Optimizing demanding I/O applications
The HL-LHC ATLAS Readout case study

IEEE RT - 26 May 2026

Carlo A. Gottardo[1], M. Bonaventura[1], S. Kolos[2],
Y. Heitz[1], E. Pozo Astigarraga[1], J. Roemer[1], V. Simola[1]

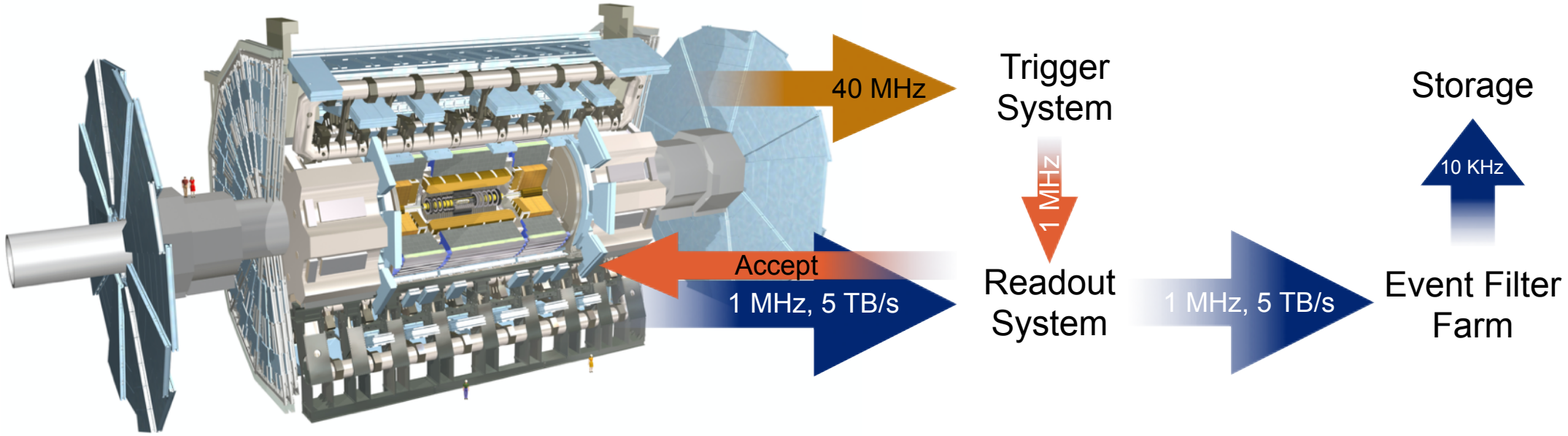
[1]CERN [2]University of California Irvine

CERN | NexTGen Next Generation Triggers

The ATLAS Experiment

At the HL-LHC, from 2030 onwards

- The Large Hadron Collider collides protons at a nominal 40 MHz rate
- ATLAS detects the collision products
- Data of interesting collisions stored at a 10 kHz rate

