2024 OFA Virtual Workshop

RecoNIC: RDMA-enabled Compute Offloading on FPGA-based SmartNIC

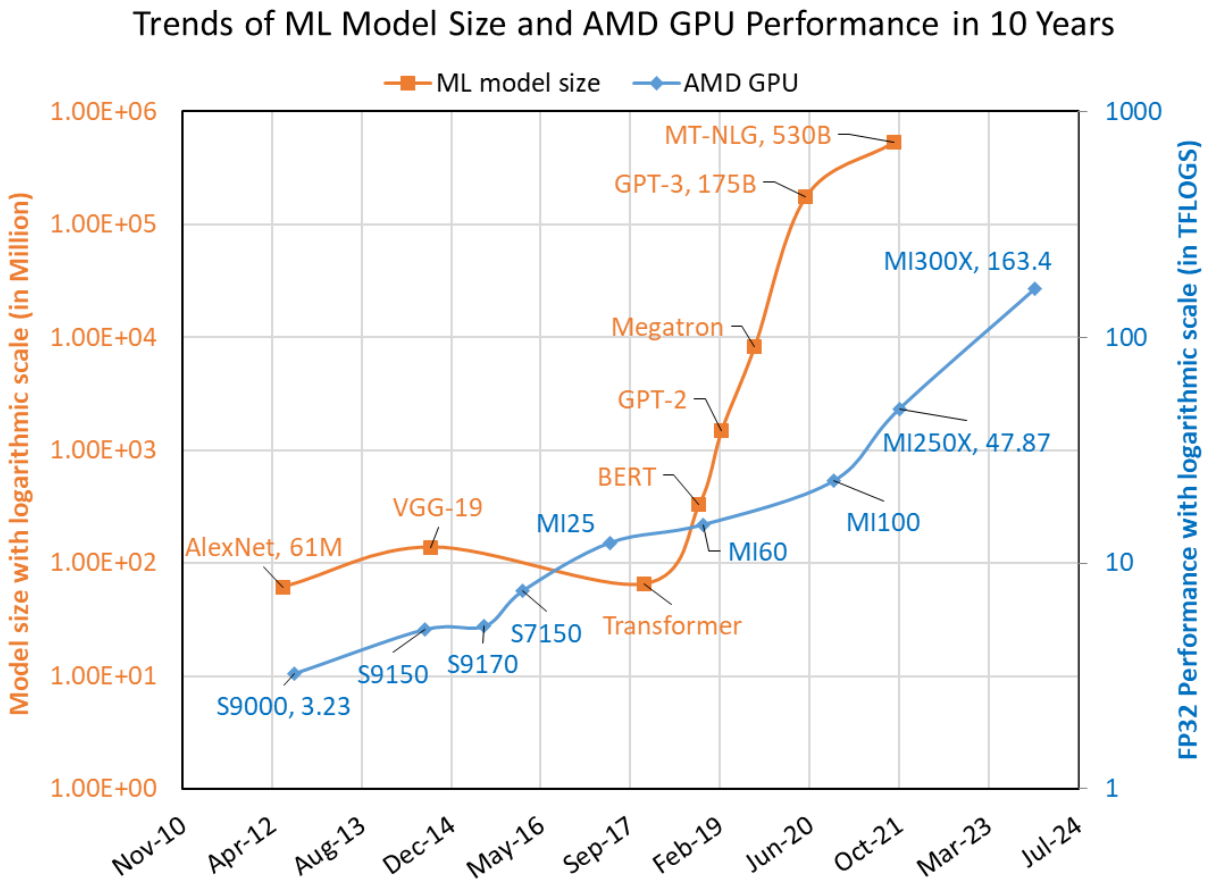
Guanwen (Henry) Zhong, Senior Researcher

AMD Research and Advanced Development

ML model size and GPU performance over the past 10 years

- ML model size over 10 years: ~8600x
  - Exponential growth from 61M in 2012 to 530B in 2021
- AMD GPU performance over 10 years: ~50x

- **ML model size has outpaced the growth in single GPU performance over the past 10 years**



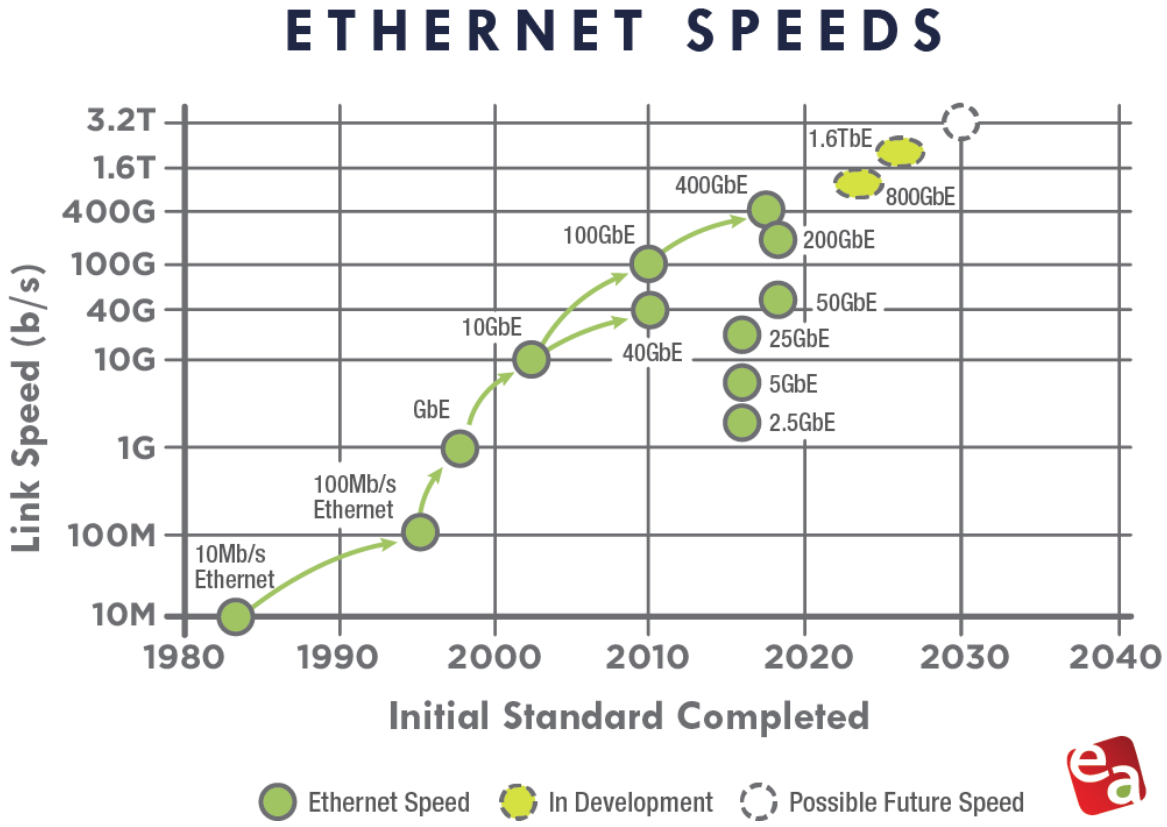
AMD

Ethernet speeds over the past 40 years

- ML model size over 10 years: ~8600x
- AMD GPU performance over 10 years: ~50x

- Ethernet speed over 10 years: ~10x
  - Significantly slower than GPU advancement and ML model size growth

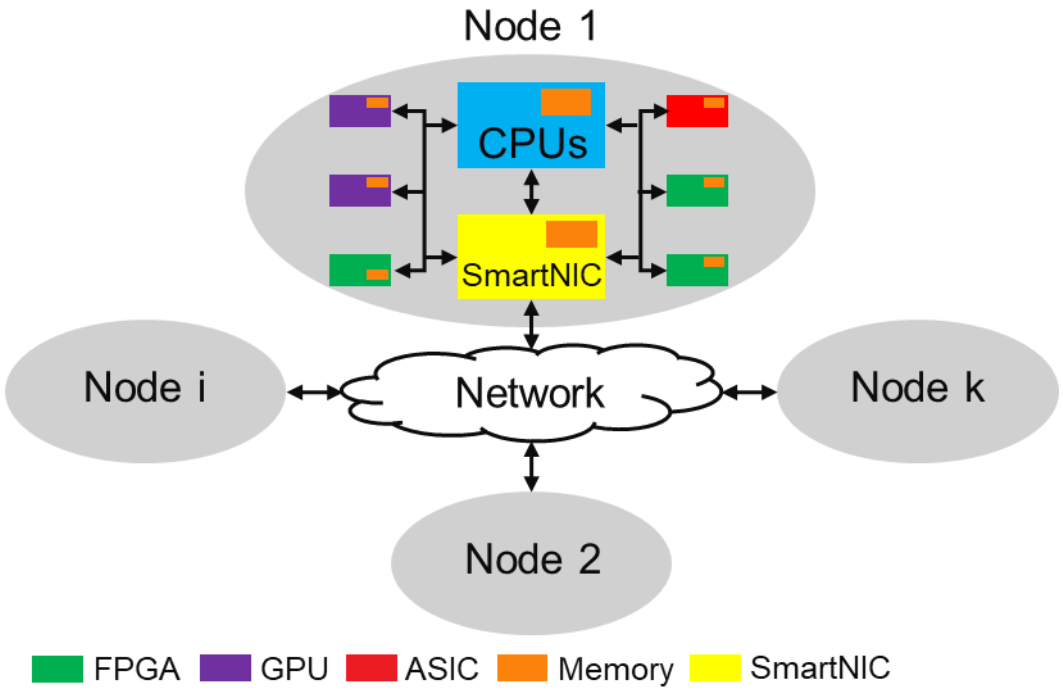
- Emergence of scale-out architectures
  - A sea of heterogeneous nodes connected via the high-speed network



