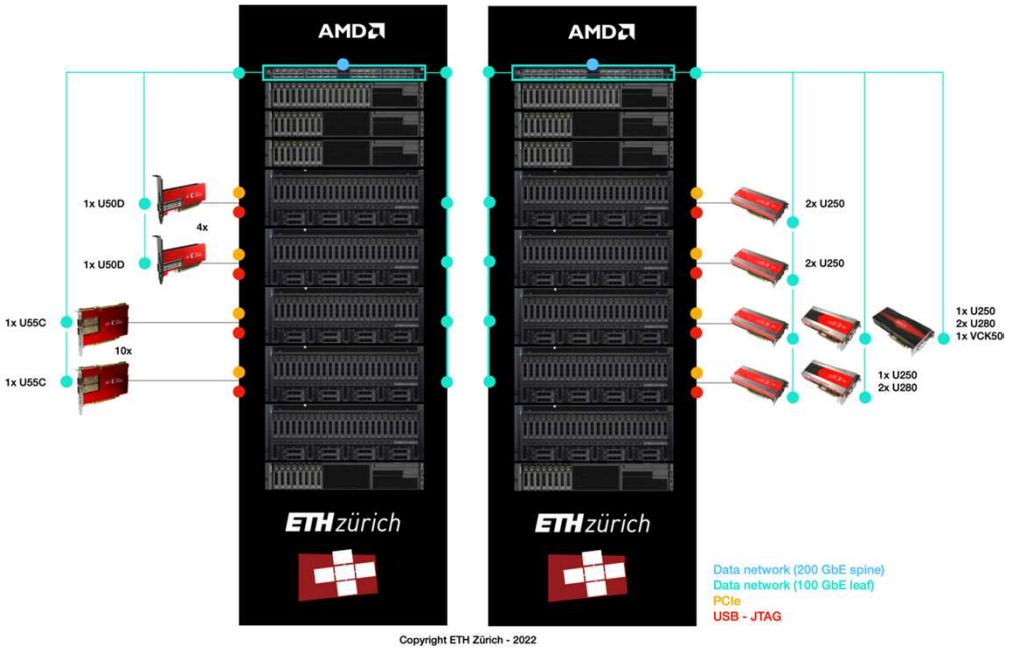
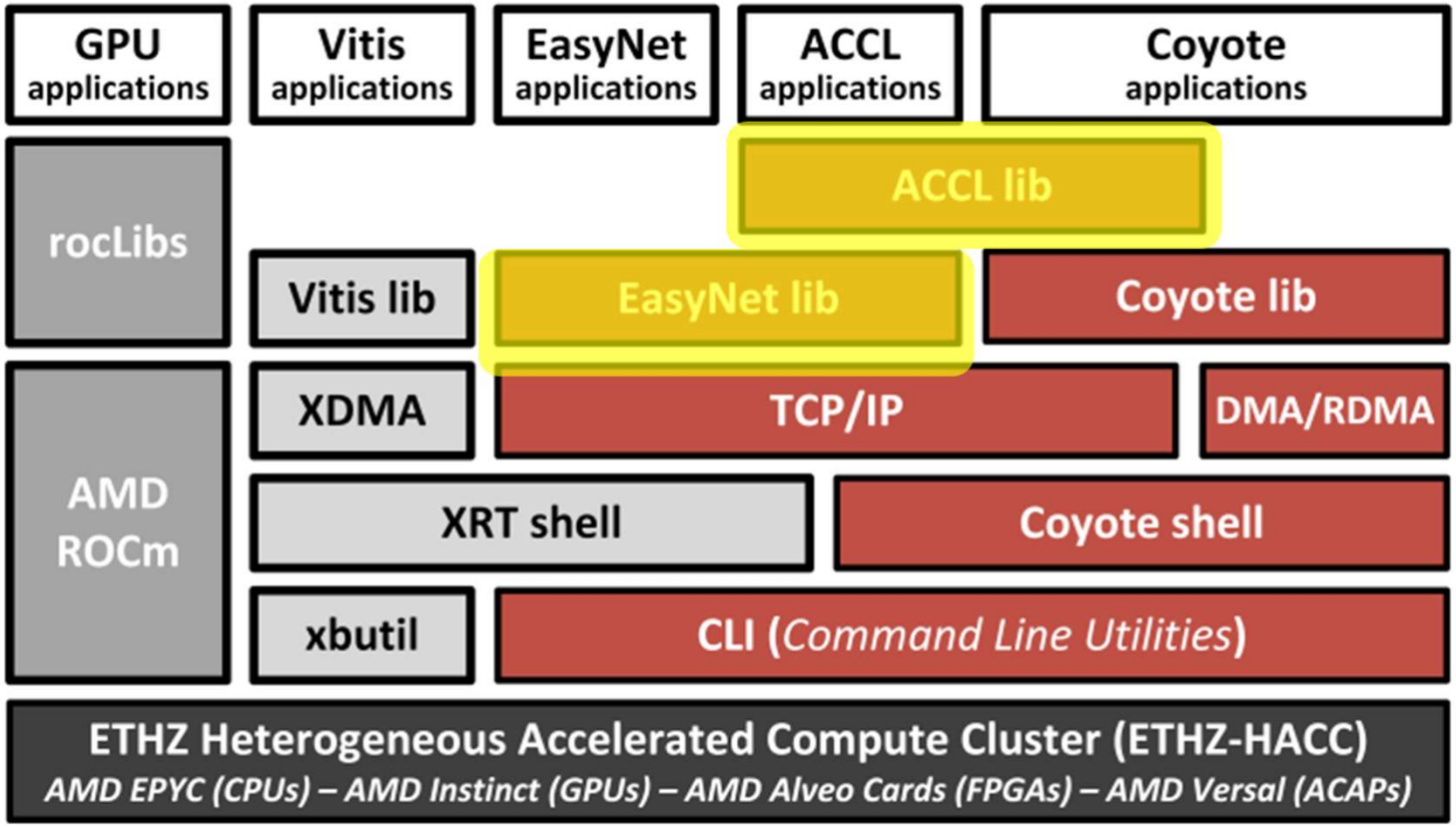
EasyNet and ACCL: Networking Support on FPGAs