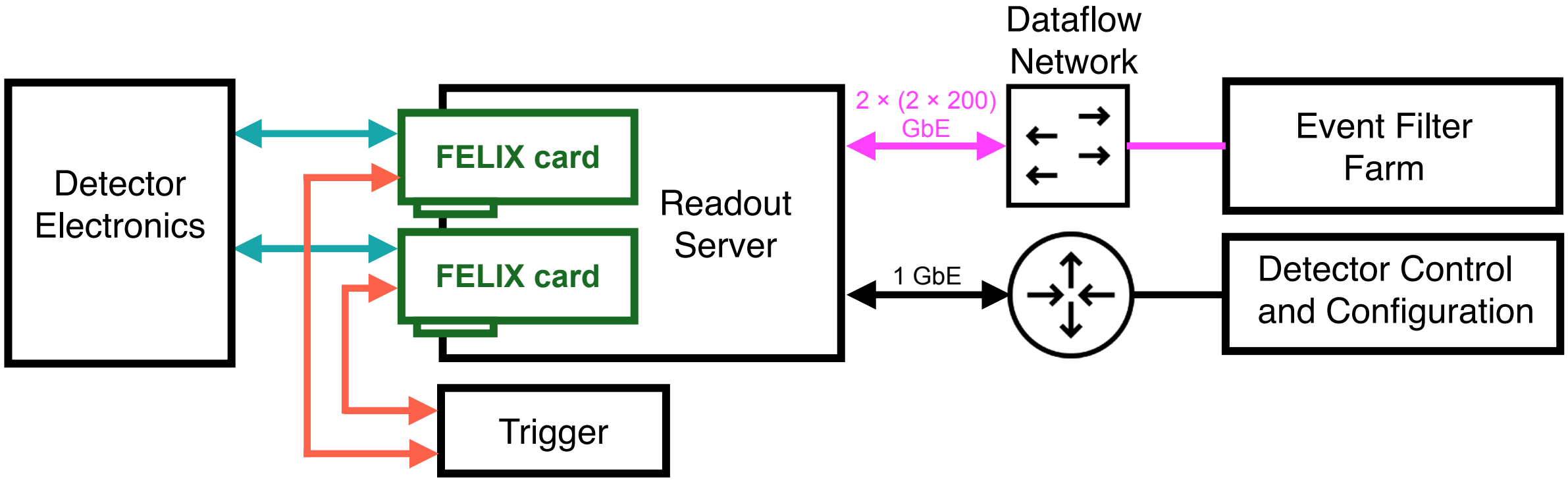


The Data Acquisition System

~15000 optical links with speed 4.8 (GBT), 9.6, 10.24 (lpGBT) and 25 Gb/s

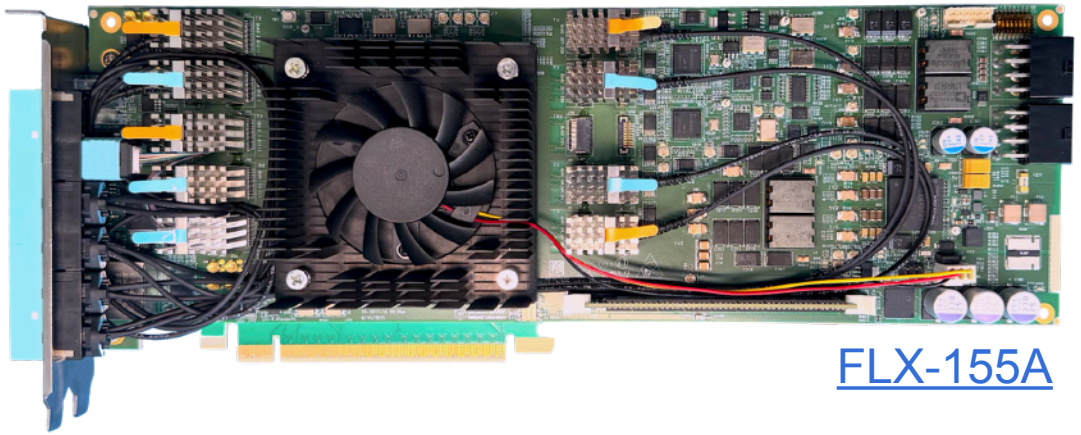
700 custom-designed PCIe “FELIX” cards in 300 Readout servers

O(1000) Event Filter servers

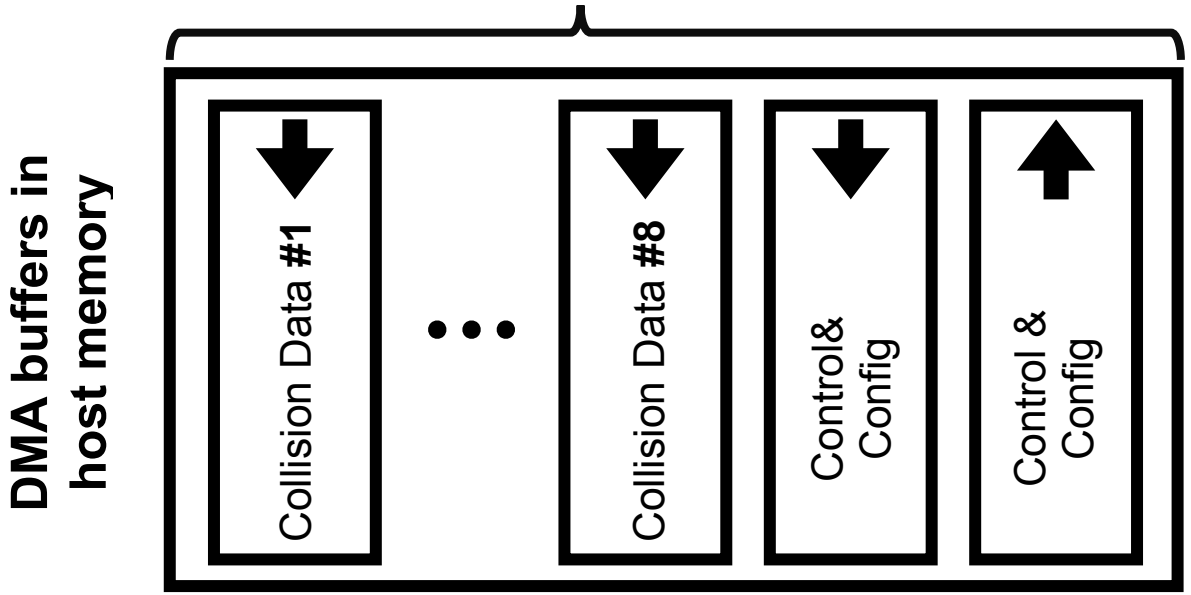


The FELIX Card

- AMD Versal Premium VP1552 FPGA
- 48 duplex optical links, 24 normally used
- PCIe Gen5×16
- Multiple Direct Memory Access (DMA) descriptors for write/read access of the host memory
- Data written to circular buffers in host memory in 1 kByte blocks, each associated with a single data stream or physical link



FLX-155A

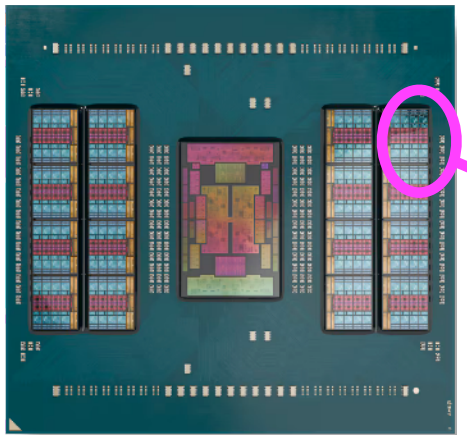


The Readout Server

- 4U form factor
  - up to ×4 FELIX cards,
  - ×2 (2 × 200) GbE interfaces

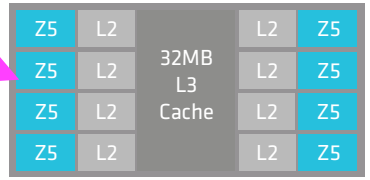
- ×2 AMD Epyc 9655 (96-cores)
  - **12 core complex dies (CCD) of**
  - **8 physical cores sharing L3 cache**

