AMD

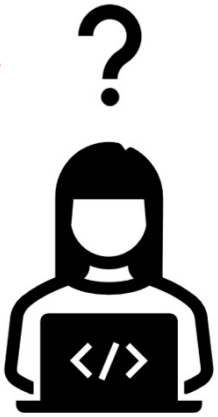
ACCL: A Vitis-compatible, FPGA-Accelerated Collective Communication Library

Lucian Petrica

A Need for Higher-Level Abstraction

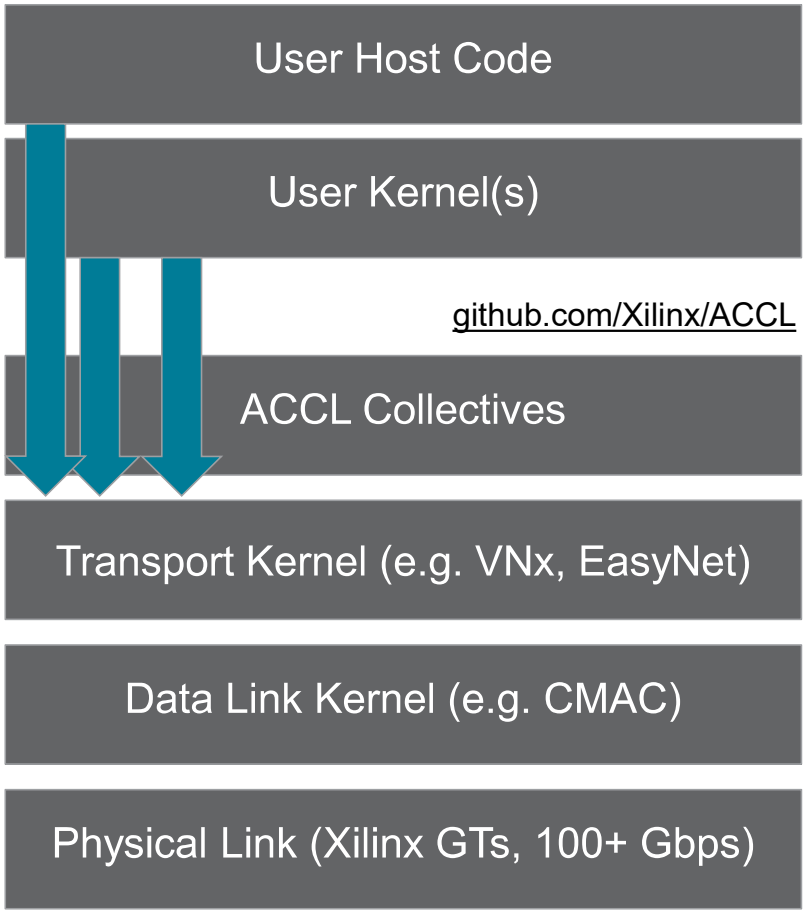
Partitioning
Transport Kernel
Between PL Clients

Orchestrating
Complex Comms
Patterns



Serving comms to both host (via memory) and PL (via memory or streams)

Simulate
Networked
Application



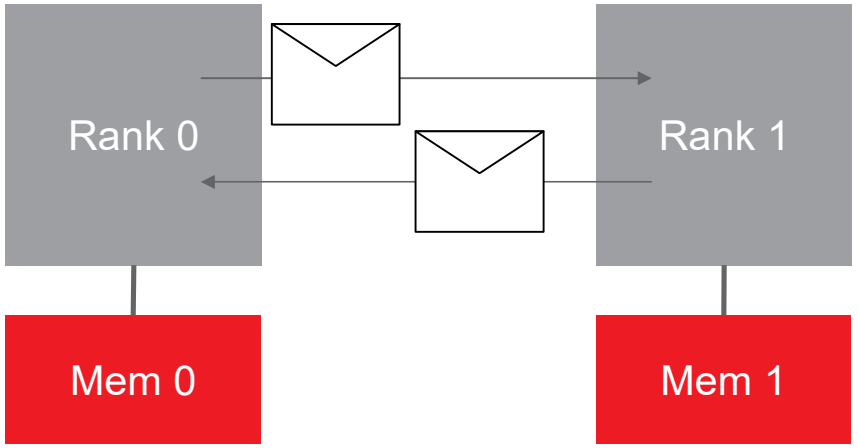
Collective Offload Kernel: ACCL (this presentation)

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

What is MPI?

