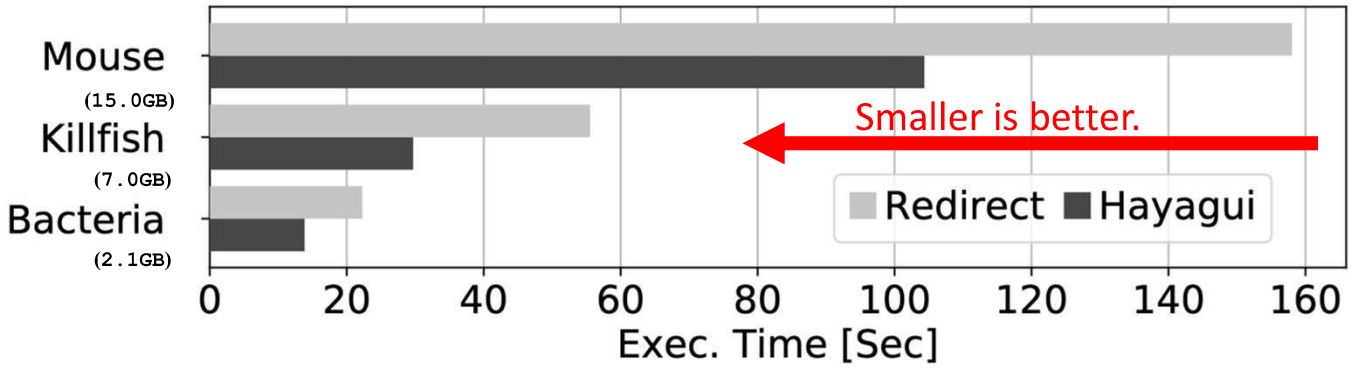
  This will be addressed by the new KVS.



Realistic application workloads: Genome sequence matching app

- The app reads the data from storage, transforms them, and stores them back in the storage.
- The app then processes the transformed data.
  - Analyse DNA sequence
- 33-46% reduction in execution time

Operation	Time [s] Bacteria	Killfish	Mouse
Load reference sequence	0.032	0.010	2.282
Index reference sequence	0.634	0.259	33.859
Save reference index	0.093	0.101	2.019
Load reference index	0.028	0.101	1.463
Load target sequence	19.997	53.943	113.768
Matching	1.500	1.354	3.326
Total	22.284	55.768	156.717



Summary & Future work

- I/O performance and software overhead matters in SoC-based accelerators.
- Direct storage I/O improves performance by up to 46% in the genome sequencing app.
- Software specialization improves performance by up to 2.5x in key value store.

- Ongoing work
  - network stack
  - SmartNIC support
  - resource sharing between the host