*Source from Ethernet Roadmap 2023 by Ethernet Alliance

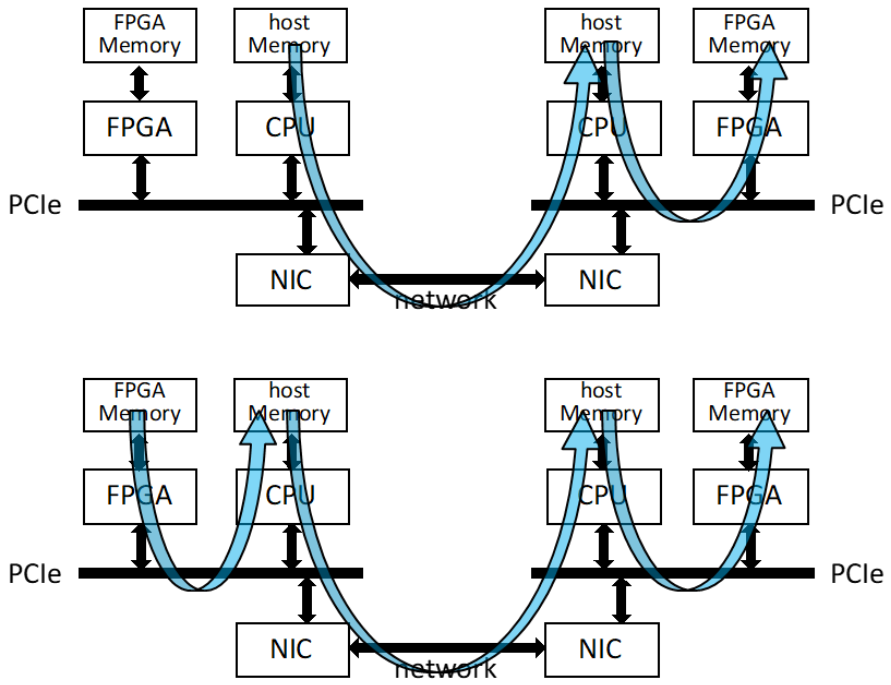
Emergence of scale-out architectures

- A sea of nodes connected via high-speed and low-latency network interconnect
- Heterogeneity within a node
  - CPUs, FPGAs, GPUs, ASICs (such as TPUs), SmartNICs …
- SmartNIC acts as an intermediate hub for various components
  - Regular “NIC” functions: protocol handling, vSwitch, crypto, …
  - Value-add “NIC” functions: TOE, RDMA, security, telemetry, …
  - Upper layer processing: transport-layer and above, accelerate streaming and lookaside applications
- High-speed and low-latency networking: RDMA

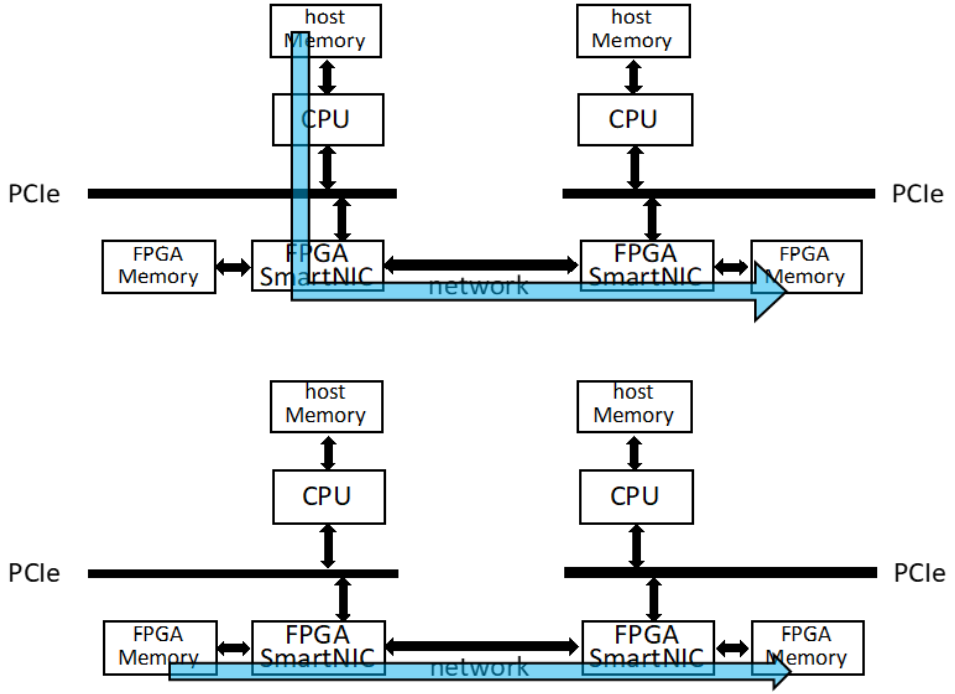


Data communication in scale-out setups

- Traditional way incurs multiple data copies



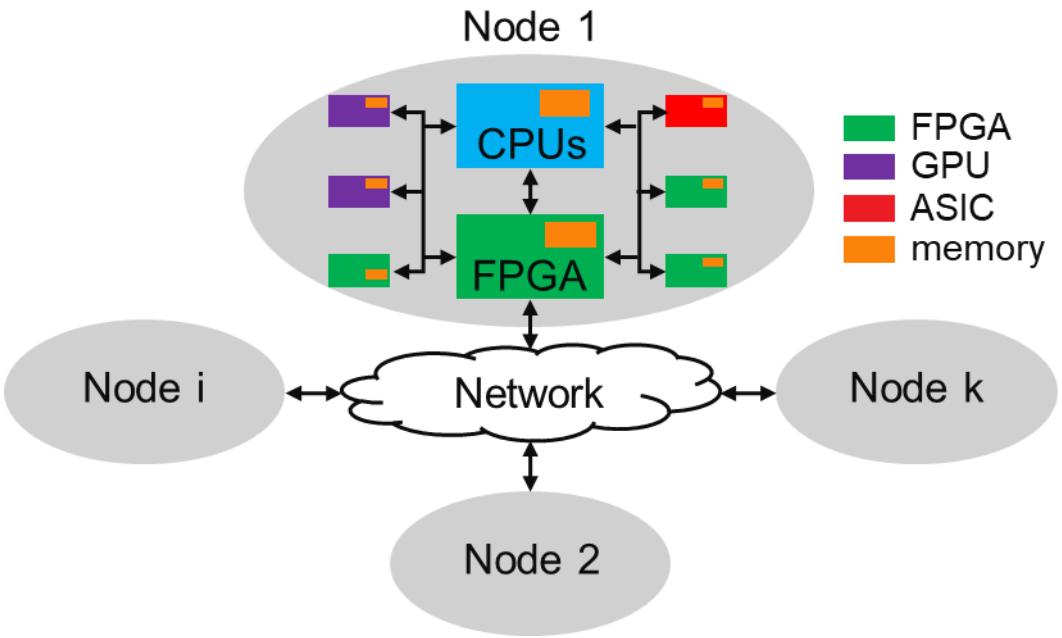
- Programmable SmartNIC-enabled system – zero copy



1. Enable direct memory access among peers
2. Bring data as close to compute as possible

What kind of programmable SmartNIC features do we need in a scale-out system?

- Normal network packets
  - TCP, UDP, DCCP, SCTP, QUIC, …
- Remote direct memory access (RDMA)
  - RoCEv2 packets
  - Shared by host, GPU and FPGA
- Bring data as close to accelerators as possible for fast and adaptable hardware acceleration
  - Compute logic for general applications inside SmartNIC
    - Streaming computation
    - Lookaside computation

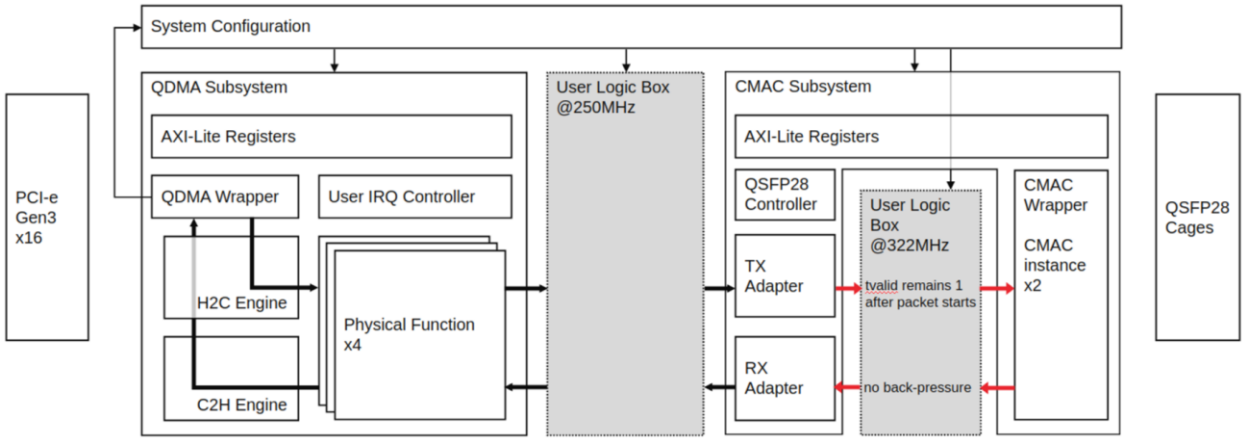


Why RecoNIC?

- RDMA is the de facto standard for high-speed data communication for ML & HPC applications

- Basic Adaptive SmartNICs without transport-layer offloading engine
  - OpenNIC [1]
  - Corundum [2]

Stand-alone transport-layer offloading engines
- Catapult LTL engine [3]
- TCP offloading engine [4] and RDMA [5] from ETH Zurich
- ERNIC [6]
  - An RDMA engine from AMD
  - RoCEv2 implementation



OpenNIC [1]

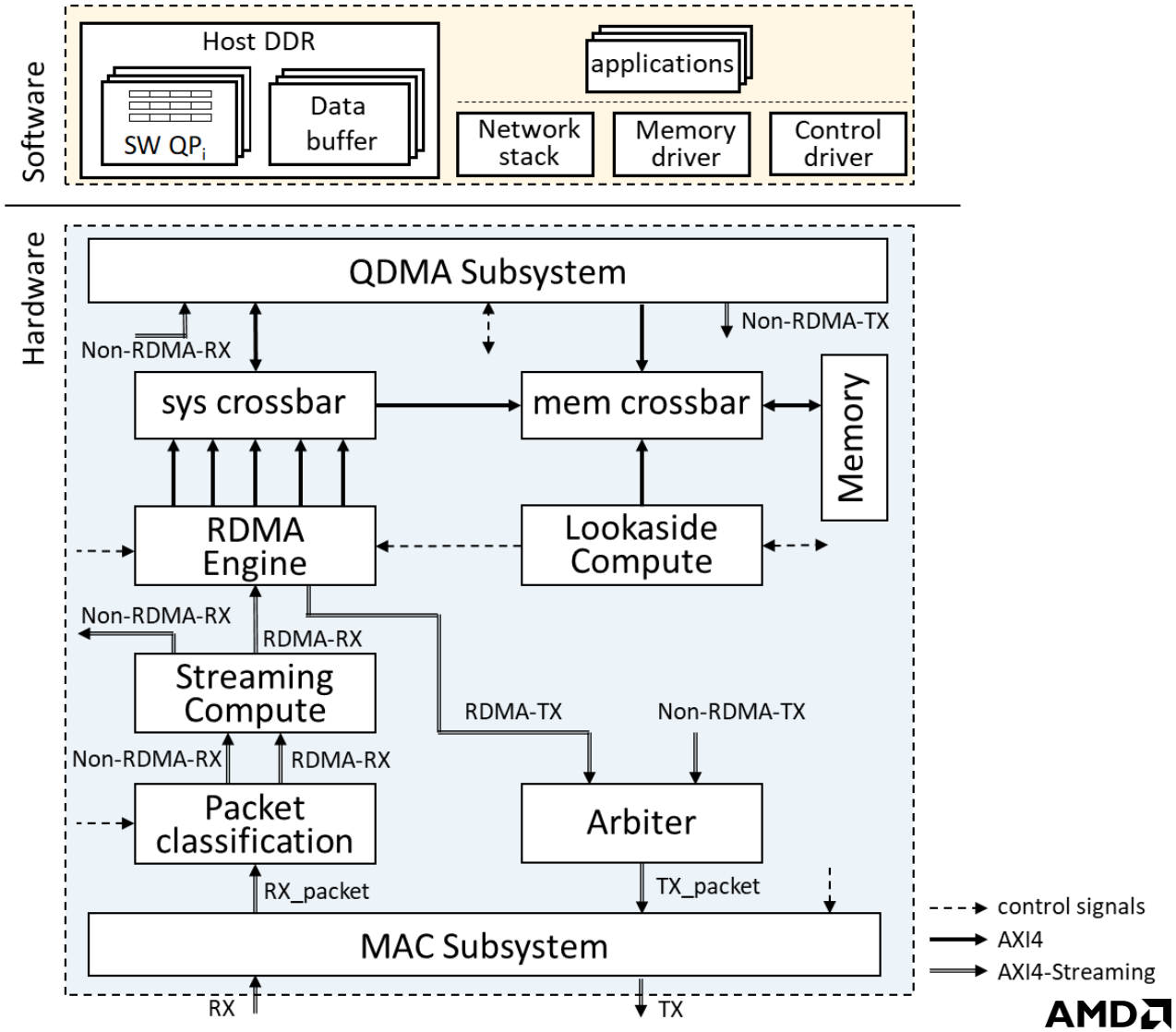
There is no open-sourced RDMA-enabled adaptive SmartNIC platform