- The Message Passing Interface (**MPI**) is an **IPC standard**.

```
[…]
comm.send(txb, dst=1)
comm.recv(rxb, src=1)
[…]
```

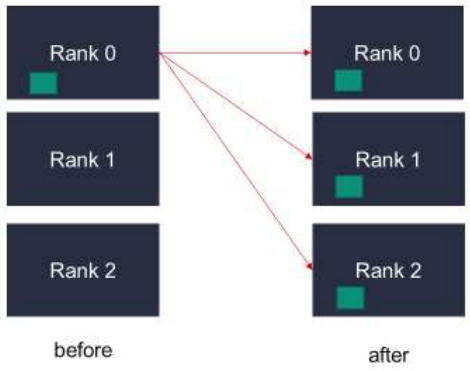



```
[…]
comm.recv(rxb, src=0)
comm.send(txb, dst=0)
[…]
```

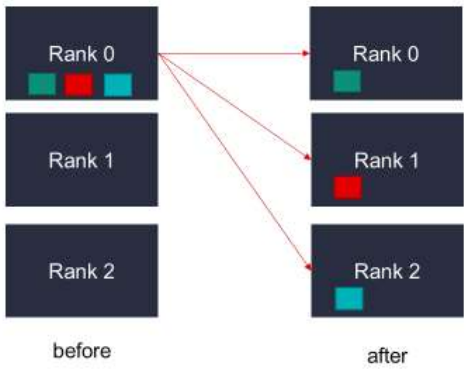
- MPI provides a set of **collective communication primitives (*collectives*)** that one can leverage **to implement highly parallel algorithms**
- Traditionally, CPU handles network stack, coordinates collective operations

What is a MPI collective? Some examples:

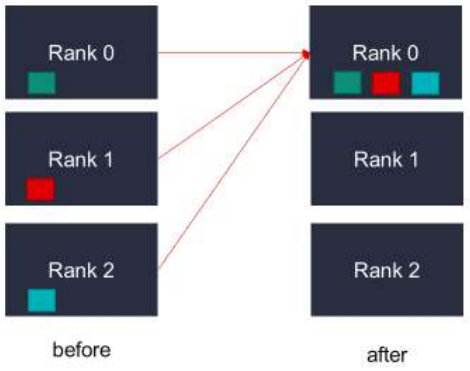
Broadcast



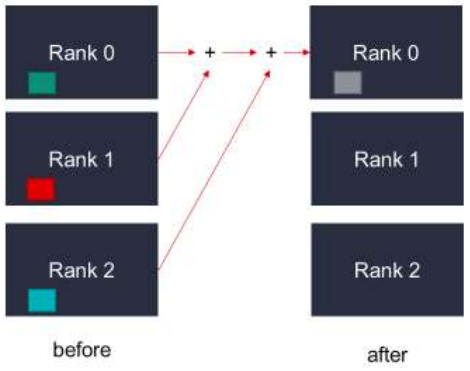
Scatter



(All)Gather



(All)Reduce



AMD

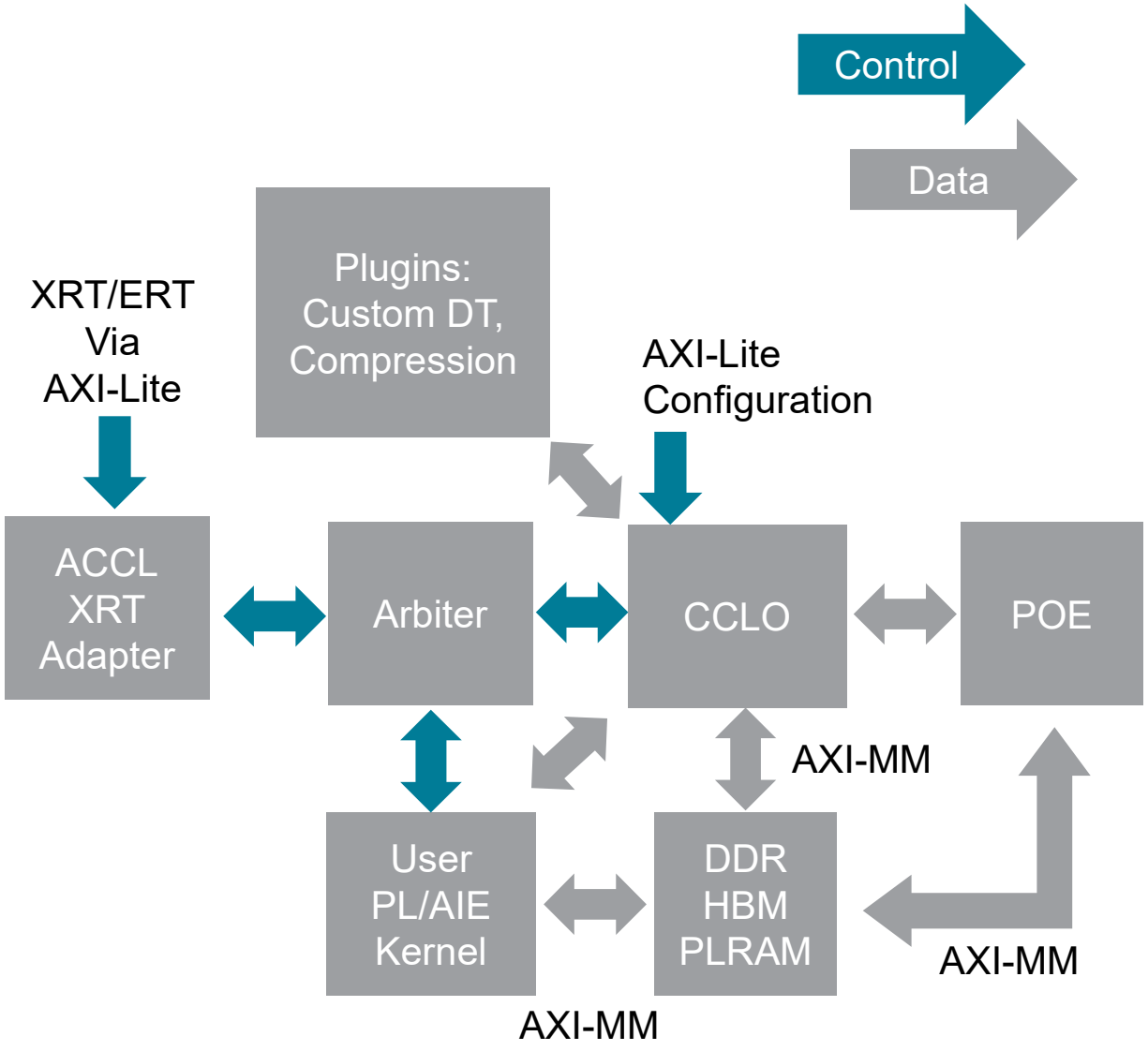
Applicability of MPI Primitives and Collectives