AMD Epyc 9005 series



AMD

CCD

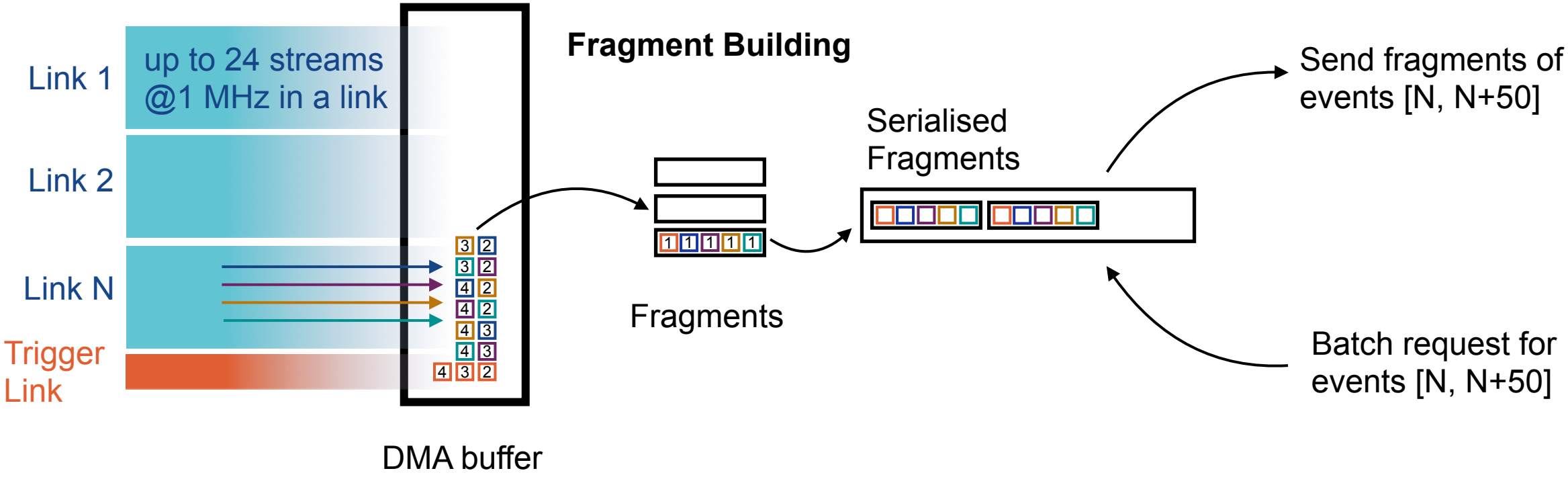


AMD

Picture by Supermicro
server loaded with 8 GPUs

Readout Software: Data Handler

- Extracts packets from 1 kB blocks in the FELIX DMA buffers
- Aggregates packets with same event number, applying optional sub-detector-specific processing
- Sends standardized fragments upon request via network



Data Handler Performance

Fragment Building Test

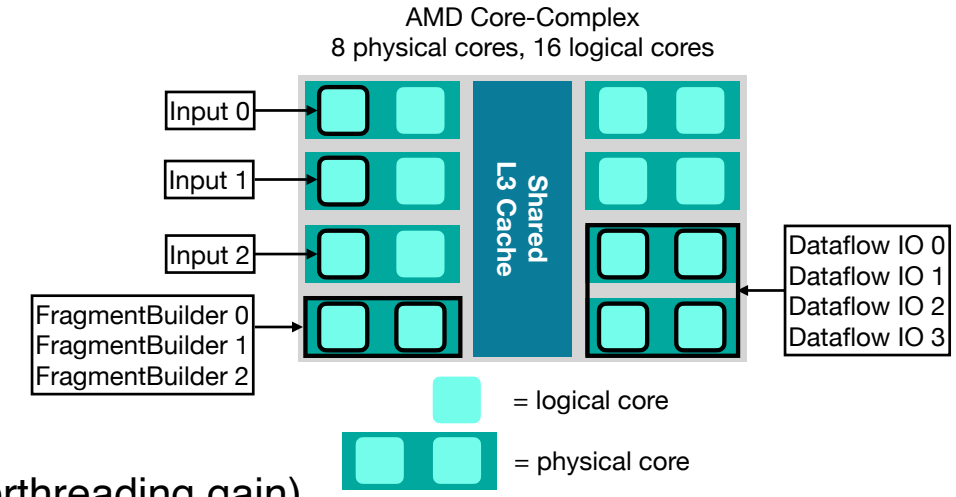
- 1 FELIX card, 336 streams, 1.1 MHz rate, 24-byte packet size (70 Gb/s)
- Streams distributed over 8 DMA buffers, 14 streams per fragment
- Fragments to remote node over 3000 TCP/IP connections upon request

Results

- Stable execution with 8 processes: ×1 Process for ×1 CCD for ×1 DMA buffer
- Threads pinned within the CCD

Test performed on Readout server with AMD 9654, 384 GB DDR5-4800, Nvidia Connect-X7
Deployment configuration may be different