Zhenhao He



ETHz HACC Cluster Infrastructure Overview

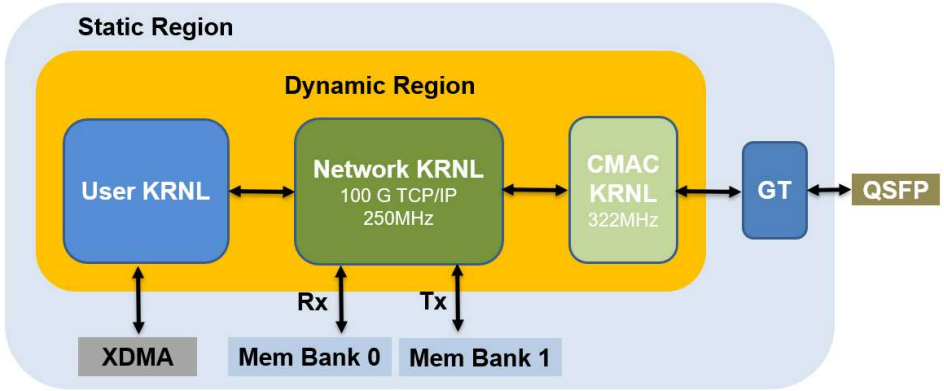


EasyNet: an Open 100 Gbps TCP Stack

TCP Network Offload Roadmap

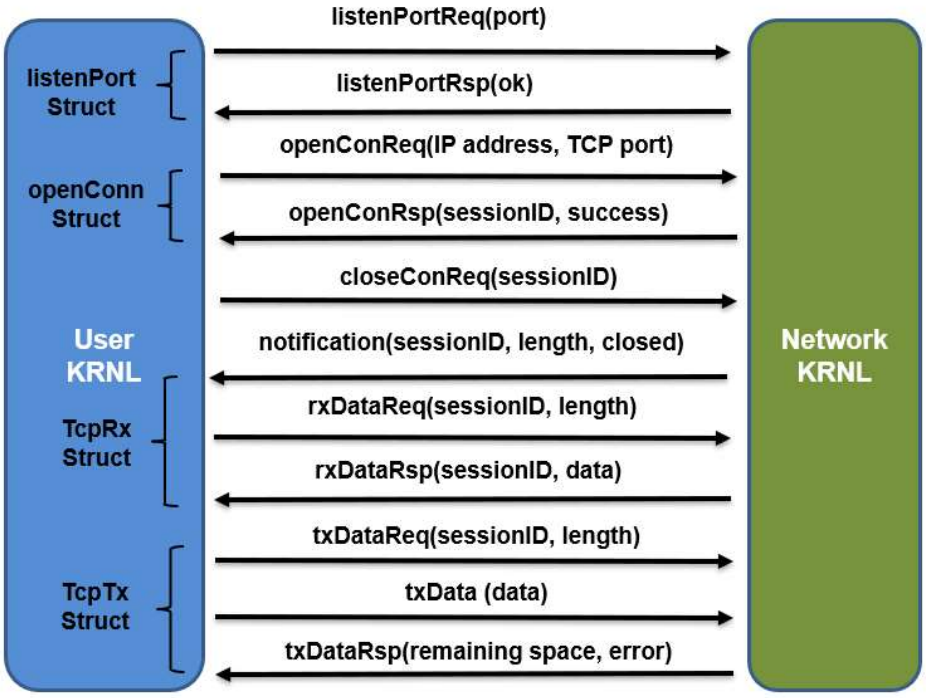
- **Years of efforts on TCP stack offload**
  - 10 Gbps TCP, FCCM'15, FPL'16
  - 100 Gbps TCP – Limago, FPL'19
  - ***100 Gbps TCP for HLS – EasyNet, FPL'21***
- **Improvement over years and feature complete**
  - Line-rate processing
  - Retransmission
  - Window scaling
  - Out-of-order
  - Thousands of connections
- **Tested and used in many projects**
  - Smart scatter-gather, SoCC'20
  - Distributed Recommendation, KDD'21, FPL'21

Overall Architecture



- CMAC Kernel
  - Ethernet subsystem, board specific
- Network Kernel
  - TCP/IP stack with streaming interfaces
- User Kernel
  - Customized unit for application
  - HLS and RTL kernel support

User – Network Kernel Streaming Interfaces



- Handshake protocol
  - Port and connection handshake
  - Tx handshake
  - Rx handshake

HLS Primitives to Abstract Low-level Handshakes

- HLS send

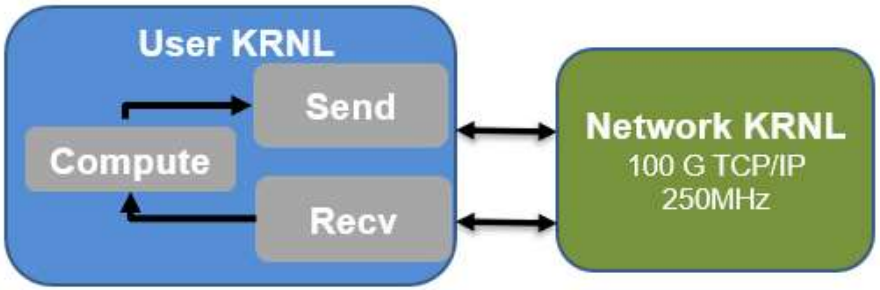
```
void send(dataType* send_data, uint64_t txByte,
    SessionStruct session, TcpTxStruct& TcpTxIntf);
```

- HLS receive

```
void recv(dataType* recv_data, uint64_t rxByte,
    SessionStruct session, TcpRxStruct& TcpRxIntf);
```

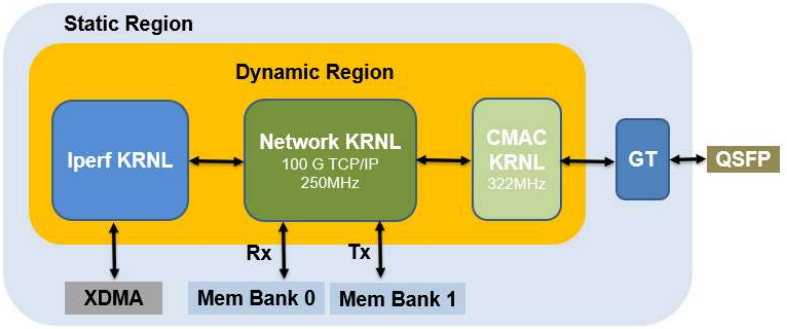
- Streaming & data flow

HLS Compile
-> Hardware



Example Kernels & Configurations

- https://github.com/fpgasystems/Vitis_with_100Gbps_TCP-IP
- User kernel supports both HLS and RTL kernels
  - HLS send, HLS recv
  - RTL iperf
- TCP configuration
  - TCP_STACK_MAX_SESSIONS
  - TCP_STACK_RX_DDR_BYPASS_EN
  - TCP_STACK_WINDOW_SCALING_EN
- Host bindings
  - OpenCL binding
  - XRT Native
- Supported boards:
  - U280, U250, U55C

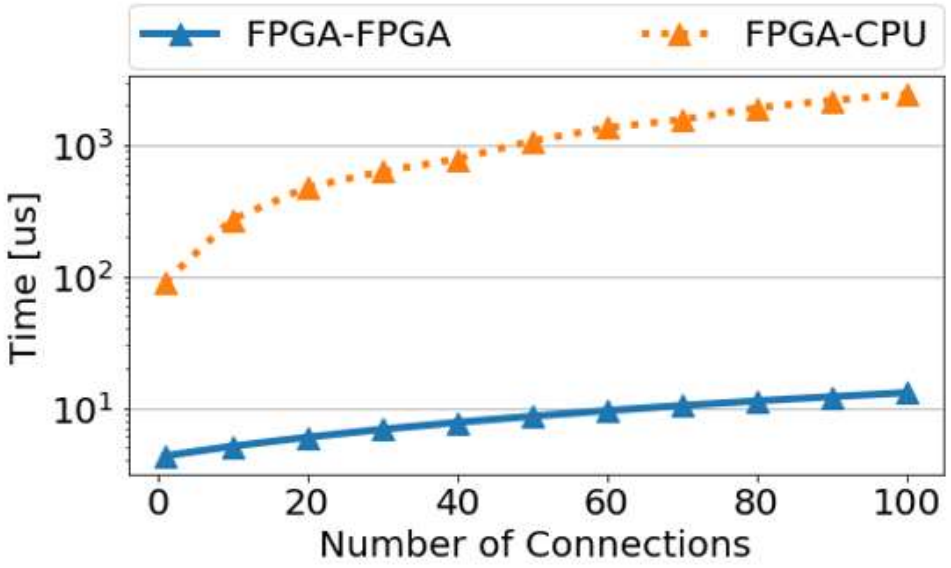



```
mkdir build
cd build
cmake .. -DFDEV_NAME=u280 -DTCP_STACK_EN=1 -DTCP_STACK_RX_DDR_BYPASS_EN=1
make installip
```

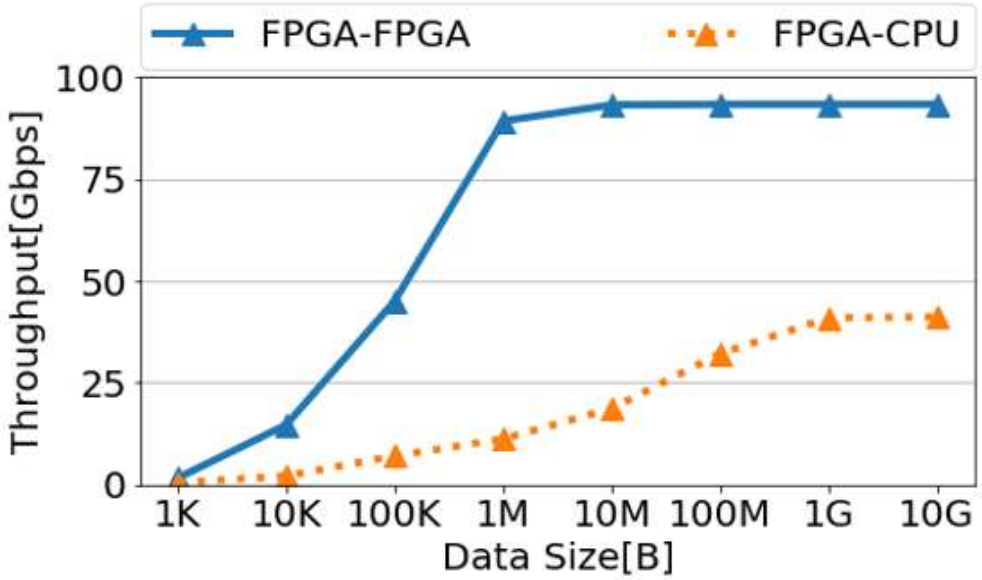
Performance

U280 FPGAs, interconnected via 100 Gbps Cisco Nexus network switch
Intel Xeon Gold 6234 processors with 100 Gbps Mellanox NIC
Performance measured in hardware