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

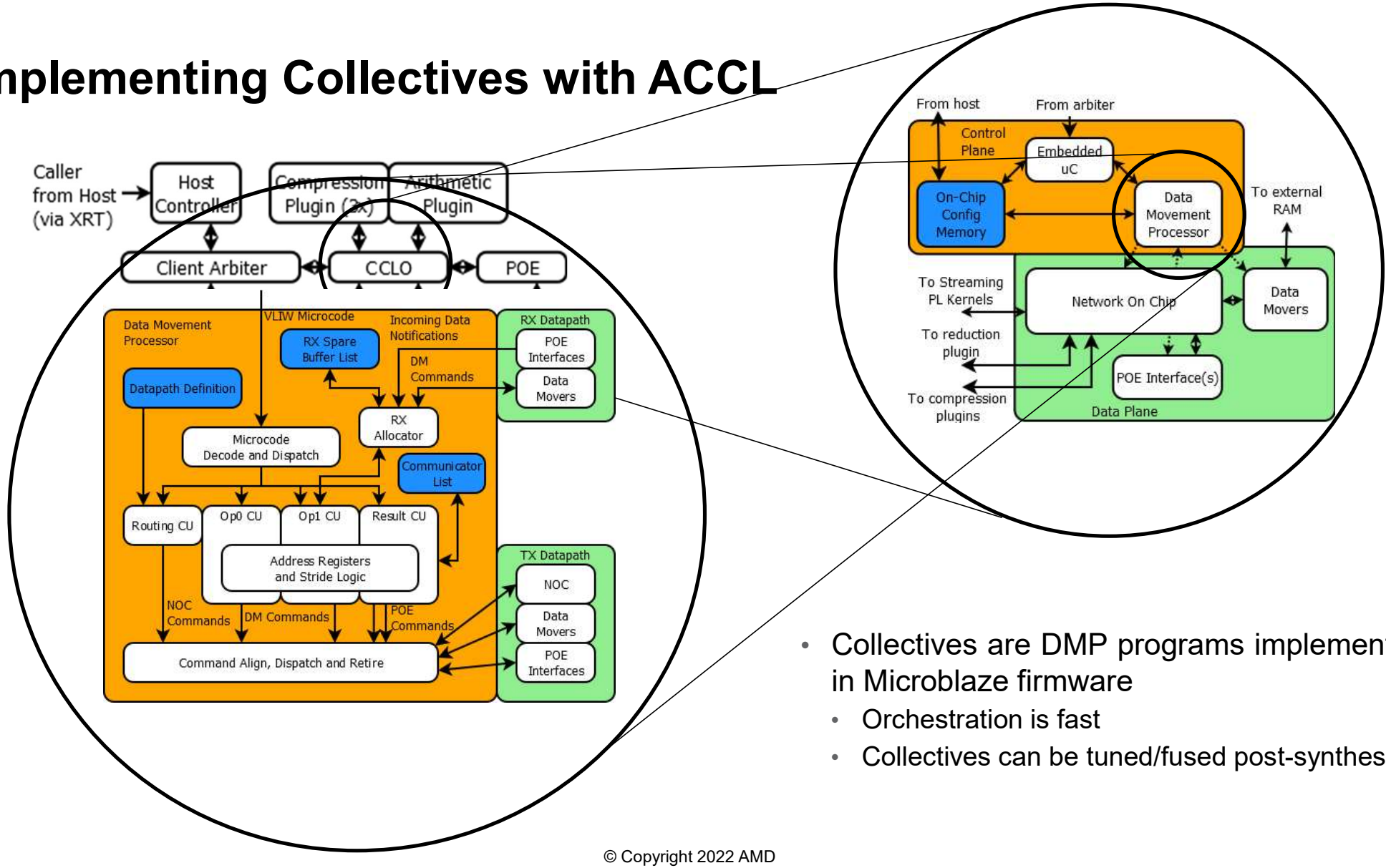
- Implement the 7 key MPI collectives
- Low latency communication control in PL
  - CCL Offload (CCLO) Vitis Kernel
  - Automatic RX buffer management
- POE configurability
  - UDP, TCP currently supported
  - ROCE in development
- Host-less and host-full invocation
  - HLS bindings and host driver provided to users
- Memory-less invocation
  - PL/AIE kernels can talk to CCLO via AXI streams
- Expandability via Plug-Ins
- All kernels Vitis compatible and portable