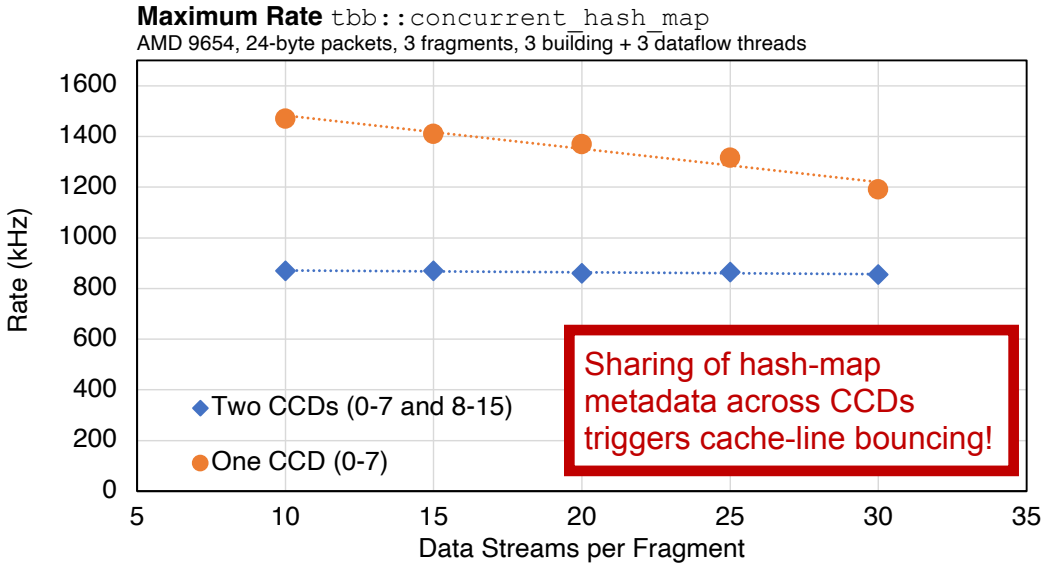
Thread	CPU use	Instances	Total CPU use
`Input`	66%	3	200%
`FragmentBuilder`	43%	3	130%
`DataflowIO`	35%	4	140%
DH process			470%
×8 DH processes			4740% / (9600% × hyperthreading gain)

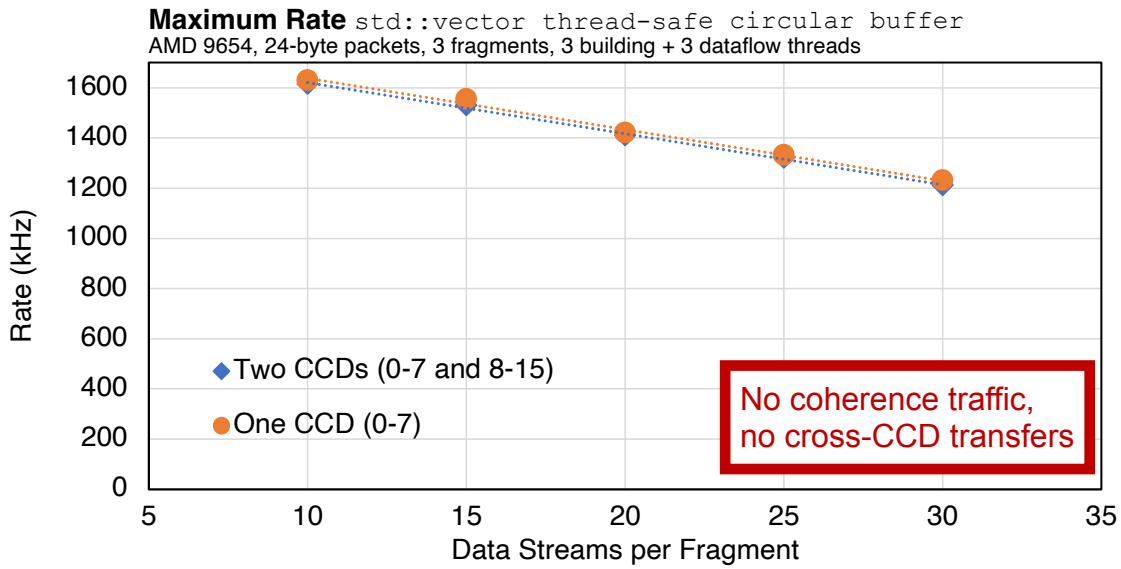


Cross-CCD communication

Input and aggregation threads within a CCD

- Output threads on **same vs other CCD**
- Two data structures to pass fragments across threads:
  `tbb::concurrent_hash_map`, lock-free circular buffer based on **`std::vector`**



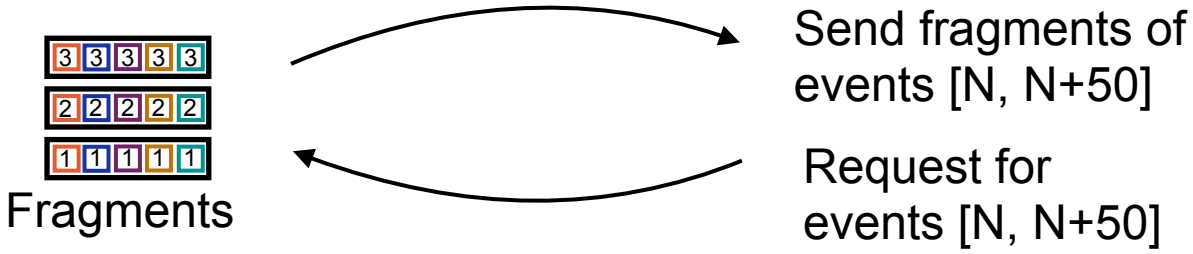


Lesson 1: High performance requires attention to the CPU architecture.