Latency RTT : FPGA-FPGA 5 us VS FPGA-CPU 46 us



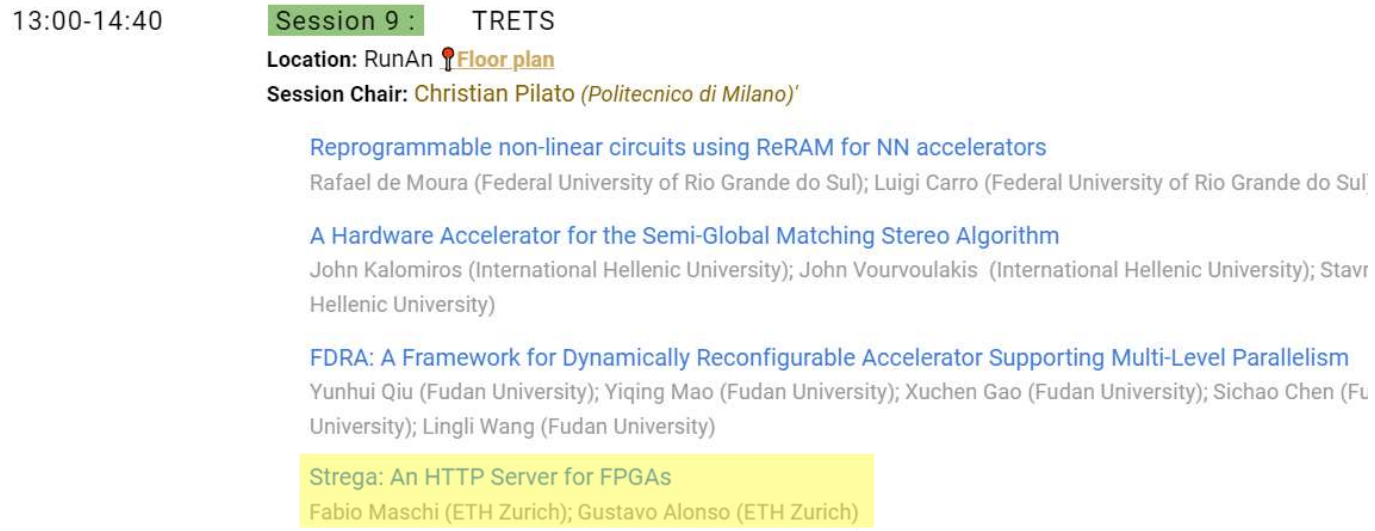
Open Connection Time



Send & Recv Primitive Throughput

Summary

- EasyNet provides reliable and high-performance transmission between FPGAs
- Easy to use with HLS bindings
- Suitable for applications with in-network-processing requirement
- Check out Strega, HTTP server built on top of EasyNet


13:00-14:40 Session 9 : TRETS
Location: RunAn Floor plan
Session Chair: Christian Pilato (Politecnico di Milano)'

Reprogrammable non-linear circuits using ReRAM for NN accelerators
Rafael de Moura (Federal University of Rio Grande do Sul); Luigi Carro (Federal University of Rio Grande do Sul

A Hardware Accelerator for the Semi-Global Matching Stereo Algorithm
John Kalomiros (International Hellenic University); John Vourvoulakis (International Hellenic University); Stavr
Hellenic University)

FDRA: A Framework for Dynamically Reconfigurable Accelerator Supporting Multi-Level Parallelism
Yunhui Qiu (Fudan University); Yiqing Mao (Fudan University); Xuchen Gao (Fudan University); Sichao Chen (Fu
University); Lingli Wang (Fudan University)

Strega: An HTTP Server for FPGAs
Fabio Maschi (ETH Zurich); Gustavo Alonso (ETH Zurich)

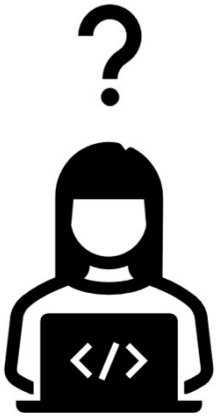
10

ACCL: FPGA-Accelerated Collective Communication Library

A Need for Higher-Level Abstraction

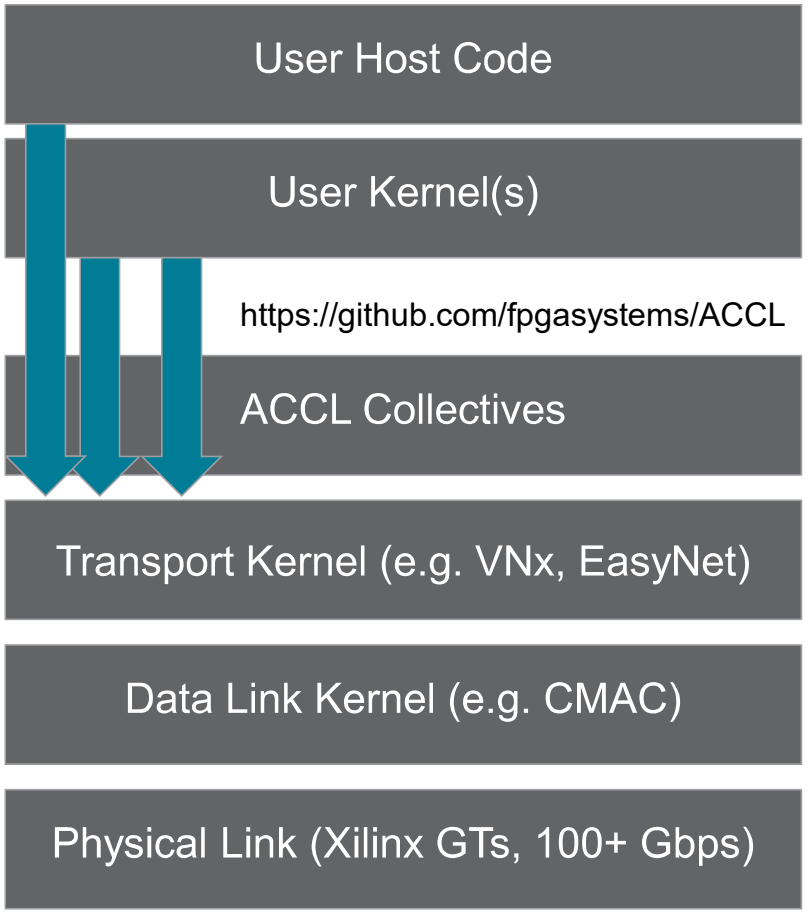
Partitioning Transport Kernel Between Different Clients

Orchestrating Complex Comms Patterns



Serving communications to both host and FPGA kernels

Simulate Networked Application

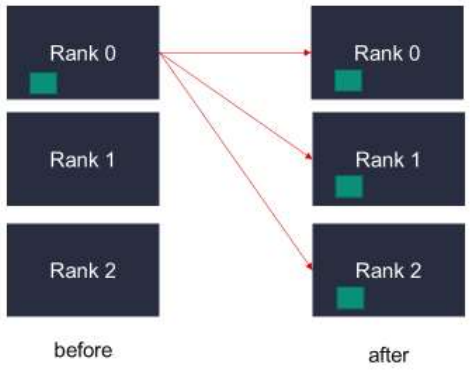


Collective Offload Kernel: ACCL

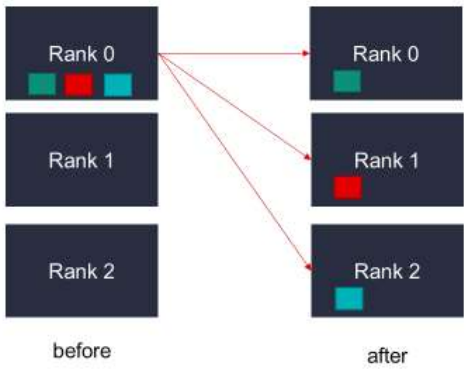
- Provides hardware-accelerated orchestration for a variety of complex communication patterns from the MPI standard
- Serves host and FPGA kernels
- As a result, reduces complexity of user design