All connections via AXI-Streams unless otherwise specified

AMD

Implementing Collectives with ACCL

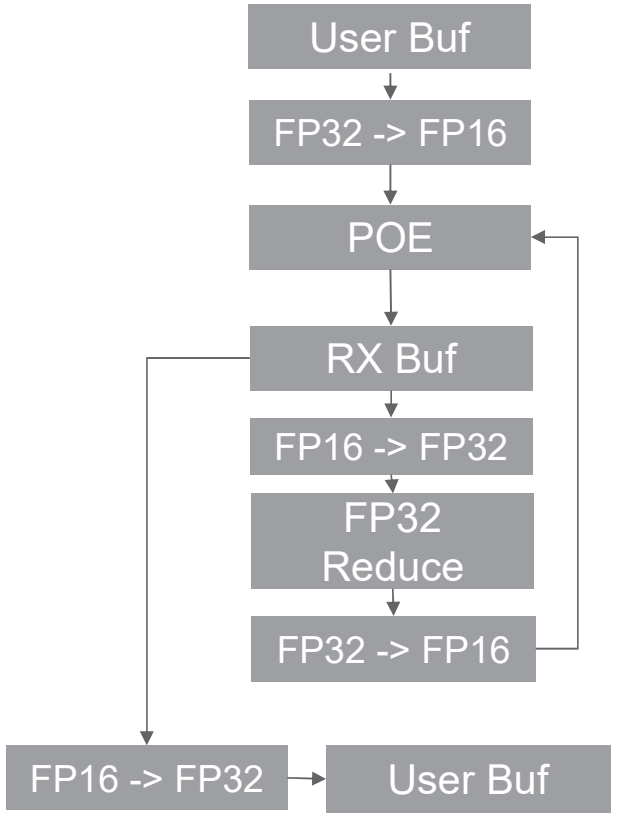


- Collectives are DMP programs implemented in Microblaze firmware
  - Orchestration is fast
  - Collectives can be tuned/fused post-synthesis

CCLO Plugins: Compression and Reduction

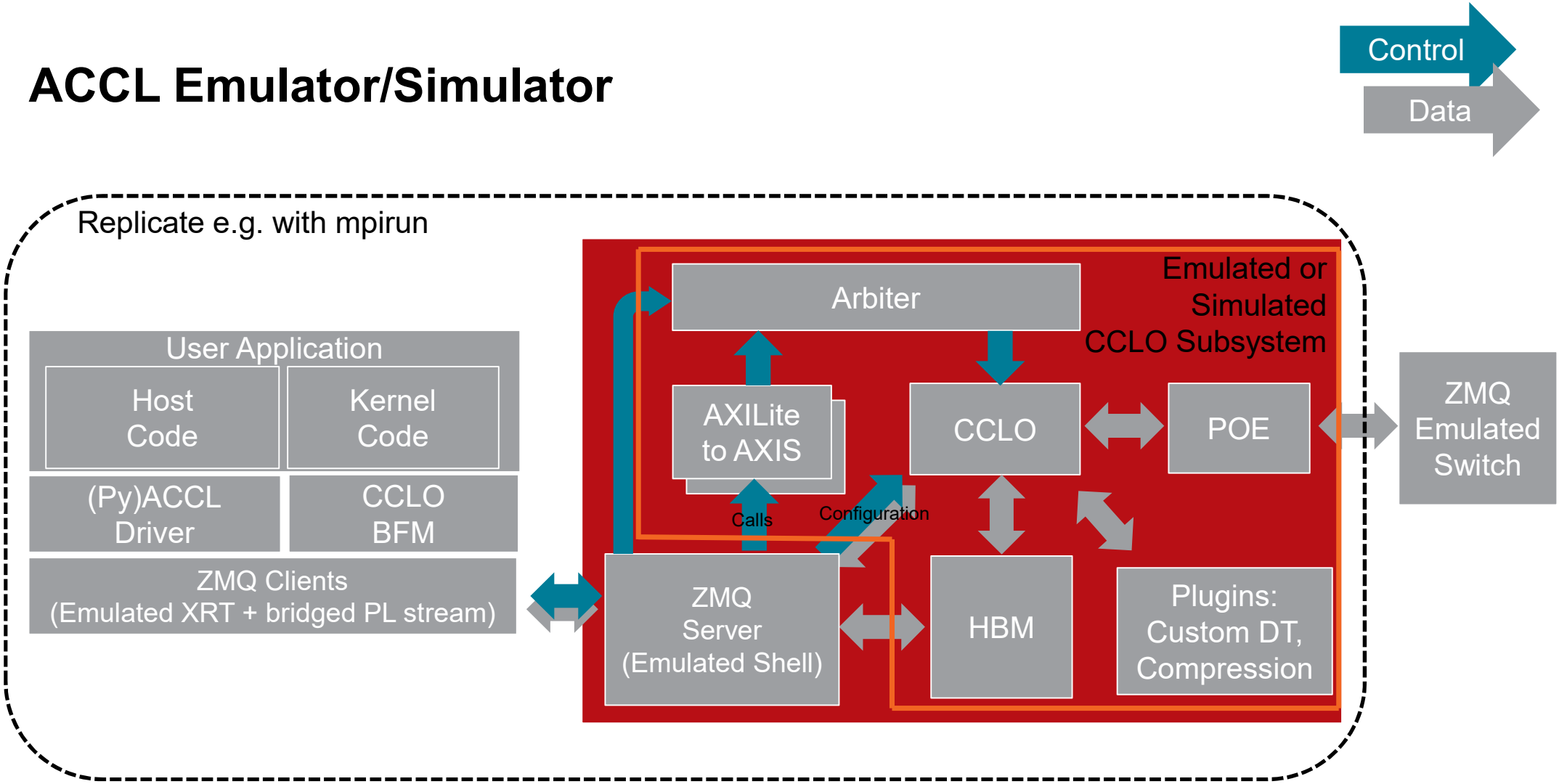
- CCLO supports two types of Plugin Kernels: unary and binary
  - Unary plugins suitable for bump-on-wire compression and encryption
  - Binary plugins suitable for implementing reduction functions
- Unary plugins
  - take one 64B stream input with TDEST, produce a 64B stream output
  - Optionally inspect TDEST on input to select between multiple functions
  - Example provided which implements elementwise cast between FP16 and FP32 (both ways)
  - Always 3 instances, for two operands and one result
- Binary plugins
  - Take two 64B stream inputs with TDEST, produce a 64B stream output
  - Optionally inspect TDEST on first input to select between multiple functions
  - Example provided which implements elementwise SUM, MAX for INT32, INT64, FP16, FP32, FP64
- CCLO configured by driver on which plugin to use and TDEST values corresponding to desired compression/reduction functions