High Throughput with TCP/IP

Better performance with an extra copy

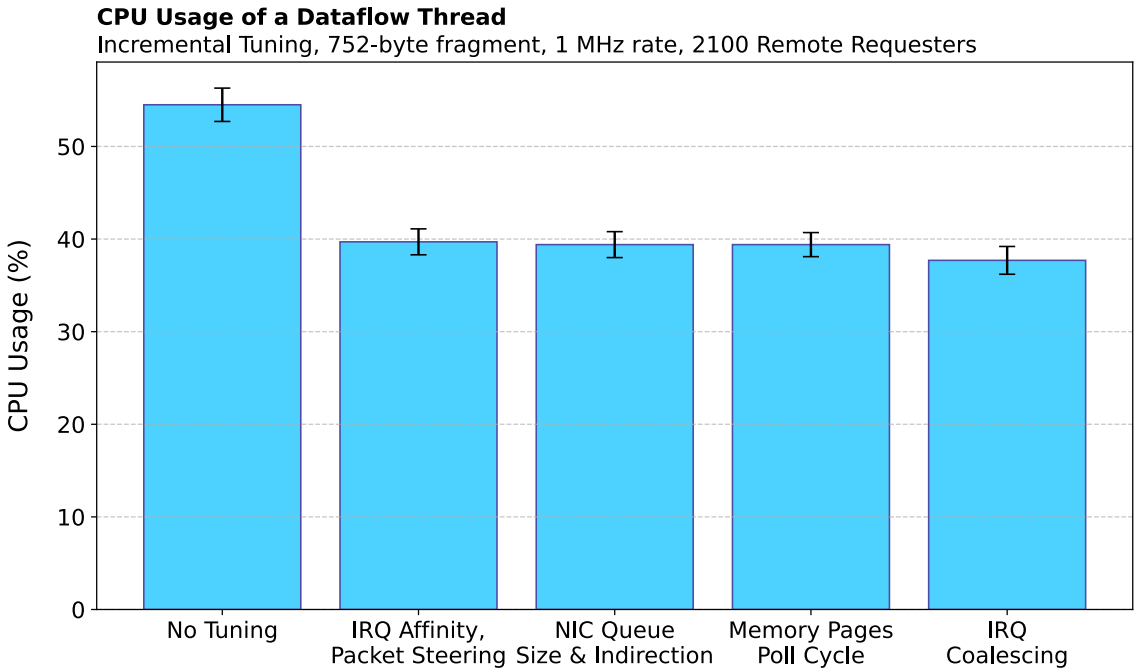
- DH receives a request for all fragments with event numbers within a range
- Output: 10^3 `iovec` entries, 20% O(1 kB), the rest O(10) byte header/trailer entries



- Small entries cause overhead in `tcp_sendmsg()`
- Performance improved **flattening IOV in user space with additional copy**
  - with copy: 54% CPU per thread sending 752 bytes × 1 MHz over 2100 connections
  - without copy: CPU saturated, 1 MHz not reached

Lesson 2: An extra copy can avoid expensive kernel work.

High Throughput with TCP/IP



Tuning of NIC and kernel parameters

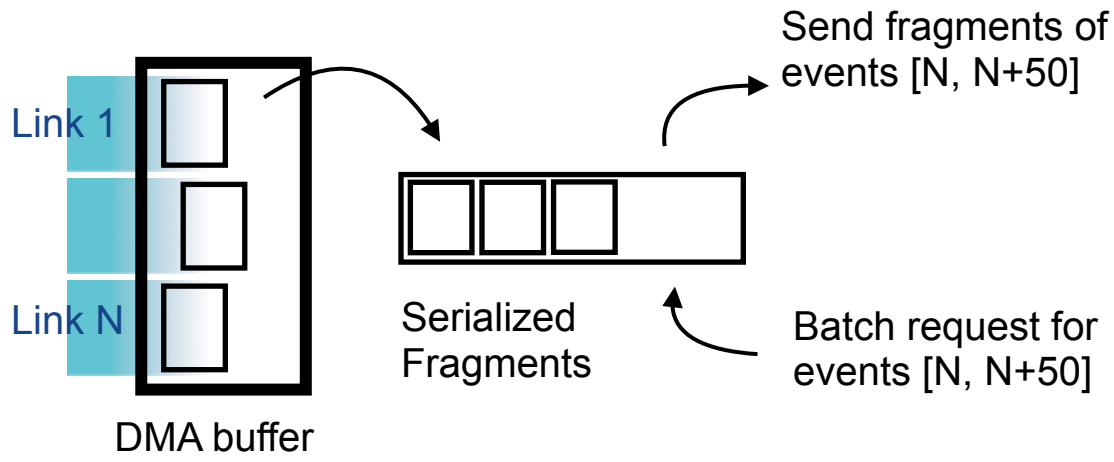
- Largest improvement from setting packet steering (RPS, XPS), flow steering (RFS), and IRQ affinity to the cores on which Dataflow threads are pinned
- 15% CPU use saved per Dataflow thread
- Up to 96 threads running on same CPU (8 threads × 12 DH applications)

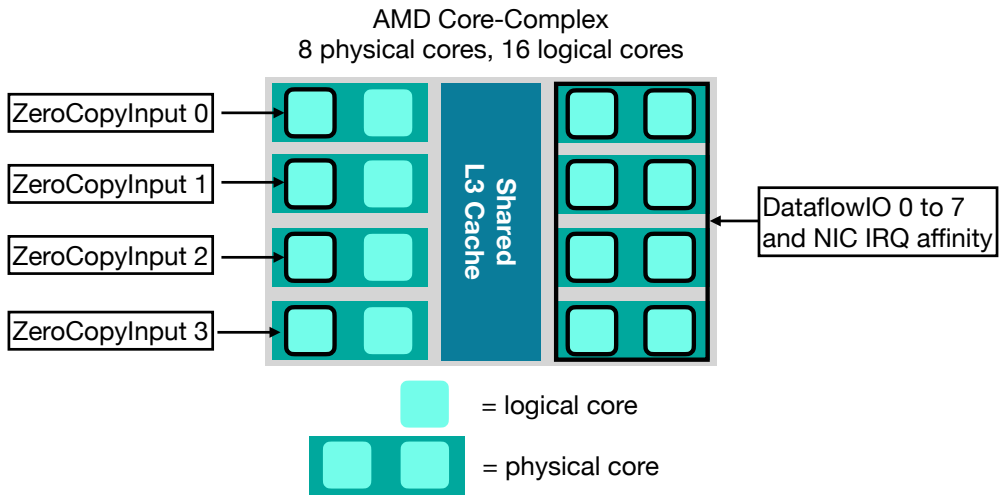
Lesson 3: Tuning the NIC and kernel to match the threading model can be worthwhile.