What is a MPI collective? Some examples:

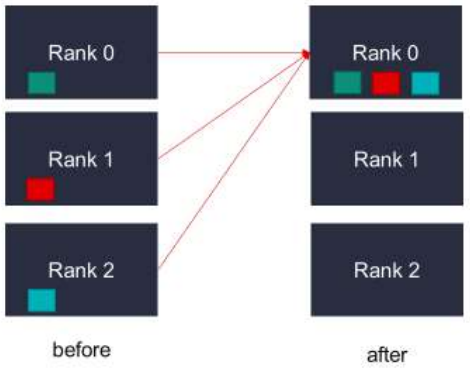
Broadcast



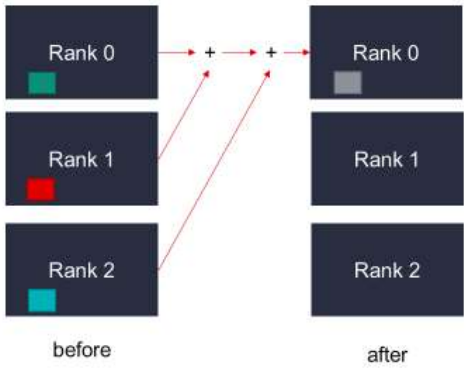
Scatter



(All)Gather



(All)Reduce



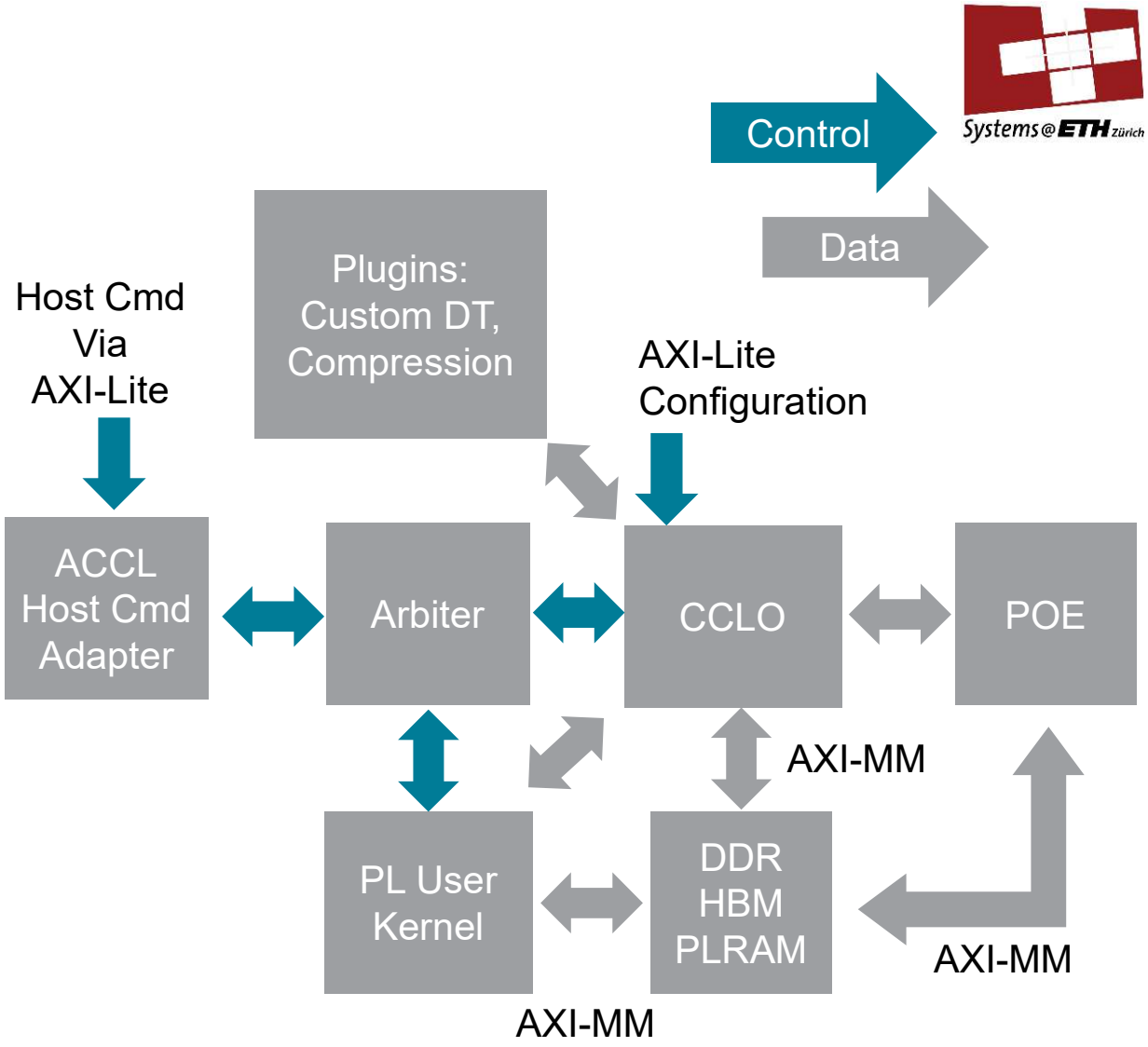
Applicability of MPI Primitives and Collectives

Application	Send/Recv	Scatter	Bcast	Gather	Reduce
Data-Parallel Training			✓		✓

- **8 out of ~400 MPI functions are enough to support a wide range of applications**
- **1 primitive, 4 collectives, 3 fused collectives (all-gather, all-reduce, scatter-reduce)**