RecoNIC: RDMA-enabled Compute Offloading on SmartNIC

- An open-source 100Gb/s FPGA-based SmartNIC infrastructure/testbed with RDMA and compute offloading
- To enable scale-out heterogeneous systems
- To enable direct memory access among network-connected peers
- To bring data as close to various types of accelerators as possible

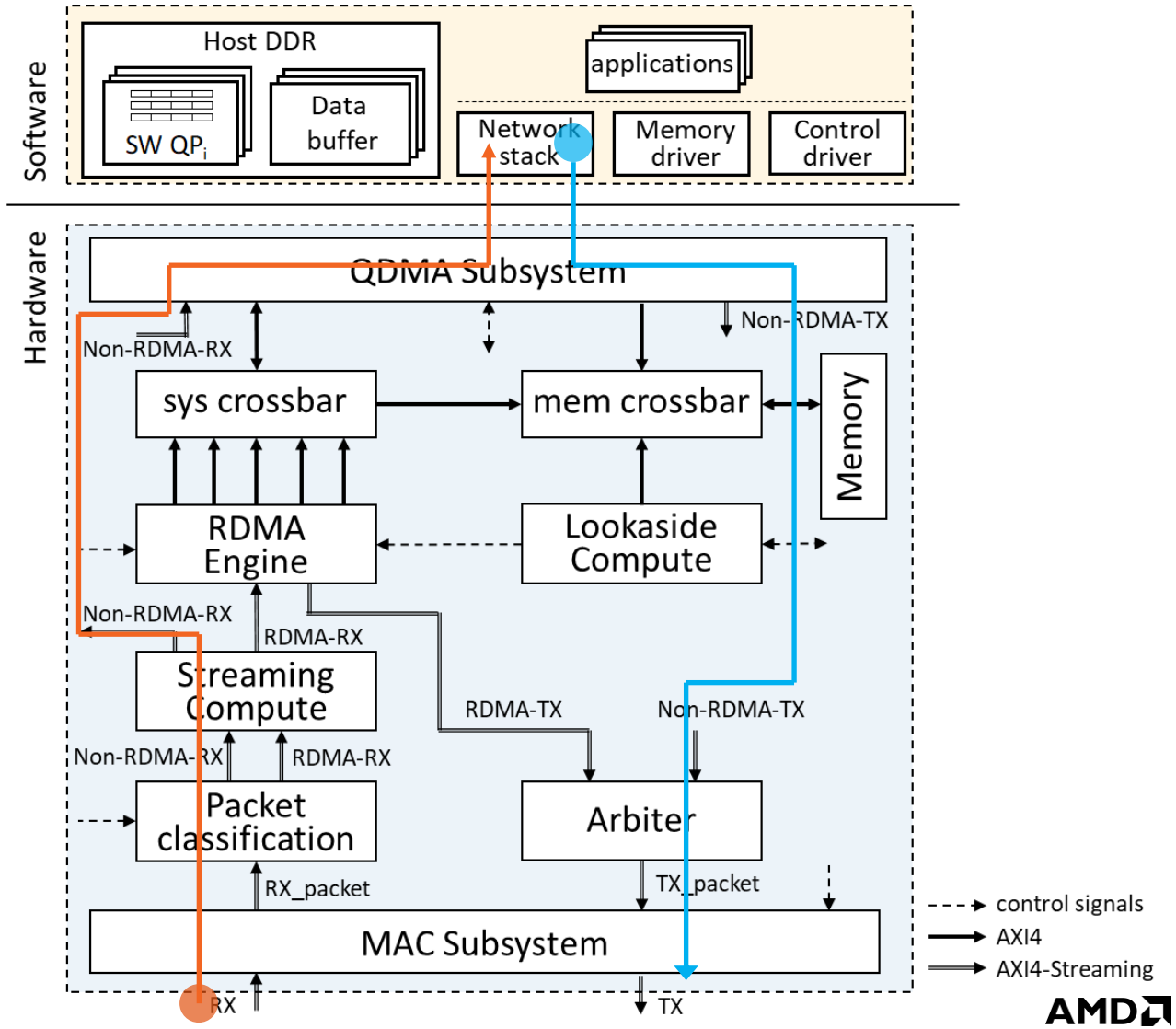
The RecoNIC system architecture

- A hardware shell
  - RDMA engine: shared by host and accelerators
  - Compute boxes for streaming and lookaside acceleration
  - Packet classification
  - Auxiliary components
    - MAC, QDMA, crossbars, arbiter
- Software stacks
  - Network stacks
    - non-RDMA traffic such as TCP/IP, UDP/IP and ARP
    - User-space RDMA APIs
  - Memory driver
    - data transfers between host and device memory
  - Control driver
    - Register configuration control
    - Compute control



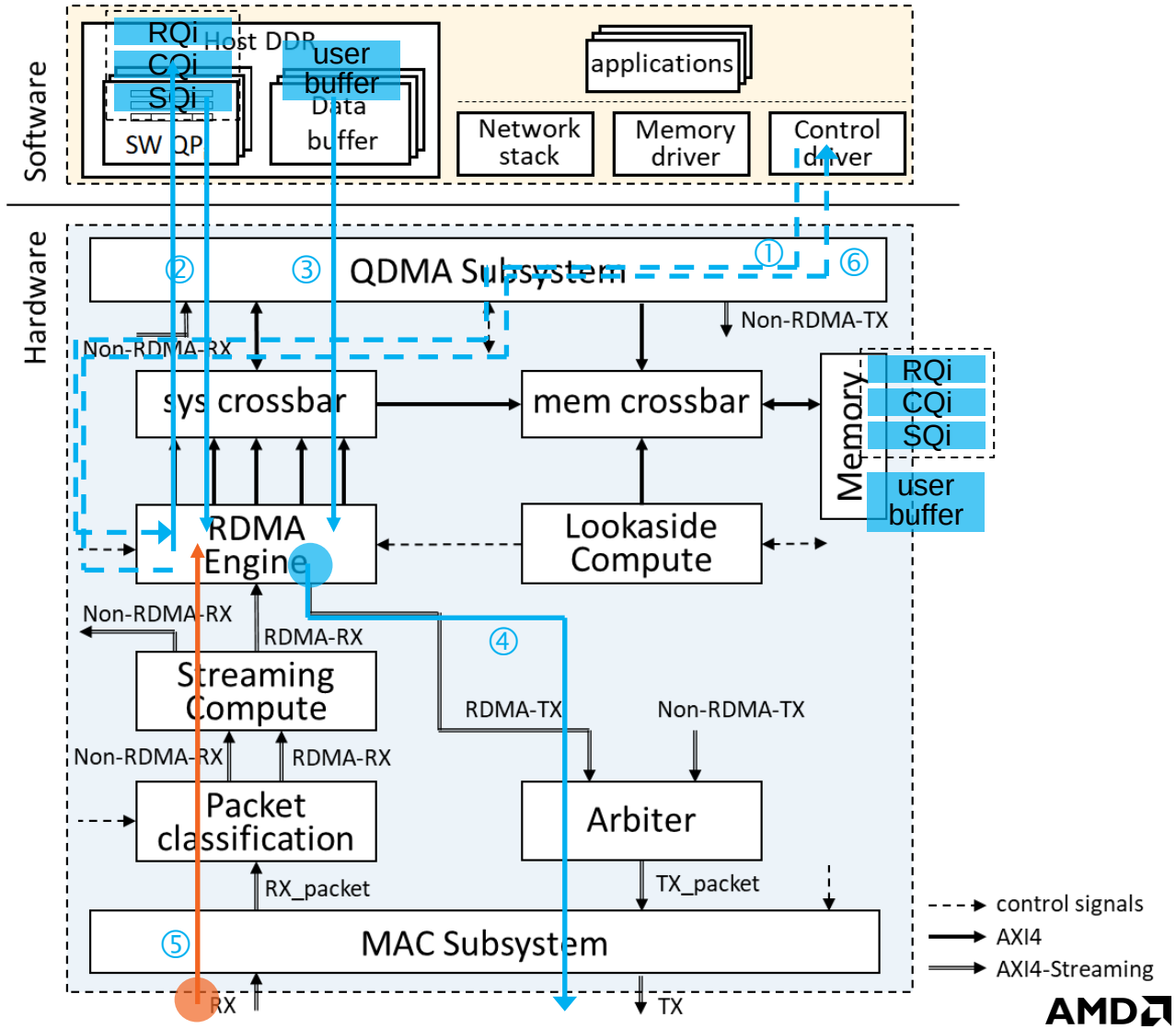
The RecoNIC network flow

- Non-RDMA traffic
  - TX path: Network stack -> QDMA subsystem TX -> Arbiter -> MAC subsystem TX
  - RX path: MAC subsystem RX -> Packet classification -> Streaming compute -> QDMA subsystem RX -> Network stack