Example: Transparently Compressed Allreduce



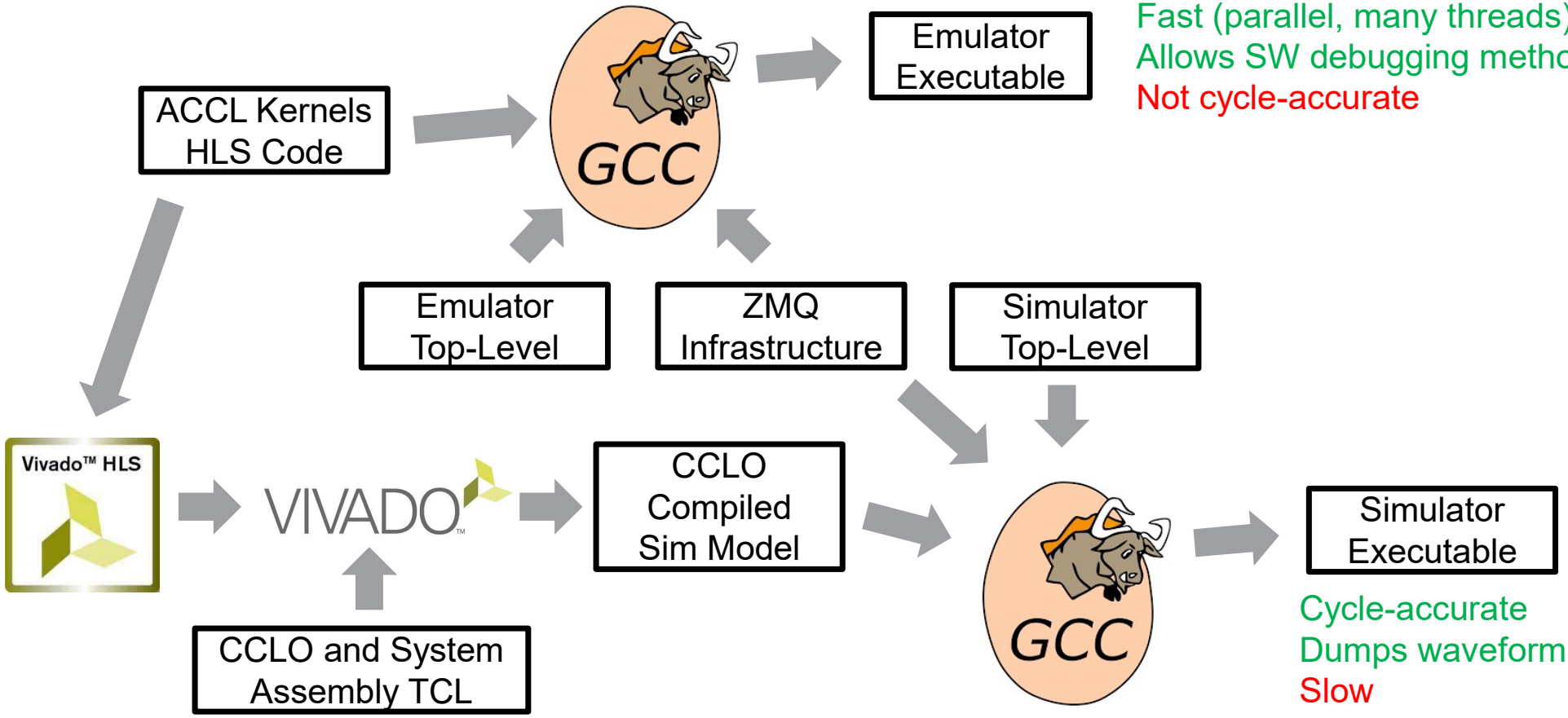
ACCL Emulation and Simulation:
Taking the FPGA out of the Development Loop