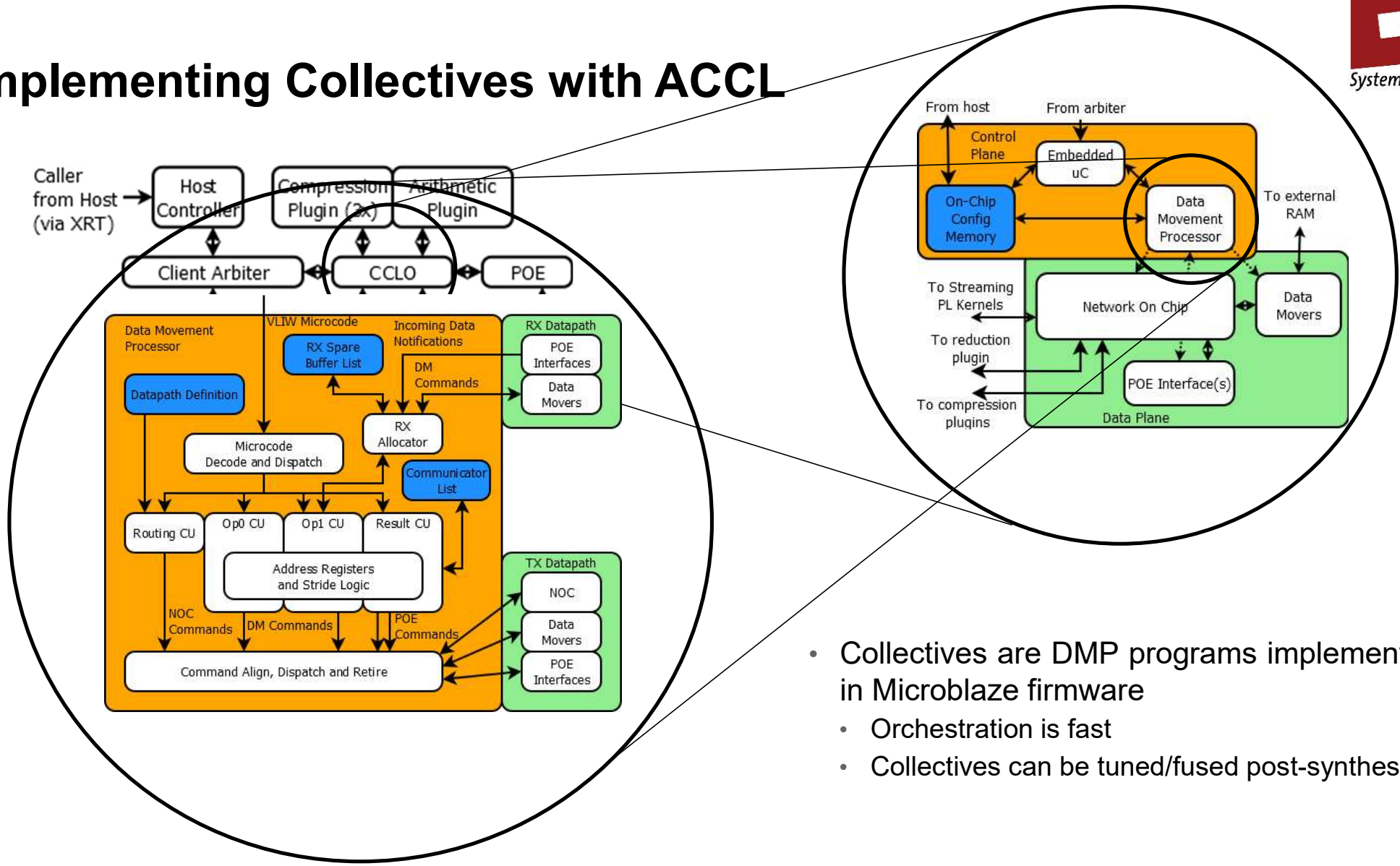
ACCL Design Goals

- Implement key MPI collectives with flexibility, portability and high performance
- Low latency communication control
  - CCL Offload (CCLO) Kernel
  - Automatic RX buffer management
- Protocol Offload Engine configurability
  - UDP, TCP
  - RDMA work in progress (Coyote integration)
- Host-less and host-full invocation
- Memory-less invocation/Streaming support
  - User kernels can talk to CCLO via AXI streams
- Expandability via Plug-Ins
- All kernels portable across boards



All connections via AXI-Streams unless otherwise specified

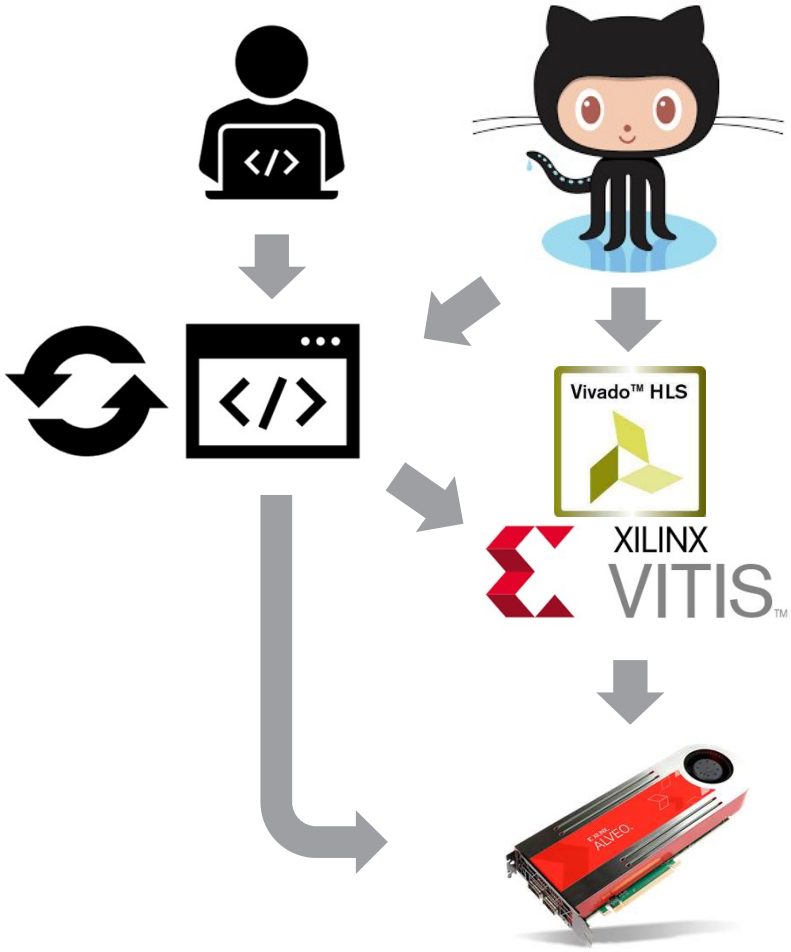
Implementing Collectives with ACCL



- Collectives are DMP programs implemented in Microblaze firmware
  - Orchestration is fast
  - Collectives can be tuned/fused post-synthesis

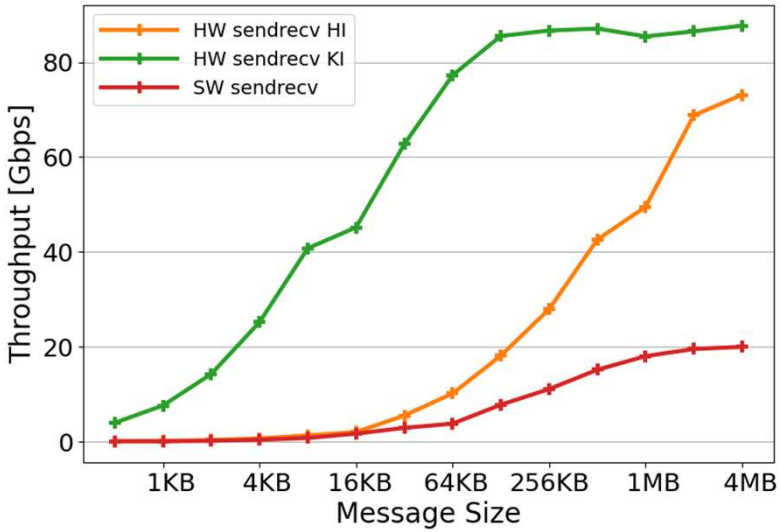
Steps to build ACCL-enabled FPGA application

- Clone ACCL repo(s):
  - https://github.com/fpgasystems/ACCL
- Build and verify your distributed application
  - With or without FPGA acceleration
  - Using ACCL HLS code emulator and RTL simulator
- Build appropriate CCLO kernel and plugins
- Link with Shell
  - Against platform, protocol offload engine (POE), and any application kernels
- Deploy to FPGA



ACCL Send & Recv throughput

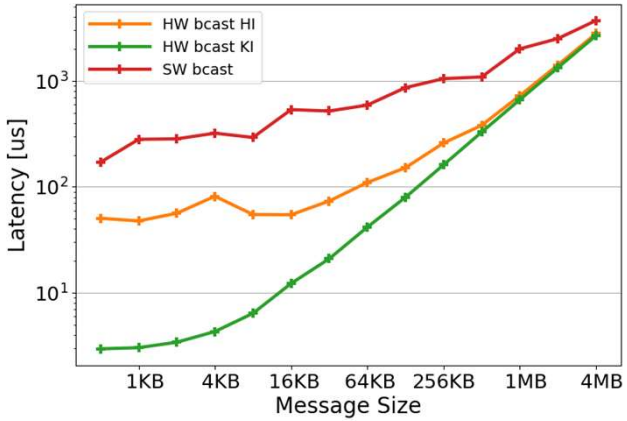
- Evaluation at ETHZ HACC:
  - ACCL dev + Alveo U55C/U280/U250 using Host Invocation (HI) and Kernel Invocation (KI)
  - MPICH 4.0.2 + Mellanox Connex X5 100G
- Performance on U280, U55C, and U250 is similar (design is **portable**)
- ACCL with TCP achieves **higher throughput** than MPICH over TCP