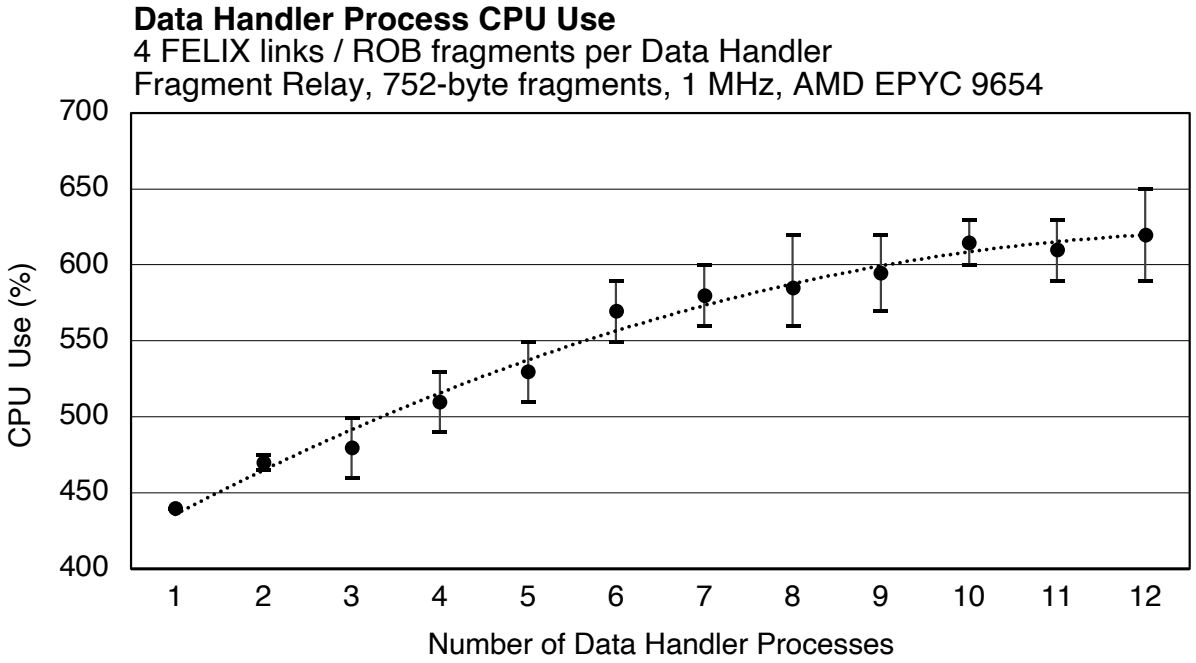
Data Handler Performance

Fragment Relay Test

- Some FELIX links transport pre-built fragments, Data Handler can just “relay” them
- Test with 48 streams, 752-byte fragments, 1 MHz (288 Gb/s)
- 12 processes per CPU, sub-optimal scalability traced to the network stack







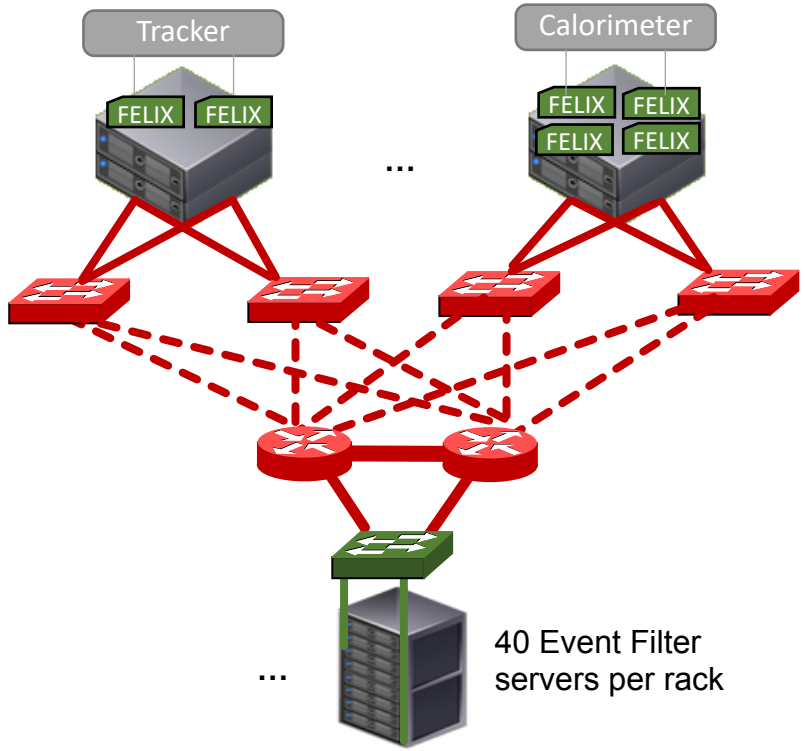
RDMA (RoCEv2)

- Network card can transfer data using DMA, bypassing the kernel
- Netio3 library, developed by ATLAS Readout, uses libfabric + ibverbs

Challenge 1: Incast

All Readout nodes send fragments for a given batch of events to one Event Filter node