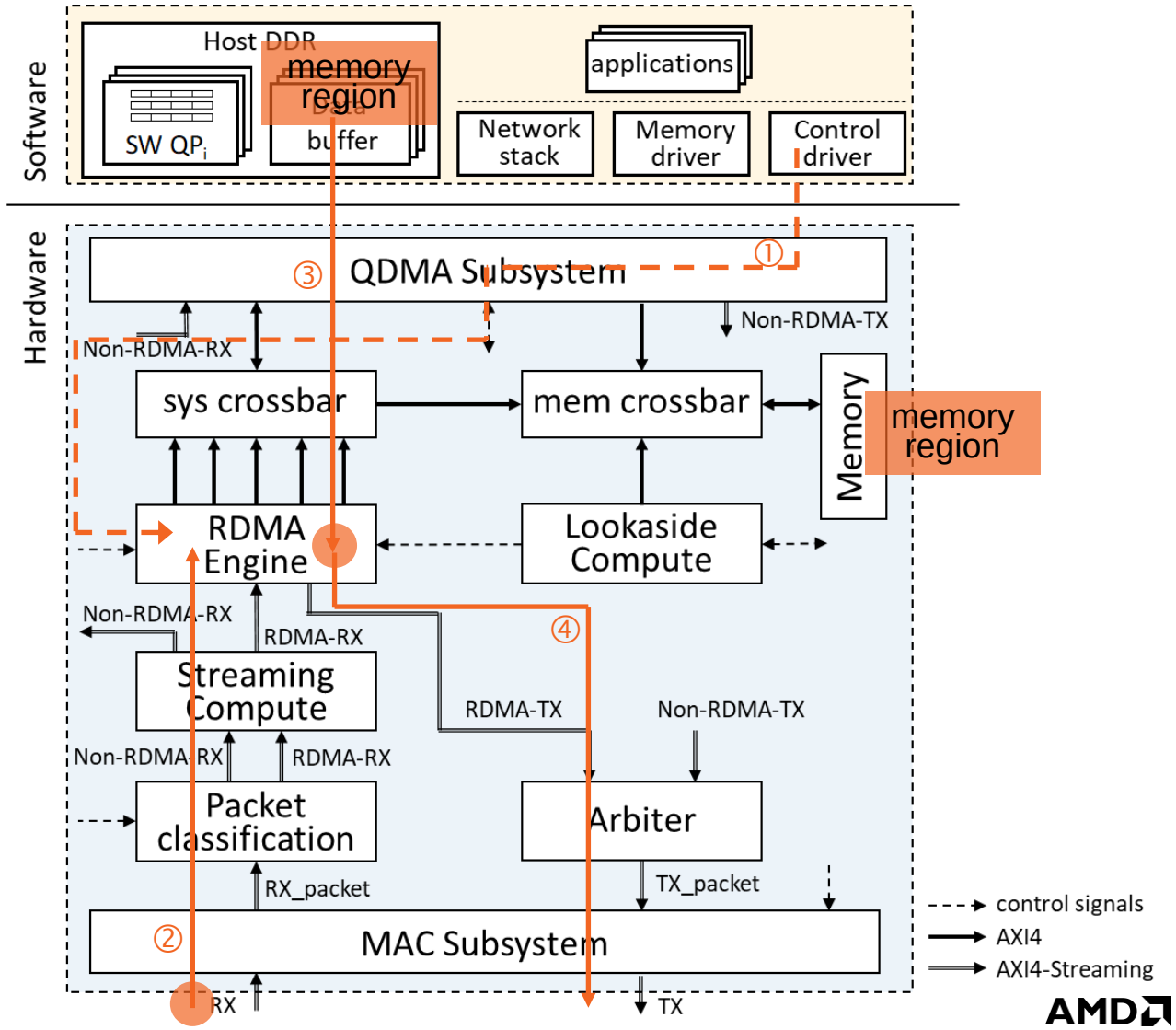
The RecoNIC network flow

- QP and data buffer can be declared either in host or device memory
- RDMA traffic
  - TX path (RDMA write as an example)
    - ① Host declares QP, configures RDMA and rings SQ doorbell
    - ② RDMA engine fetches WQE from SQ
    - ③ RDMA engine fetches payload from user buffer and constructs RDMA write packets
    - ④ RDMA engine sends RDMA write packets
    - ⑤ RDMA engine updates CQ when receiving RDMA write acknowledgement packets
    - ⑥ Host polls CQ doorbell to detect when RDMA write is done



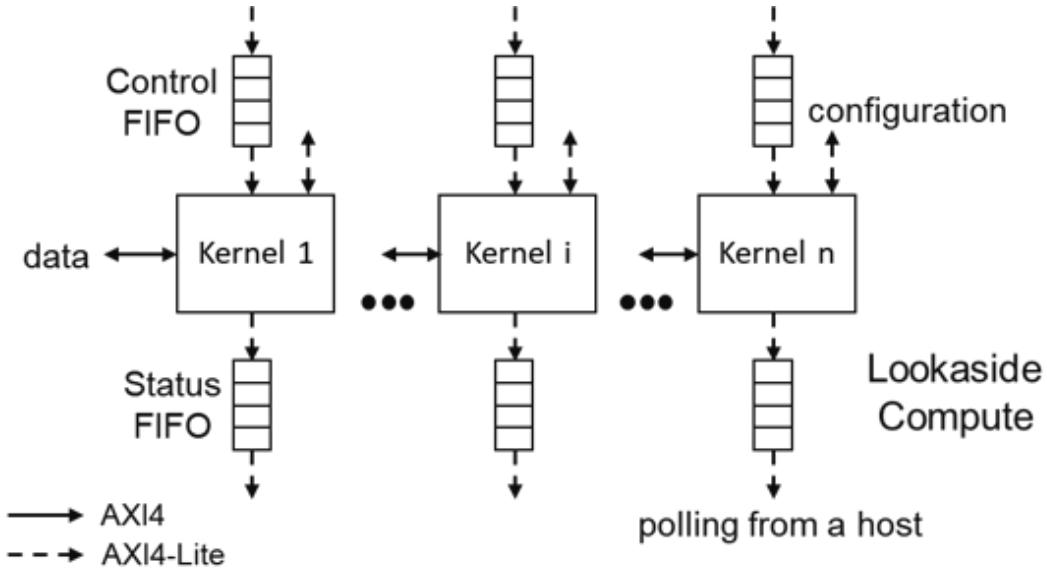
The RecoNIC network flow

- RDMA traffic
  - RX path (RDMA read response as an example)
    - ① Host registers memory region
    - ② RDMA engine waits for RDMA read request from a remote peer
    - ③ RDMA engine validates read requests, fetches payload and constructs RDMA read response packets
    - ④ RDMA engine sends RDMA read response packets
- Memory region can be declared either in host or device memory



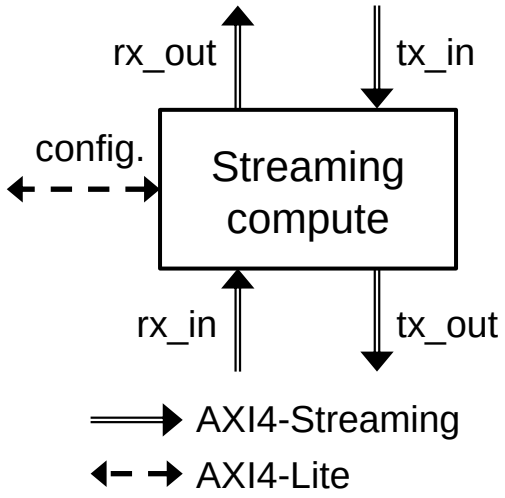
Hardware component: lookaside compute

- Compute acceleration over data stored in device memory
- Datapath interface: AXI4 memory mapped, access data from either device memory or host memory
- Register control interface: AXI4-Lite
- Compute control: two FIFOs
  - Control FIFO: stores user-defined compute control commands
  - Status FIFO: stores completion signals such as kernel ID, job ID, …
- Kernels can trigger RDMA operations
- Potential use cases:
  - Applications required to wait for data from multiple peers before computation
- Supports HLS and RTL implementations

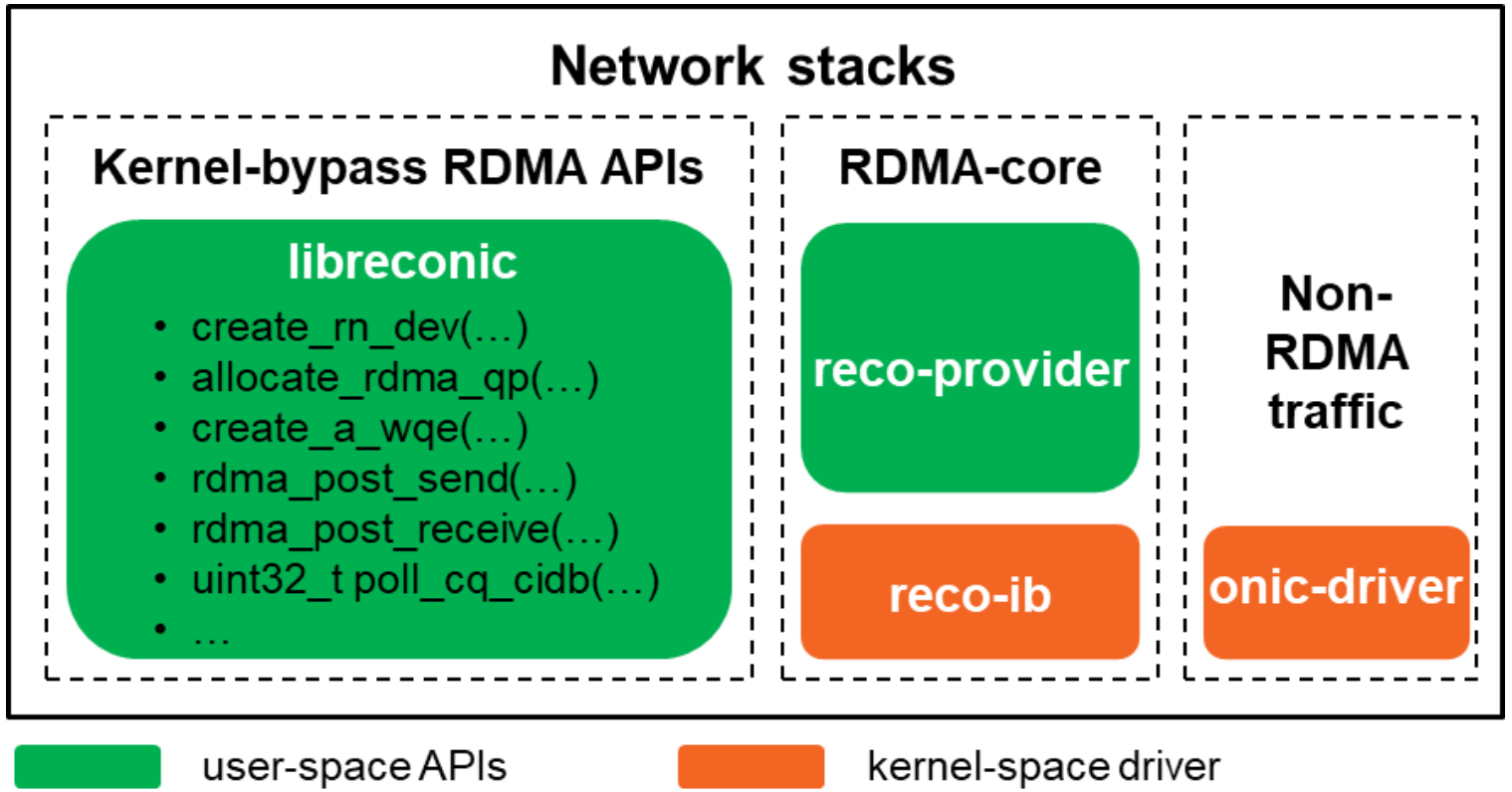


Hardware component: streaming compute

- Compute acceleration over network data at line-rate
- Datapath interface: AXI4-Streaming
  - Network data
- Control-path interface: AXI4-Lite for internal registers
- Potential use cases
  - Packet processing applications (e.g., packet classification, protocol handling, forwarding, crypto, checksum offloading, security, upper-layer processing, …)
  - Telemetry
  - line-rate application processing such as in-network aggregation
  - …
- Supports Vitis Networking P4, HLS and RTL implementations