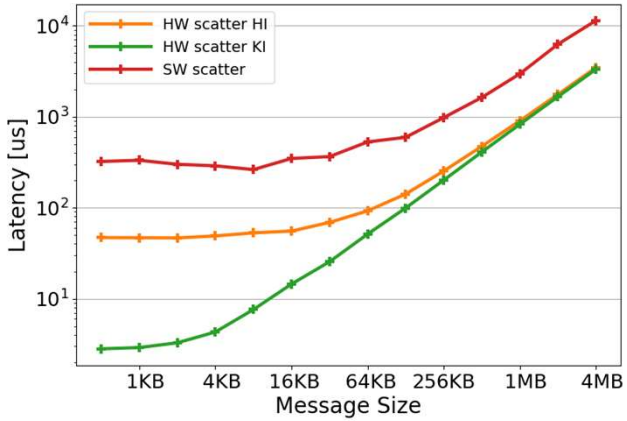
ACCL Collectives Performance

- Latency of Stream-to-Stream Collectives at ETHZ HACC, 8 ranks

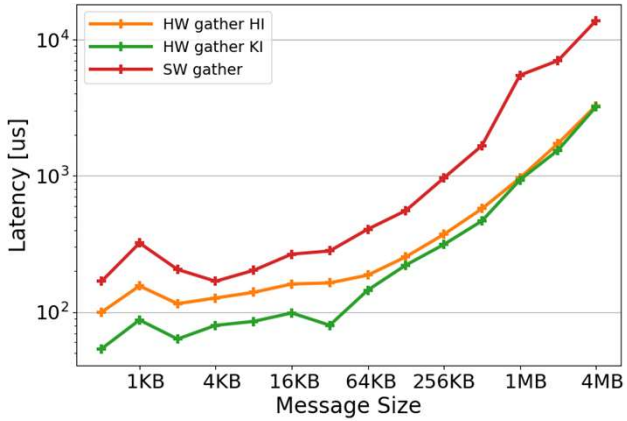
Broadcast



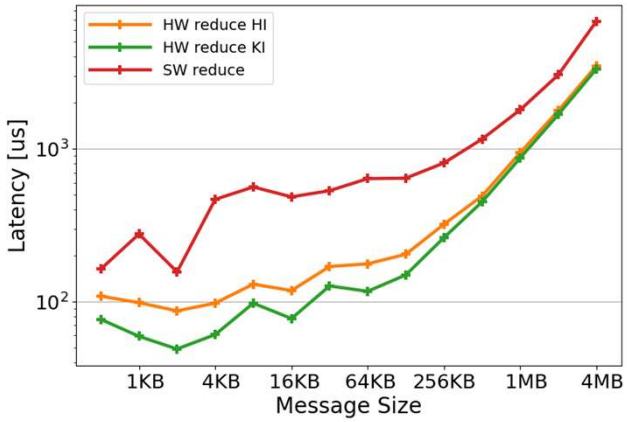
Scatter



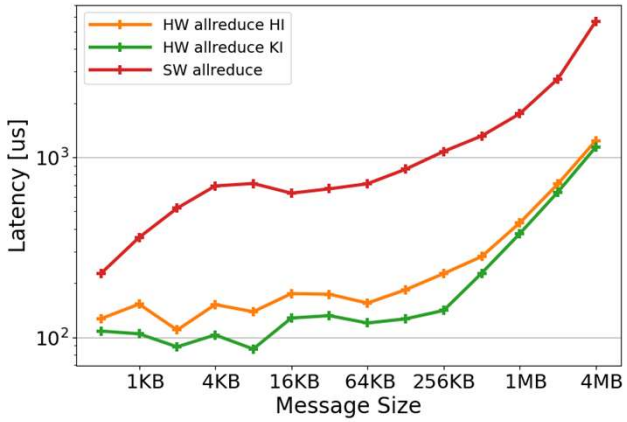
Gather



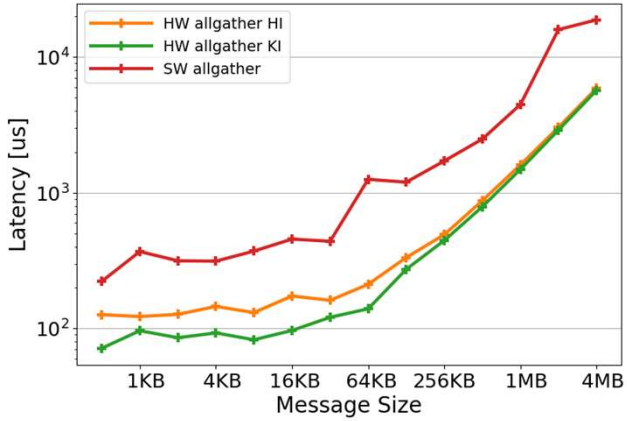
Reduce



All-Reduce

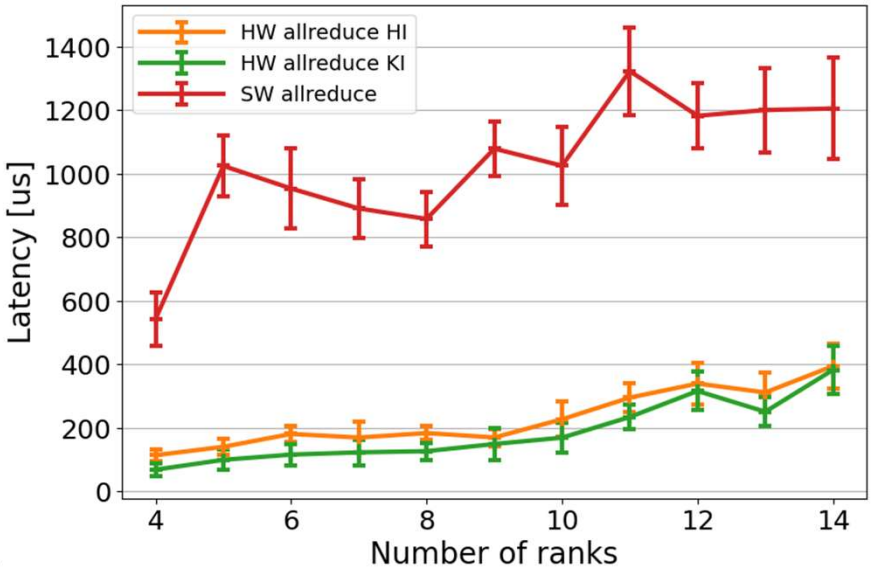


All-Gather



Scalability and Resource Consumption

- Scalability of All-reduce evaluated
  - Up to 14 ranks (10x U55C, 4x U280)
  - message size 128KB
  - 250 runs, average & range
- Compared to MPICH+TCP, ACCL+TCP has
  - Predictable latency vs. scale characteristic
  - Lower jitter



Component	kLUT	DSP	BRAM18	URAM
CCLO	81	27	75	0
TCP POE	111	0	813	1
UDP POE	23	0	115	0
CMAC	12	0	34	9

- Resource consumption
  - CCLO ~15% of LUTs on a U250
  - Choice of POE most significant for resource usage