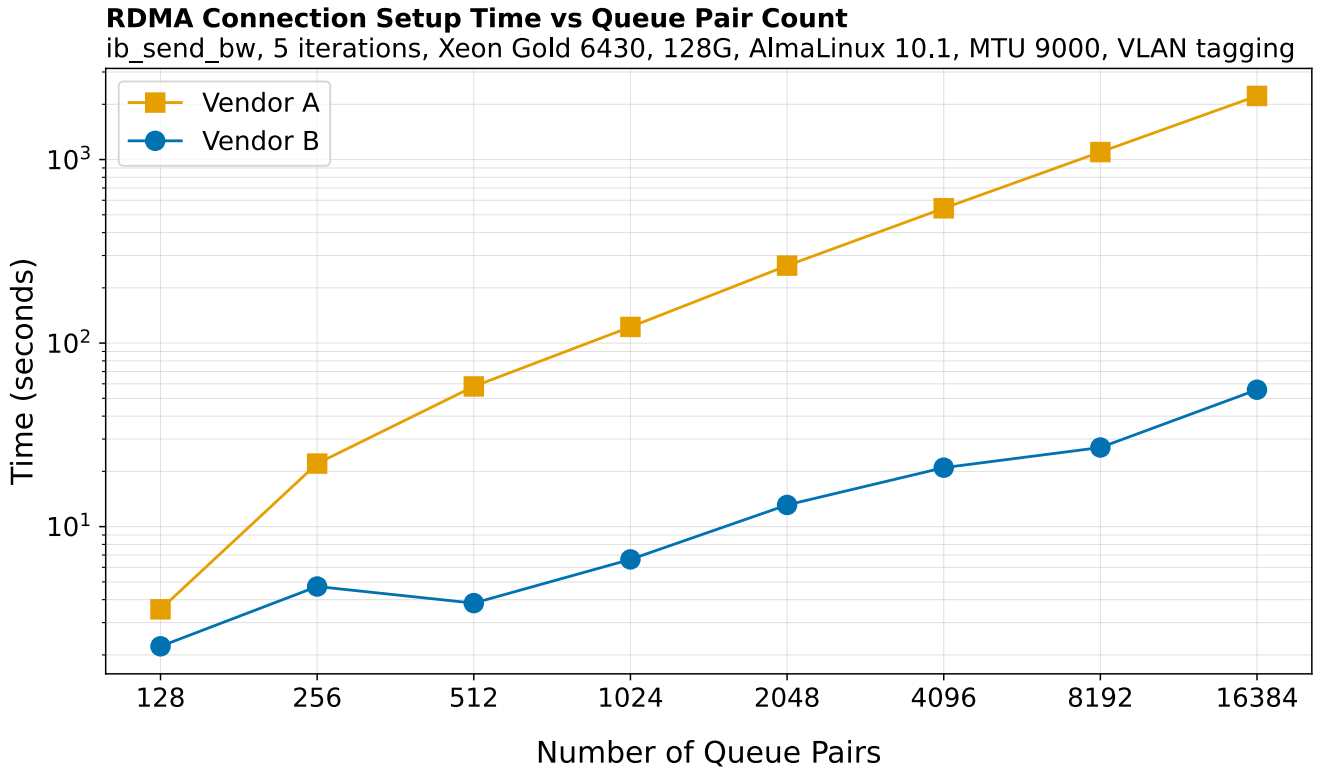
- Traffic shaping in software
- RDMA requires a lossless network, must tune switch and NIC parameters:
  - Explicit Congestion Notification (4 parameters, huge phase space)
  - Priority Flow Control



RDMA (RoCEv2)

Challenge 2: Number of Connections

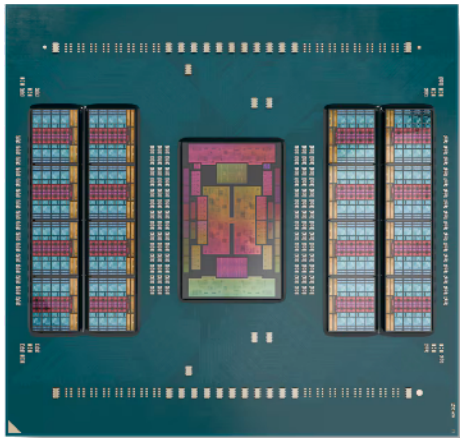
- O(10^4) connections per Readout server
  (10 applications × 10^3 Event Filter servers)
- Measured large overhead in establishing connections
- Consistent measurement results for two switches

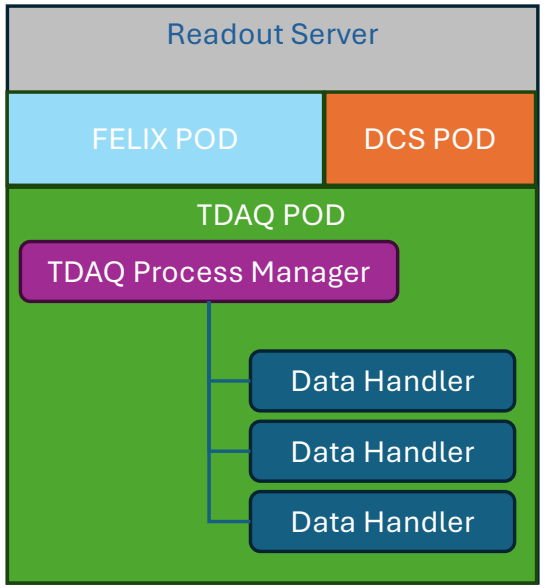


Lesson 4: RDMA not easy to deploy for all workloads.

Software Management

- Many heavy I/O applications sharing CPU and memory
- One runaway process can take down the entire node
- Detector control and monitoring traffic must remain uninterrupted
- Strict resource isolation is mandatory





Kubernetes

- automatic lifetime management of applications with manual override
- resource management and isolation
- out-of-the-box monitoring
- no indication of performance drop using containers, including RDMA

Lesson 5: Large multi-process workloads require resource isolation.