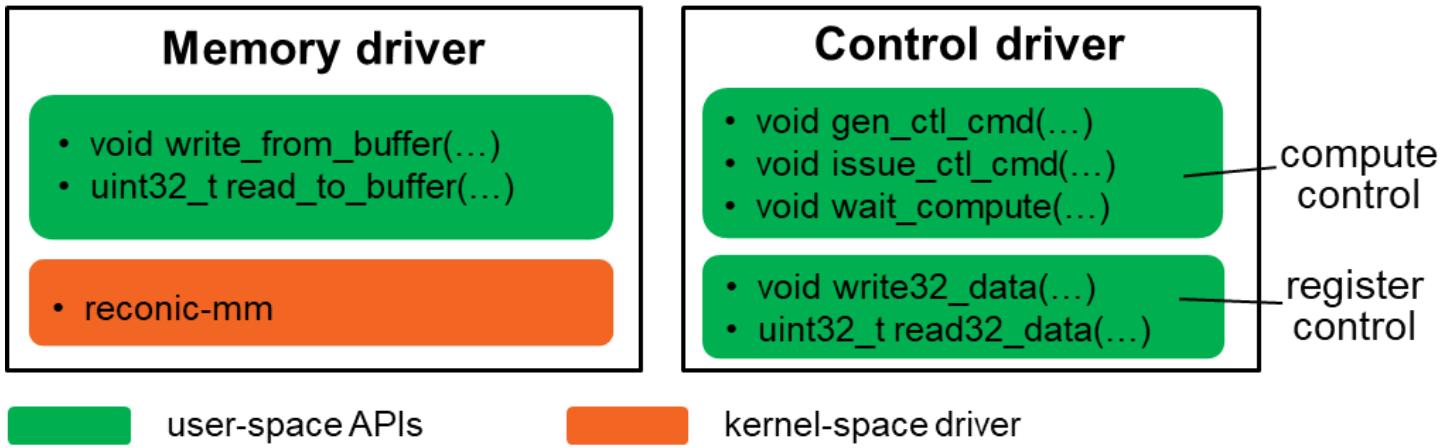
RecoNIC software stacks



- Non-RDMA traffic: onic-driver
- RDMA traffic
  - Kernel-bypass RDMA APIs: libreconic
  - RDMA-core library (In-progress): reco-provider and reco-ib

RecoNIC software stacks

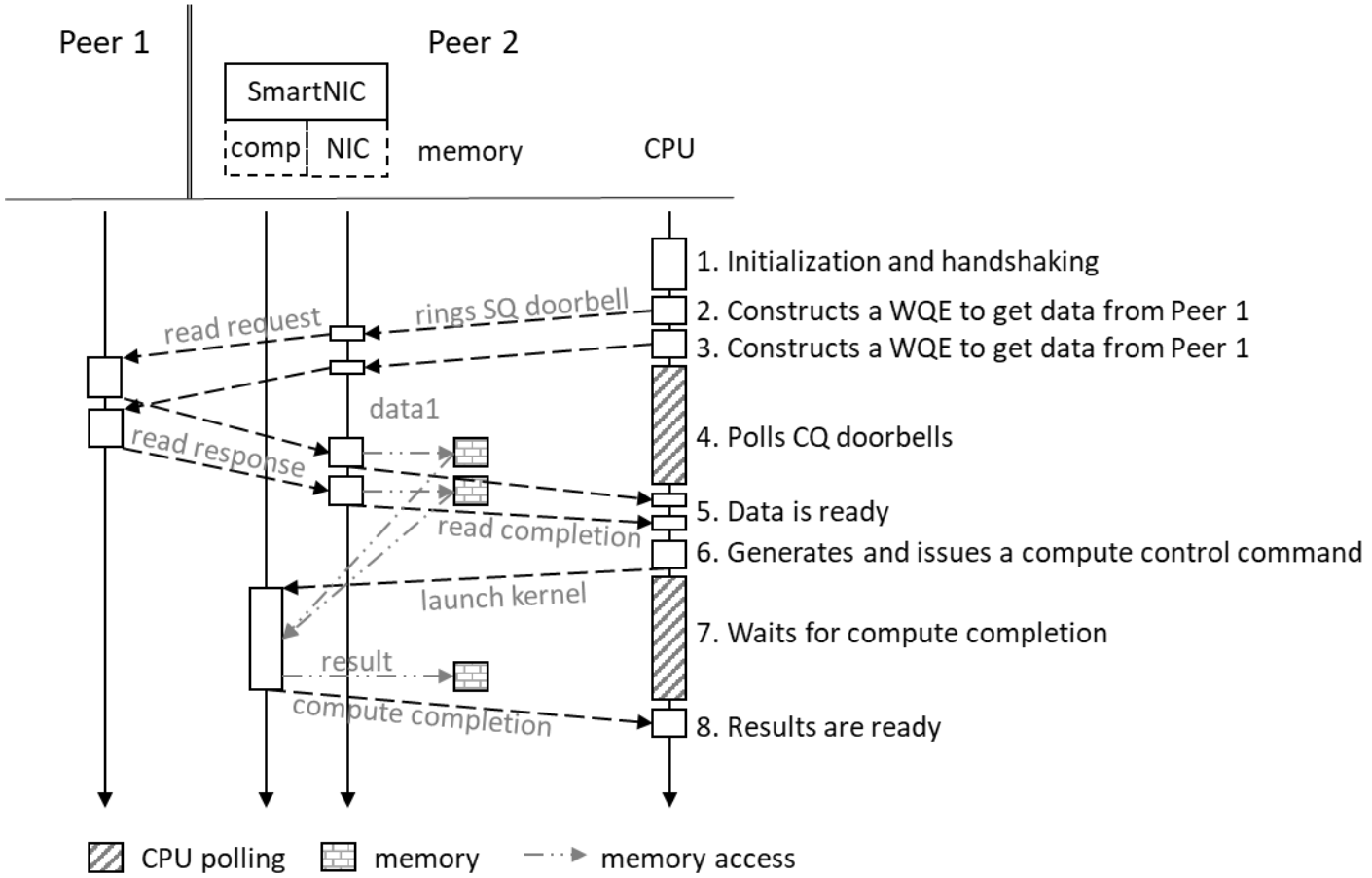
- Network stacks
- Memory driver
  - Data communication between host and FPGA memory
  - Host as a master
- Control driver
  - Register control
  - Compute control



Built-in lookaside example: network-attached systolic-array MM

- Two peers connected via 100Gbps network
  - data at Peer 1
  - Compute at Peer 2
- Compute control command

```
48  typedef struct {
49    uint32_t ctl_cmd_size;
50    uint32_t a_baseaddr;
51    uint32_t b_baseaddr;
52    uint32_t c_baseaddr;
53    uint16_t a_row;
54    uint16_t a_col;
55    uint16_t b_col;
56    uint16_t work_id;
57  } ctl_cmd_t;
```



Workflow of the example

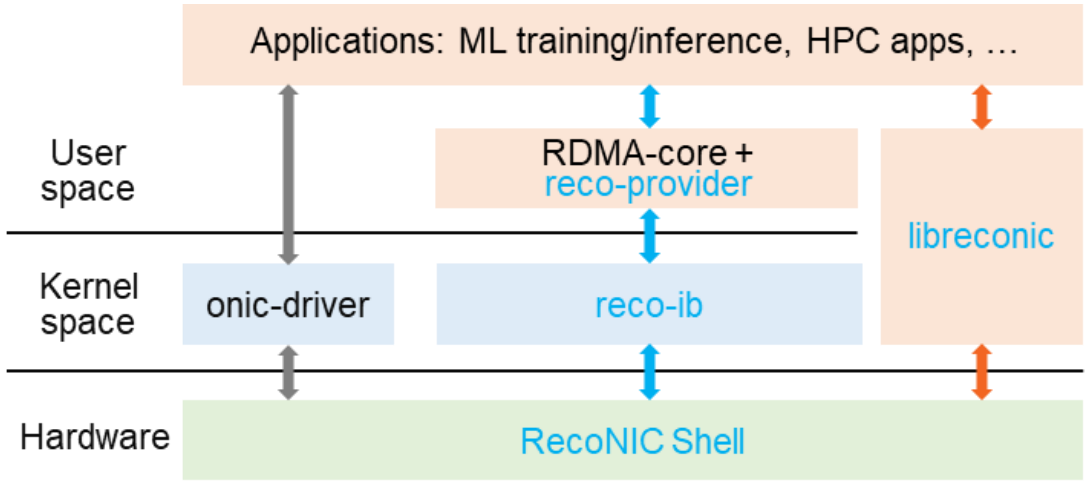
Built-in streaming example: packet classification

- To identify RDMA or non-RDMA traffic
- Designed with Vitis Networking P4
  - Parser
  - Forward
  - De-parser
- Input / output data in AXI4-Streaming