ACCL: Coyote Integration and RDMA Extension

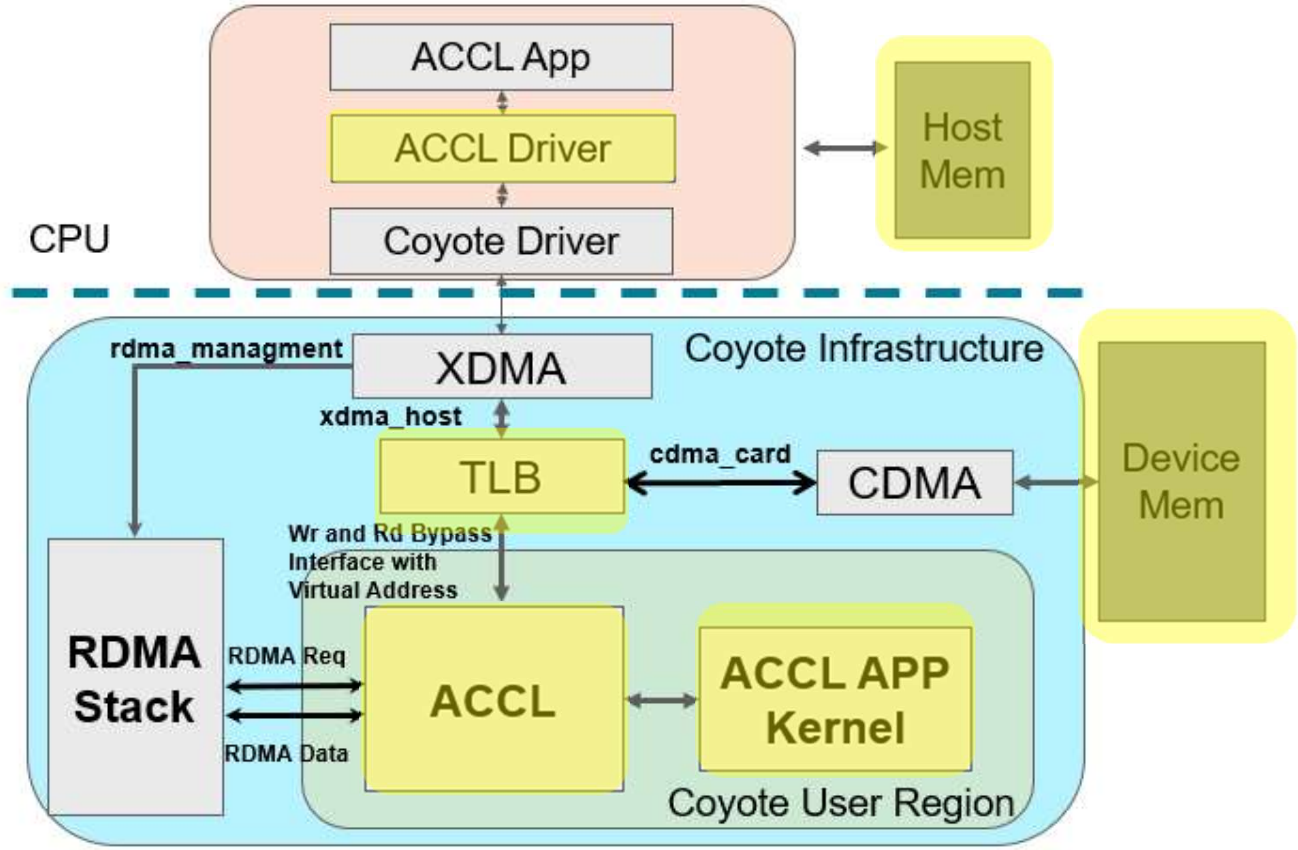
ACCL – Coyote Integration

- What can ACCL + XRT/Vitis provide us?
  - Low latency collective to move data resides on the FPGA
  - No need to move data through the CPU NIC, and so on….

- However,
  - XRT/Vitis host invocation latency is high: ~50 us
    - High overhead for small messages/short-lived program
  - Data movement between the host and device must be staged at FPGA memory
    - Data can not be streamed from host memory directly to FPGA network
  - Most Xilinx shell doesn't allow FPGA to access host memory bypassing CPU
    - No Viable solution for RDMA to host memory

- **ACCL integration with Coyote**
- Low invocation overhead from the host: ~2 us
- Unified memory space across device and host
- Enable ACCL with RDMA backend
- More use case can be explored

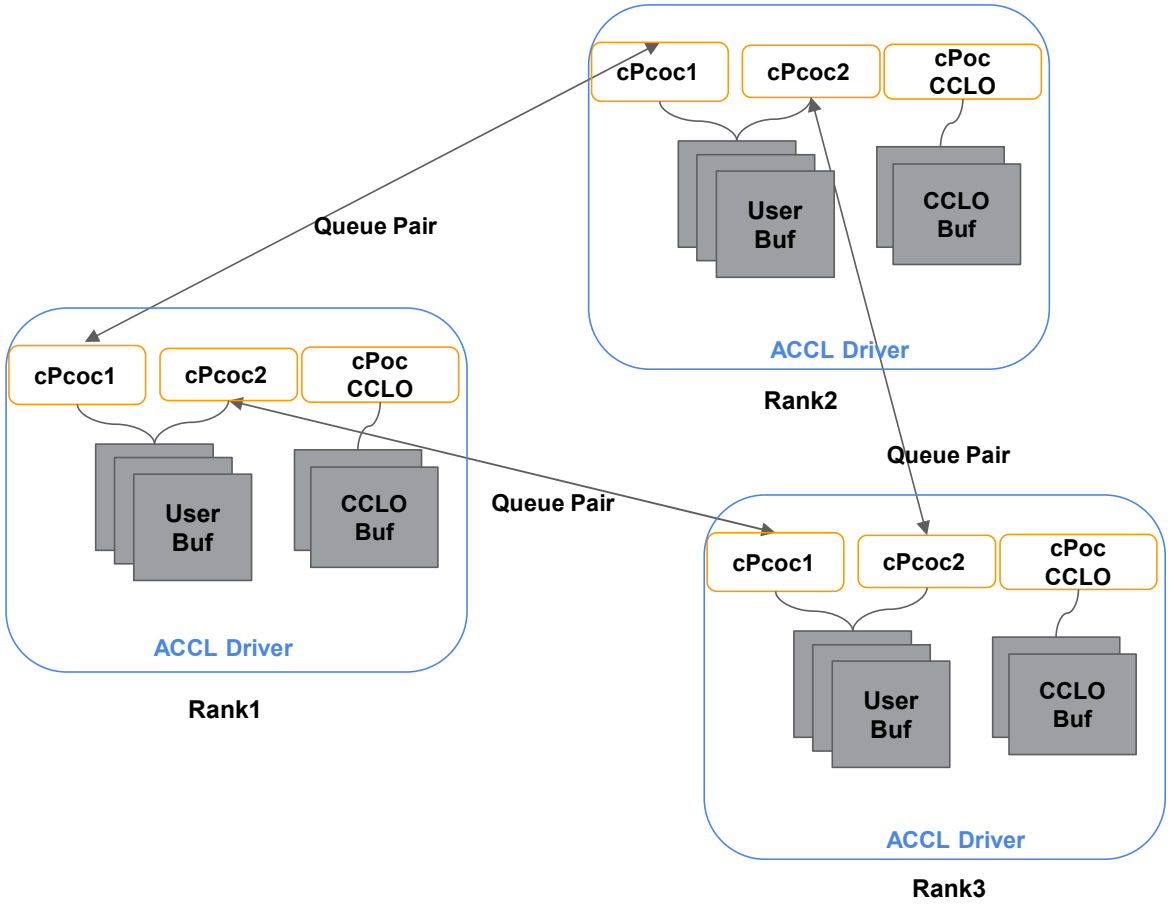
ACCL – RDMA with Coyote



- ACCL hardware sits on RDMA path to host/device memory
- ACCL supports both host buffer and device buffer
- ACCL manages virtual address and address translation is done by Coyote TLB
- ACCL allows host invocation and direct invocation from the FPGA kernels

ACCL – RDMA Buffer and Queue Pair Management

- ACCL needs to managed user buffers and Rx buffers
- Rx buffers are managed by one dedicated Coyote Process (cProc)
- One user buffer can be associated with multiple cProc corresponding to different ranks.
- Queue pair is constructed with a pair of Coyote Process identifiers
- All ACCL buffers are mapped into TLBs