Summary

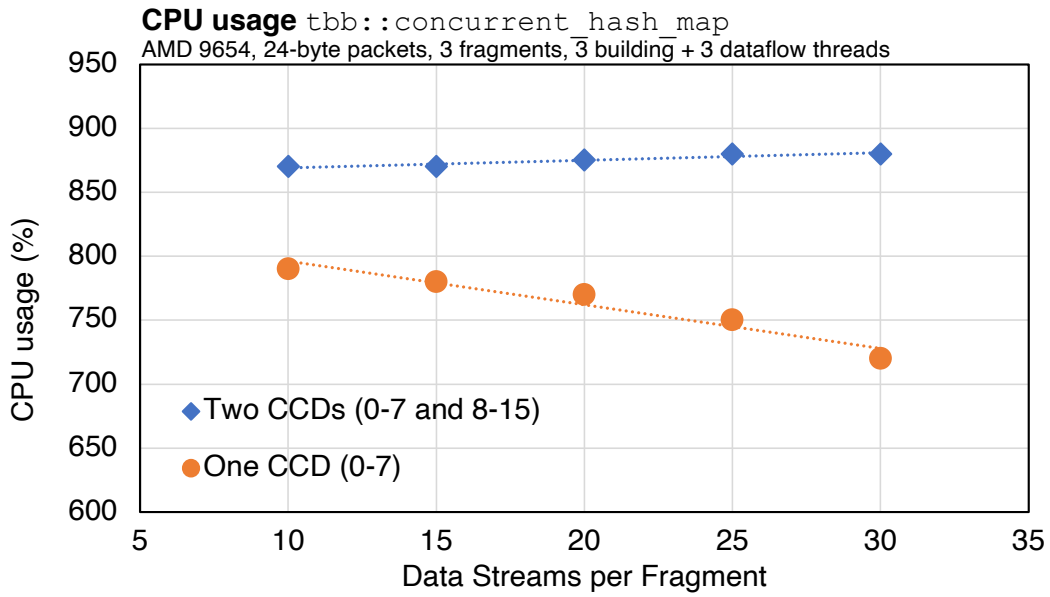
Lessons Learned

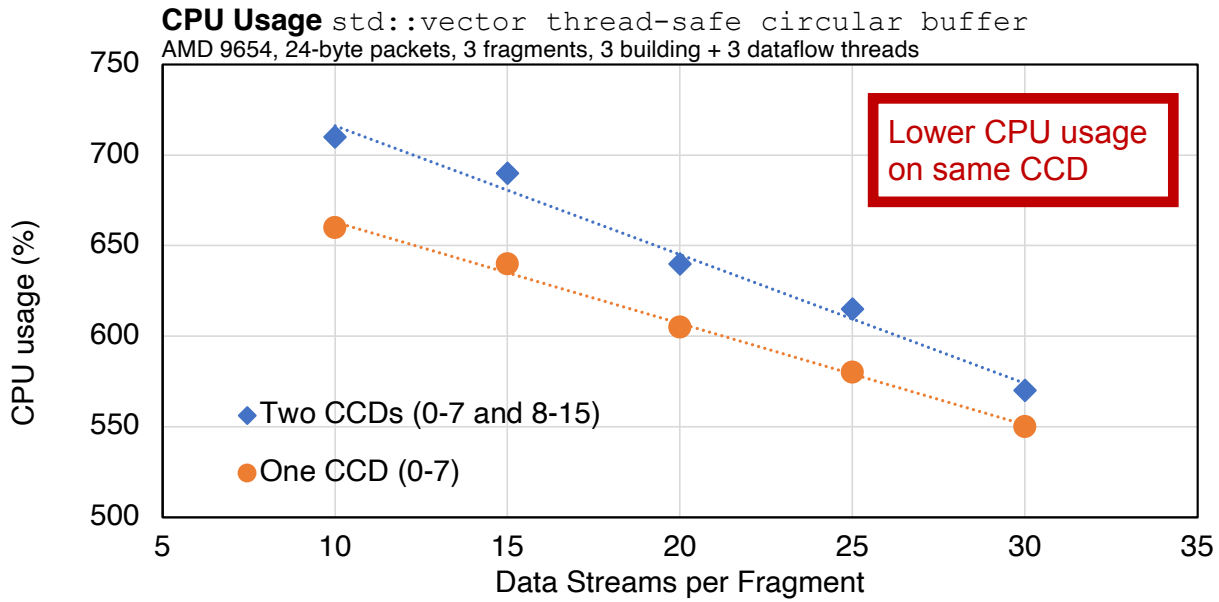
1. High performance requires attention to the CPU architecture.
2. An extra copy can avoid expensive kernel work.
3. Tuning the NIC and kernel to match the threading model can be worthwhile.
4. RDMA not easy to deploy for all workloads.
5. Large multi-process workloads require resource isolation.


CERN
NexTGen
Next Generation Triggers

BACKUP

Cross-CCD communication





- Total CPU use can decrease with increasing number of streams per fragment because
  - Data aggregation threads fully utilize CPU resources in all configurations
  - CPU use of Dataflow threads proportional to the event rate
  - more streams → lower rate

TCP/IP Tuning

IRQ Affinity & Packet Steering

```
RPS echo <core mask> > /sys/class/net/eth4/queues/rx-${i}/rps_cpus
XPS core <core mask> > /sys/class/net/eth4/queues/tx-${i}/xps_cpus
RFS
  echo 131072 > /proc/sys/net/core/rps_sock_flow_entries;
  echo 4096 > /sys/class/net/eth4/queues/rx-${i}/rps_flow_cnt
```

Queue Size & Indirection

```
ethtool -L eth4 combined 32
ethtool -X eth4 equal 32
```

Memory and Budget Tuning

```
sysctl vm.percpu_pagelist_high_fraction= 0 --> 96
sysctl net.core.netdev_budget_usecs=2000 --> 8000
sysctl net.core.netdev_budget=300 --> 1200
```

IRQ Coalescing

```
ethtool -C eth4 adaptive-rx off adaptive-tx off rx-usecs 256 rx-frames 512 tx-usecs 64 tx-frames 512
```