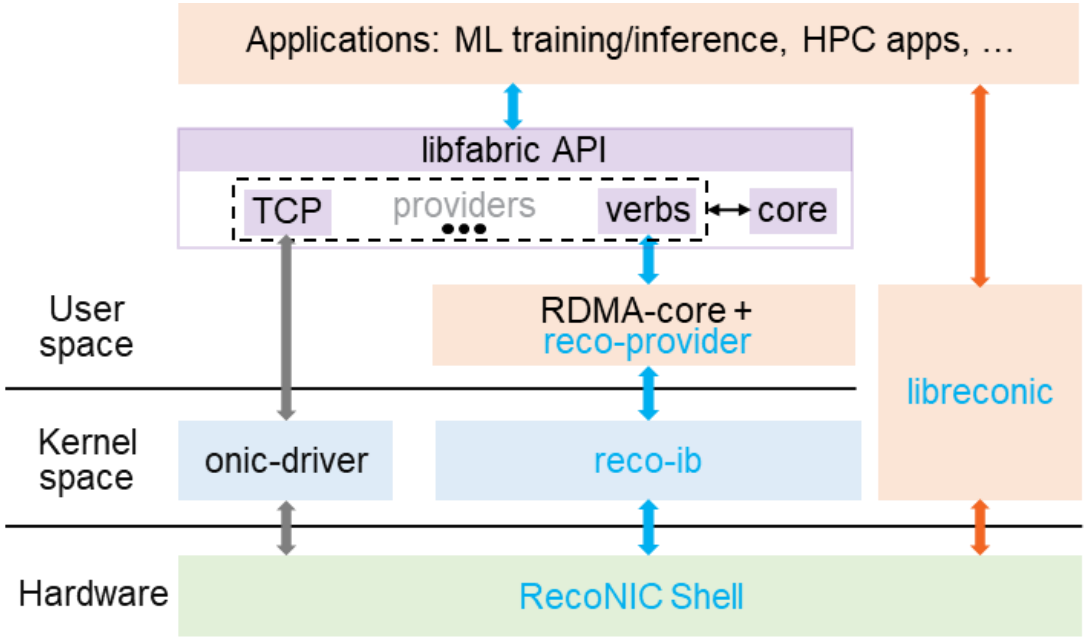
```
/* parser - Only accept UDP/IPv4 packets in the current implementation */
parser parser_inst(...) {
  state start { ... }
  state parse_ipv4 { ... }
  state parse_ipv4_options { ... }
  state dispatch_on_protocol { ... }
  state parse_udp { ... }
  state parse_reth { ... }
  state parse_aeth { ... }
  state parse_ieth { ... }
  state parse_immdt { ... }
}

/* RDMA atomic operations are not supported */
control forward_inst(...) {
  ...
  apply {
    ...
    if (hdr.udp.isValid()) {
      ip_src = hdr.ipv4.src;
      ip_dst = hdr.ipv4.dst;
      udp_sport = hdr.udp.sport;
      udp_dport = hdr.udp.dport;
      if(hdr.reth.isValid()) {
        ...
        is_rdma    = (bit<1>) 1;
      }
      if(hdr.aeth.isValid()) {
        ...
        is_rdma = (bit<1>) 1;
      }
      if(hdr.ieth.isValid()) {
        ...
        is_rdma = (bit<1>) 1;
      }
      if(hdr.bth.isValid()) {
        ...
        if(hdr.bth.connType != ((bit<3>) 0)) {
          is_rdma = (bit<1>) 0;
        } else {
          // Only consider reliable connection RDMA operations
          is_rdma = (bit<1>) 1;
        }
      }
    }
    pc_meta.*     = *;
    ...
  }
}

control deparser_inst(...) {
  apply {
    pkt.emit(...);
    ...
  }
}

XilinxPipeline(
  parser_inst(),
  forward_inst(),
  deparser_inst()
) main;
```

Libfabric over RecoNIC: possible integration

Current software/hardware system



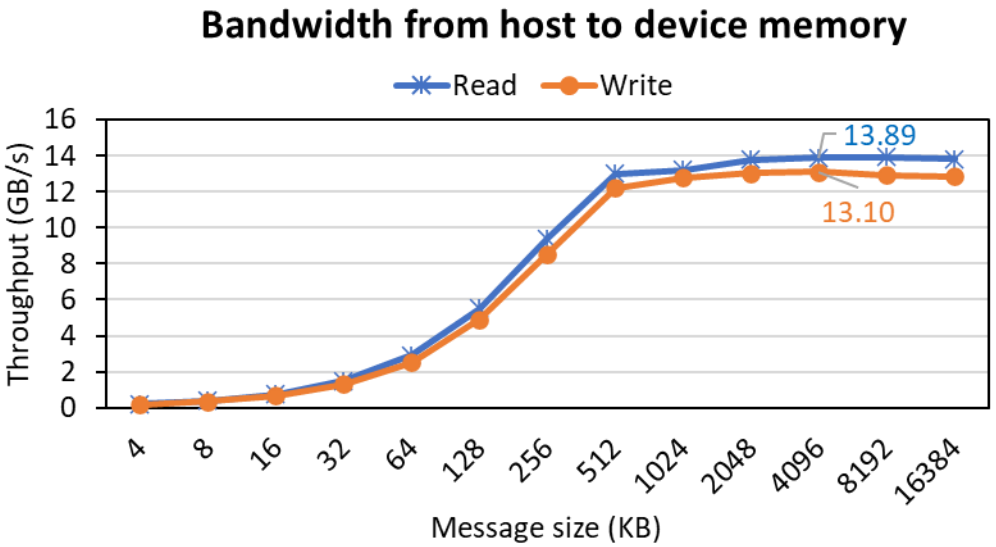
Interfacing with libfabric

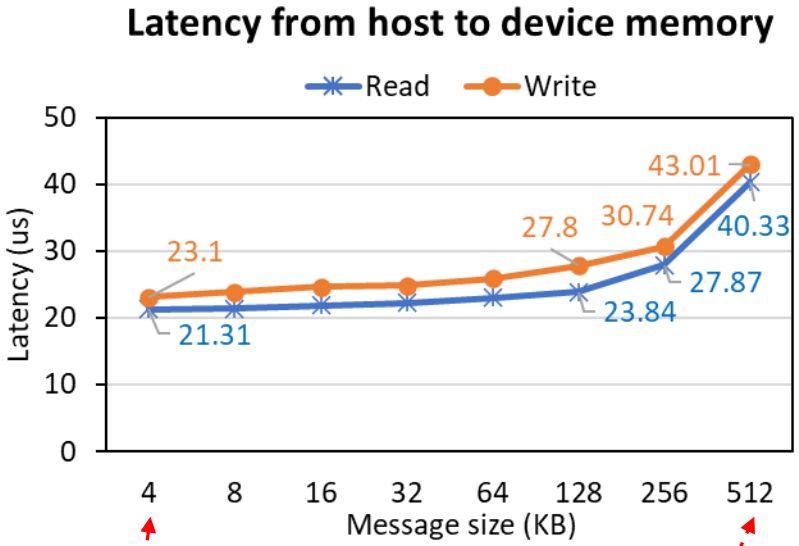


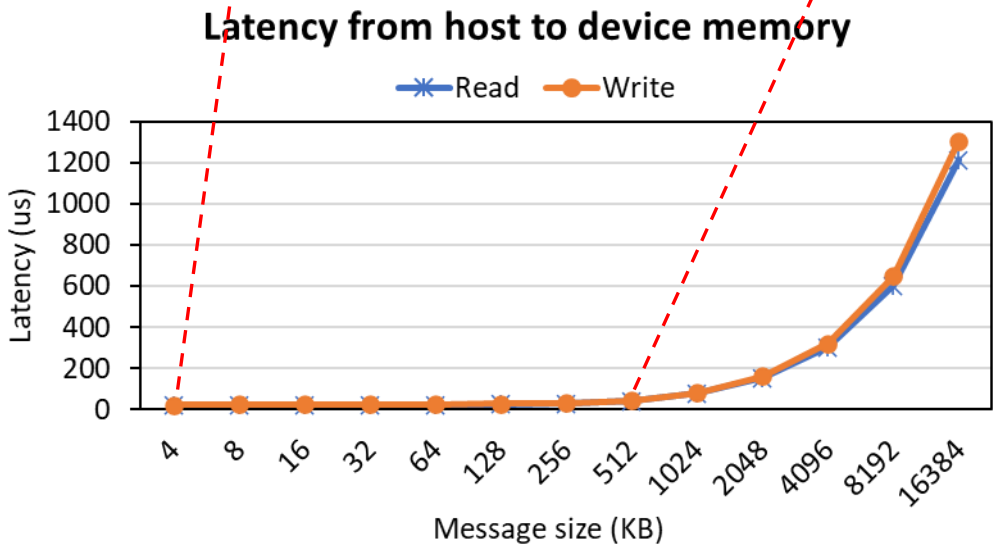
- RecoNIC supports RDMA-core
- RDMA-core provides ***libibverbs***, which can be leveraged by the ***verbs*** provider in libfabric

Data movement performance – Host as a master

- Host as a master to access device memory via QDMA AXI-MM channel
- ~13GB/s for transmitting data >= 512KB
- ~22us for small messages
  - Control overhead