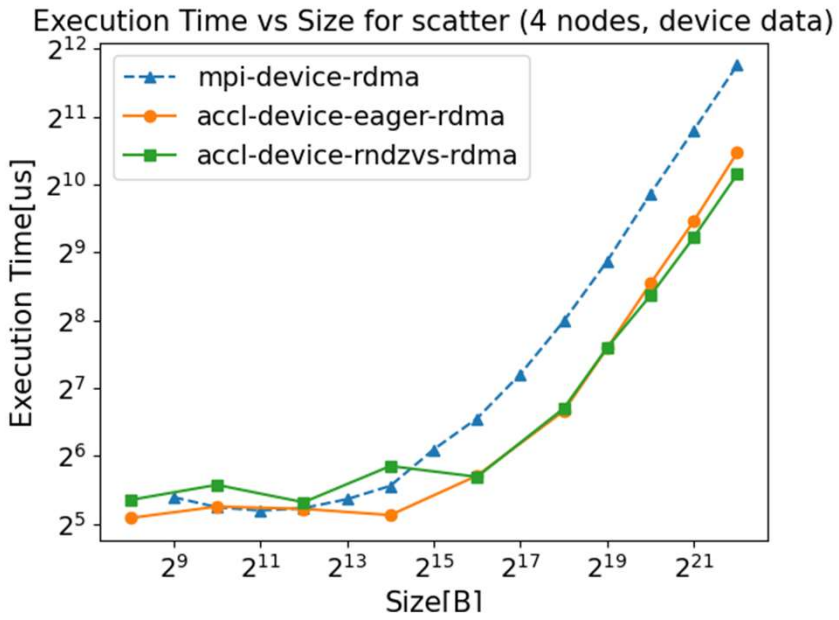


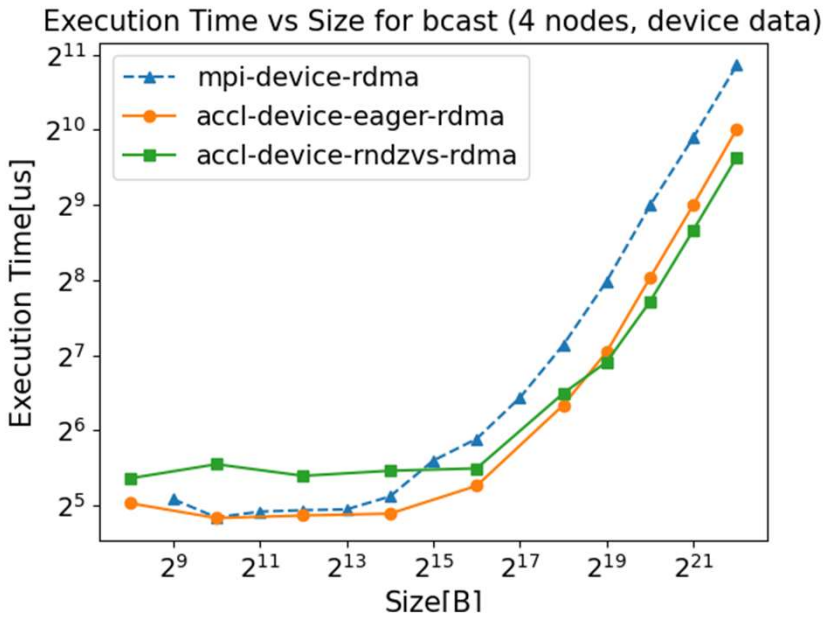
ACCL – RDMA: Eager and Rendezvous Protocol

- Eager Protocol:
  - Message sent assuming destination can store
  - Temporary buffering required
  - Applied protocol for ACCL – TCP implementation
- Rendezvous:
  - Message not sent until destination address is resolved
  - Makes more sense with RDMA to avoid redundant mem copy
- ACCL – RDMA supports both Eager and Rendezvous with same piece of hardware
- Eager Protocol
  - Implemented with RDMA SEND Verb
  - SEND Verb doesn’t contain destination address information
- Rendezvous Protocol
  - Address resolution with RDMA SEND Verb
  - Data transfer with RDMA WRITE Verb

25

Preview: ACCL – RDMA Performance with Device Data

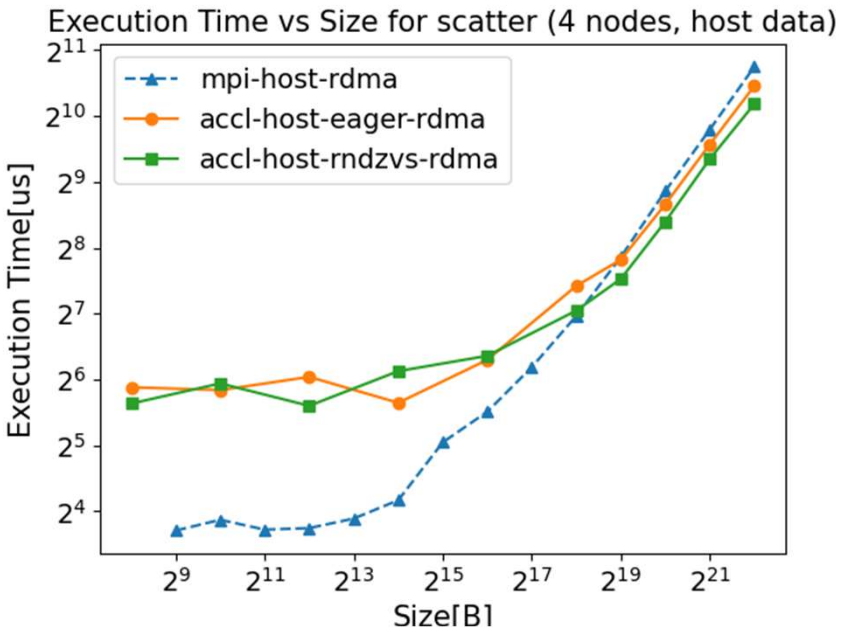


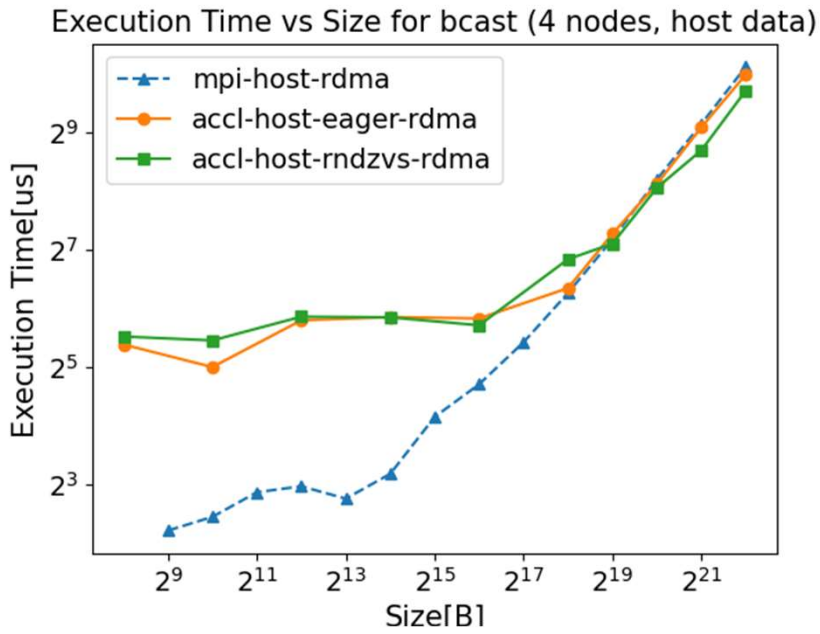


- ACCL-RDMA running on FPGAs compared with MPI-RDMA with Mellanox NIC running on CPUs
- Data resides on FPGA memory; MPI-RDMA numbers include data movement data from/to FPGA memory
- ACCL-RDMA outperforms MPI-RDMA in all cases
- ACCL Rendezvous outperforms ACCL Eager with large message sizes
- ACCL-RDMA is suitable for distributed applications running on top of FPGAs

26

Preview: ACCL – RDMA Performance with Host Data





- ACCL-RDMA running on FPGAs compared with MPI-RDMA with Mellanox NIC running on CPUs
- Data resides on CPU memory; ACCL-RDMA numbers include data movement data from/to FPGA memory
- ACCL-RDMA outperforms MPI-RDMA with medium/large message size; CPU and commodity NIC has high frequency
- ACCL-RDMA is suitable for distributed CPU applications that involves large/medium data transfer

27

Summary

- ACCL with device data shows superior performance compared to MPI-RDMA solutions
  - Great enabler for distributed applications on top of FPGAs
- ACCL also shows reasonable performance to move host data
  - Can be considered as smart-NIC for HPC applications running on CPUs
  - With moderate computation along the network path to amortize the cost of offloading
- ACCL has great potential for other scenarios, e.g., hybrid distributed CPU/FPGA applications
  - Example: machine learning pipeline, where CPUs pre-process incoming data and FPGAs process machine learning inference.

- *Disclaimer: results shown with ACCL – RDMA are early results, subject to be improved*
- *Full release of ACCL – RDMA coming soon, stay tuned: https://github.com/fpgasystems/ACCL*

28

Acknowledgement: AMD Xilinx

More information can be found:
https://systems.ethz.ch/research/data-processing-on-modern-hardware/hacc/tutorial-fpl-2023.html