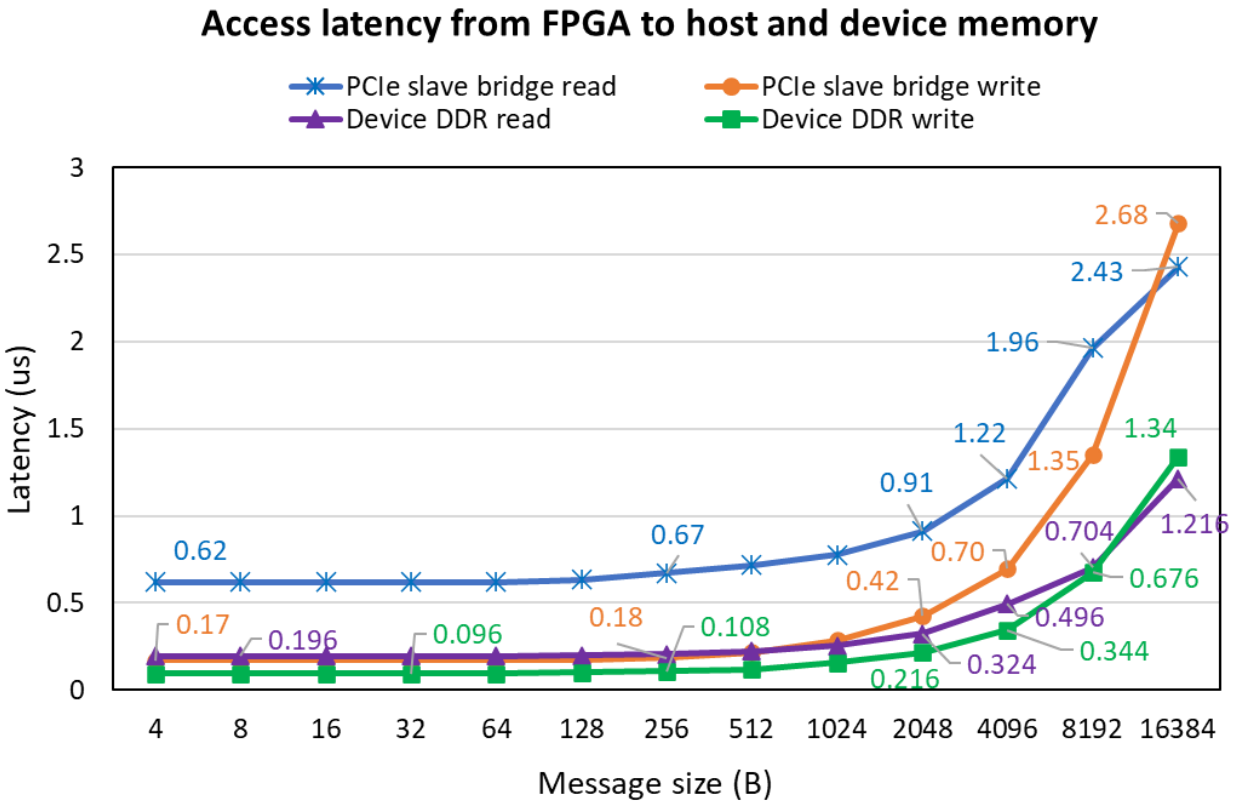






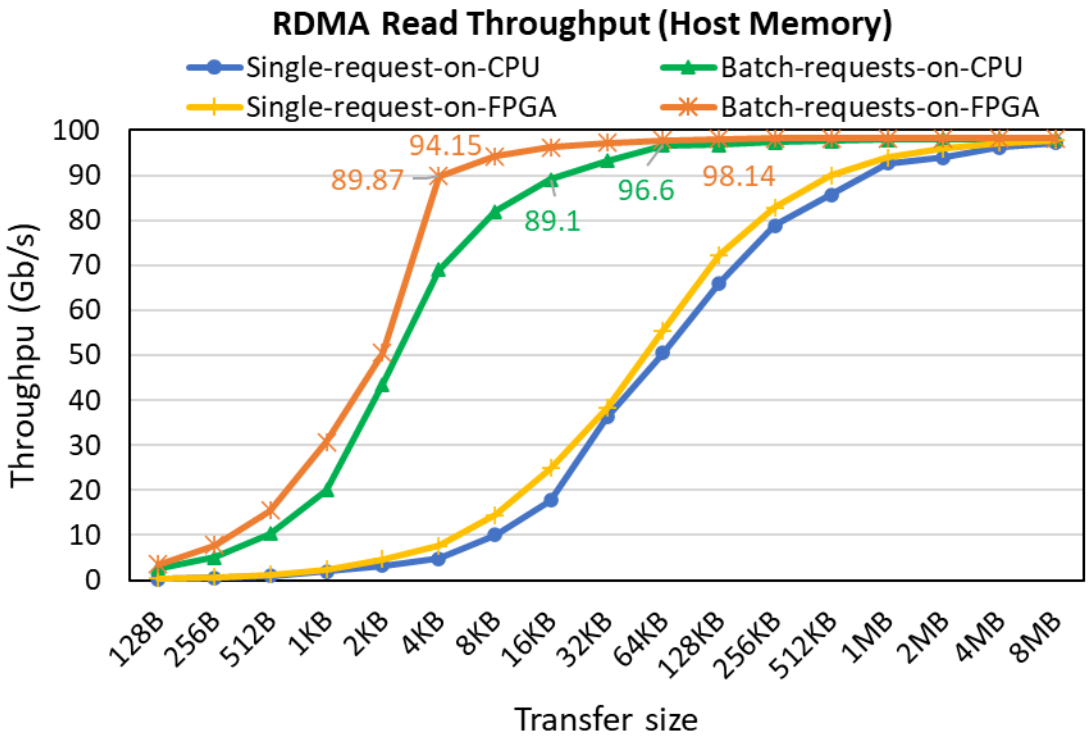
Data movement performance – FPGA as a master

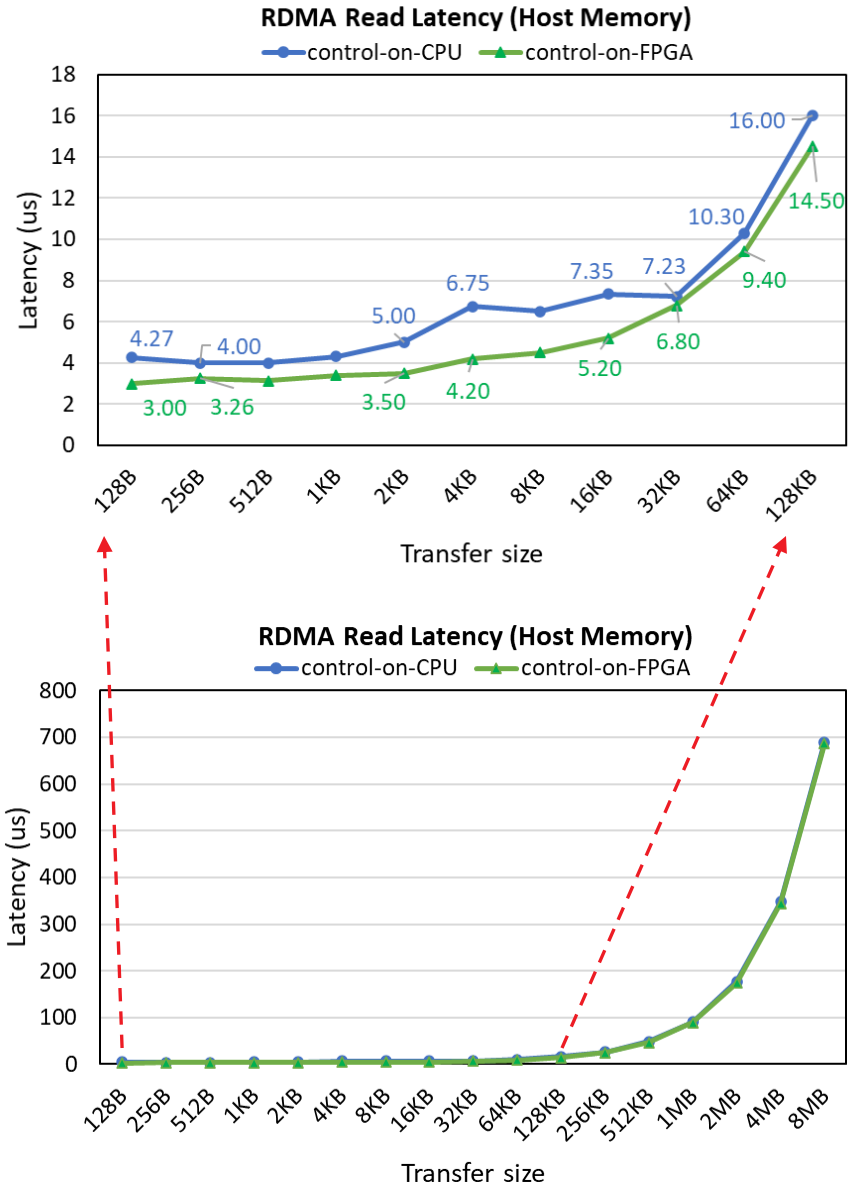
- FPGA as a master to access host memory via PCIe slave bridge
  - Low latency (e.g., 64B)
    - Write (in orange): ~0.17us
    - Read (in blue) : ~0.62us
- Memory access latency over PCIe slave bridge is much faster than that via QDMA AXI-MM channel
- FPGA to access device DDR
  - Low latency (e.g., 64B)
    - Write (in green) : ~0.096us
    - Read (in purple): ~0.196us
- Access latency to device memory is lower than host memory



RDMA read performance

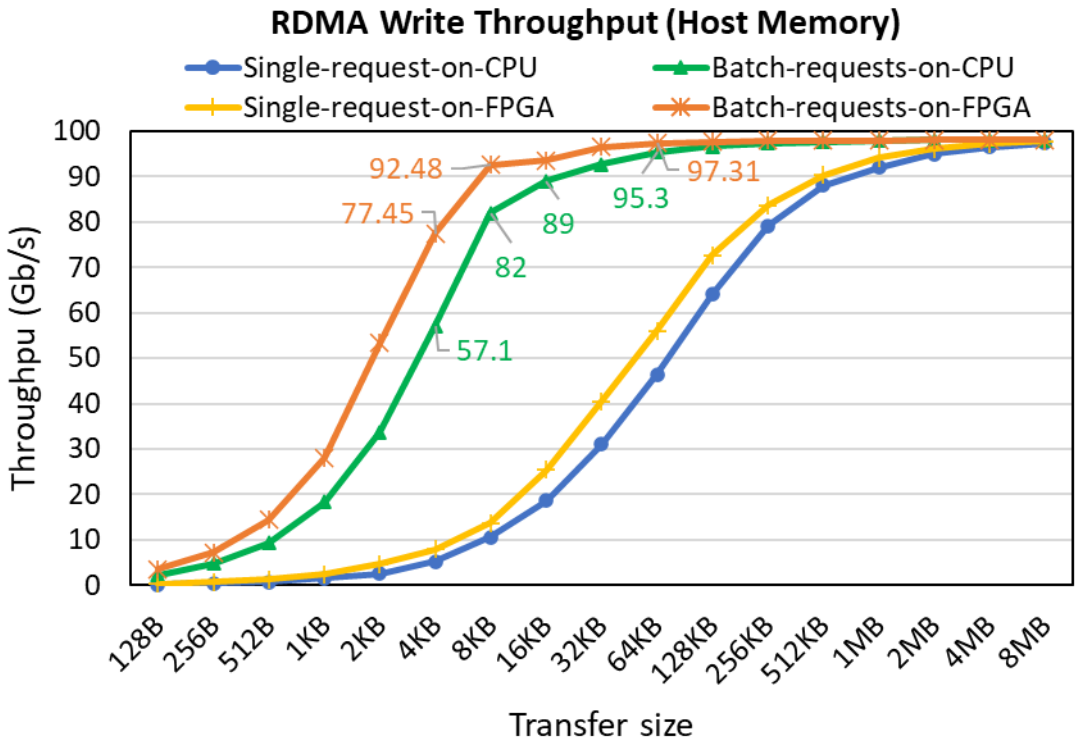
- QP defined in host memory
- Control offloading on FPGA can reduce 22% read latency for small message size (<= 128KB)
- Near line-rate throughput for 4KB message

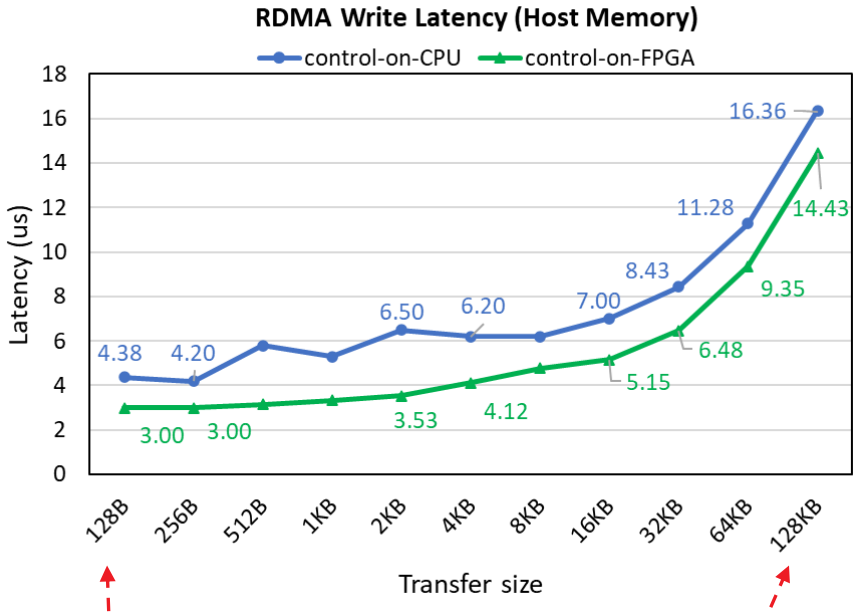


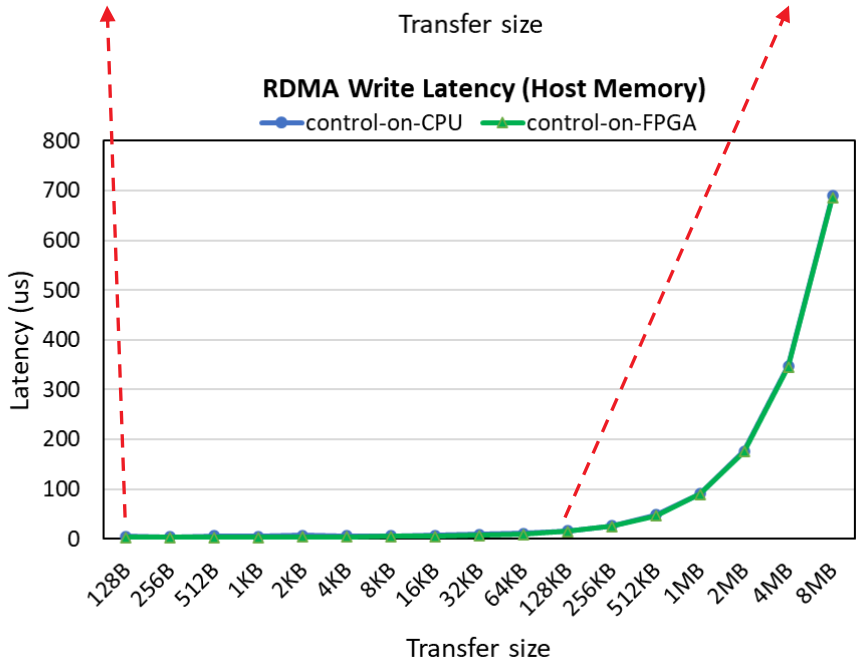


RDMA write performance

- QP defined in host memory
- Control offloading on FPGA can reduce ~29% write latency for small message size (<= 128KB)
- Near line-rate throughput for 8KB message

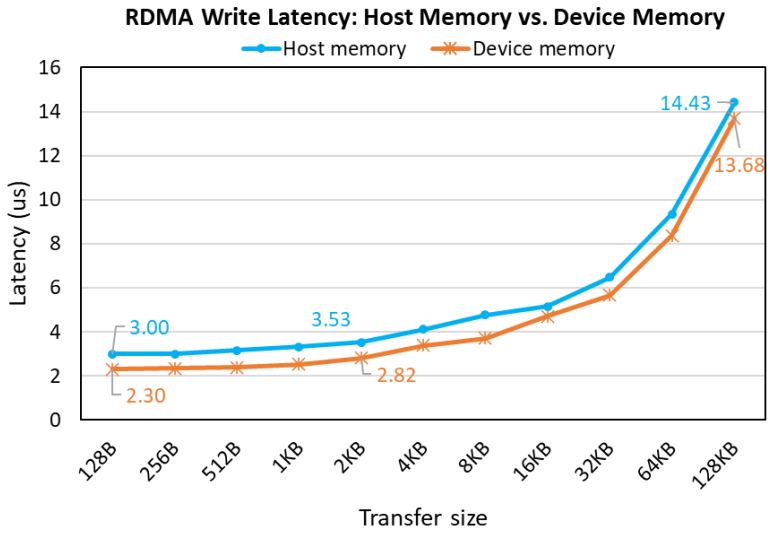


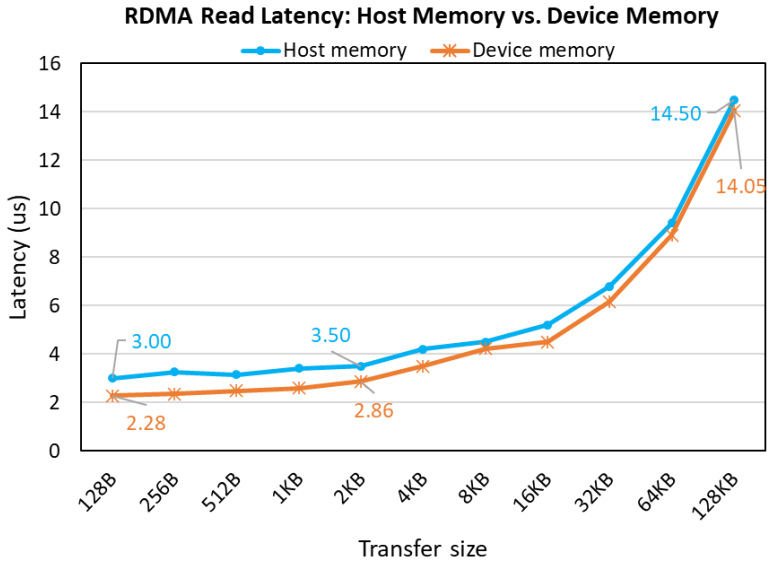




RDMA latency: host memory vs. device memory

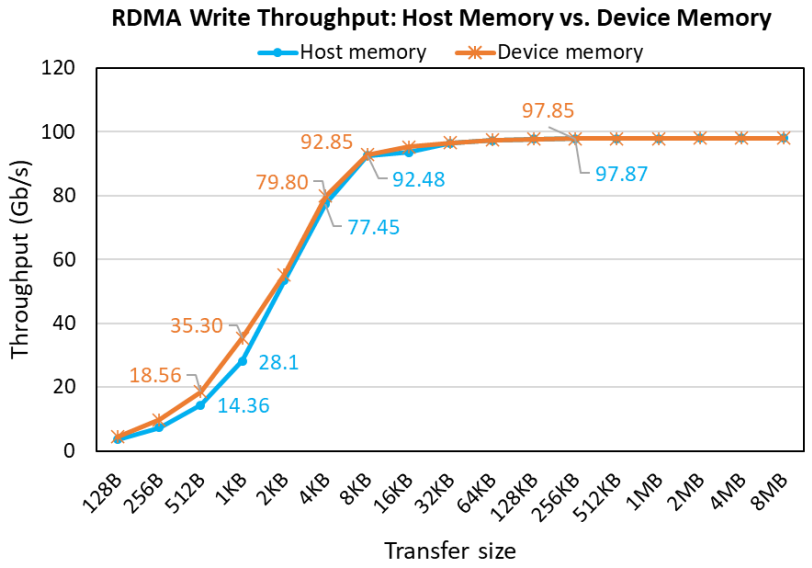
- QP declared in host memory and device memory
- Control offloading on FPGA
- RDMA write latency with QP in device memory is ~17.32% better than that in host memory
- RDMA read latency with QP in device memory is ~15.44% better than that in host memory
- DDR access latency is lower than PCIe access latency

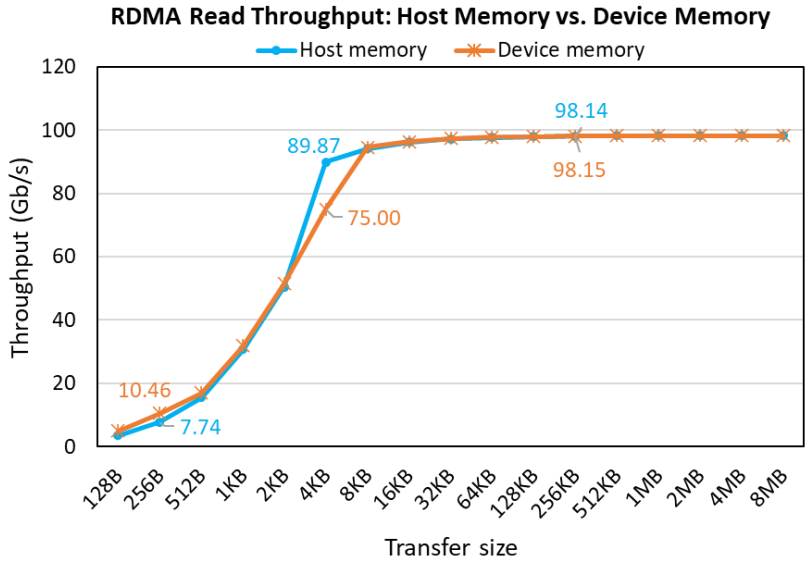




RDMA throughput: host memory vs. device memory

- QP declared in host memory and device memory
- Control offloading on FPGA
- RDMA write throughput with QP in device memory is slightly better than that in host memory
- RDMA read throughput with QP in device memory is almost the same with that in host memory, except for 4KB payload size





Conclusion

- RecoNIC is an open-sourced SmartNIC infrastructure/testbed for scale-out computing
  - First SmartNIC platform that interfaces ERNIC with x86 CPUs
  - Provides 100Gb/s line-rate RDMA traffic with low latency
  - Supports streaming and lookaside acceleration via VitisNetP4, HLS or RTL to process network data

- RecoNIC is available at https://github.com/Xilinx/RecoNIC
- A Primer on RecoNIC is available at https://arxiv.org/abs/2312.06207

- If you are interested in RecoNIC, please reach out to Henry (henry.zhong AT amd.com)

References

[1] AMD, “*AMD OpenNIC Project*”, https://github.com/Xilinx/open-nic, Accessed: 2024-04-09.

[2] A. Forencich, et al., ”*Corundum: An Open-Source 100-Gbps Nic*,” 2020 IEEE 28th Annual *International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, Fayetteville, AR, USA, 2020, pp. 38-46, doi: 10.1109/FCCM48280.2020.00015.

[3] A. M. Caulfield, et al. "*A cloud-scale acceleration architecture*." *2016 49th Annual IEEE/ACM international symposium on microarchitecture (MICRO)*. IEEE, 2016.

[4] G. Sutter, et al., "*FPGA-based TCP/IP Checksum Offloading Engine for 100 Gbps Networks*," *2018 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, Cancun, Mexico, 2018, pp. 1-6, doi: 10.1109/RECONFIG.2018.8641729.

[5] D. Sidler, et al. "*StRoM: smart remote memory.*" *Proceedings of the Fifteenth European Conference on Computer Systems*. 2020.

[6] AMD, “*AMD ERNIC*”, https://www.xilinx.com/products/intellectual-property/ef-di-ernic.html, Accessed: 2024-04-09.

COPYRIGHT AND DISCLAIMER

AMD, the AMD Arrow logo, AMD EPYC, AMD Infinity Fabric, AMD Infinity Cache, AMD Instinct MI250X, AMD Instinct 300 Series and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate releases, for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

THIS INFORMATION IS PROVIDED 'AS IS." AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.


AMD