ACCL Emulator/Simulator



AMD

Differences between Emulator and Simulator



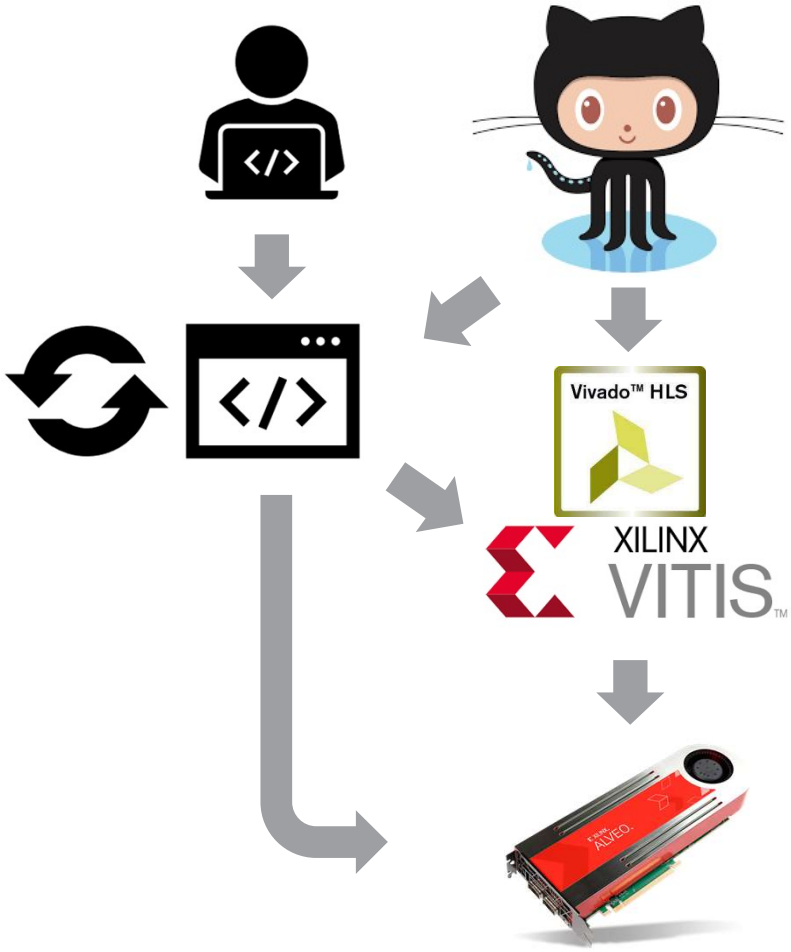
Using ACCL

ACCL open-source components

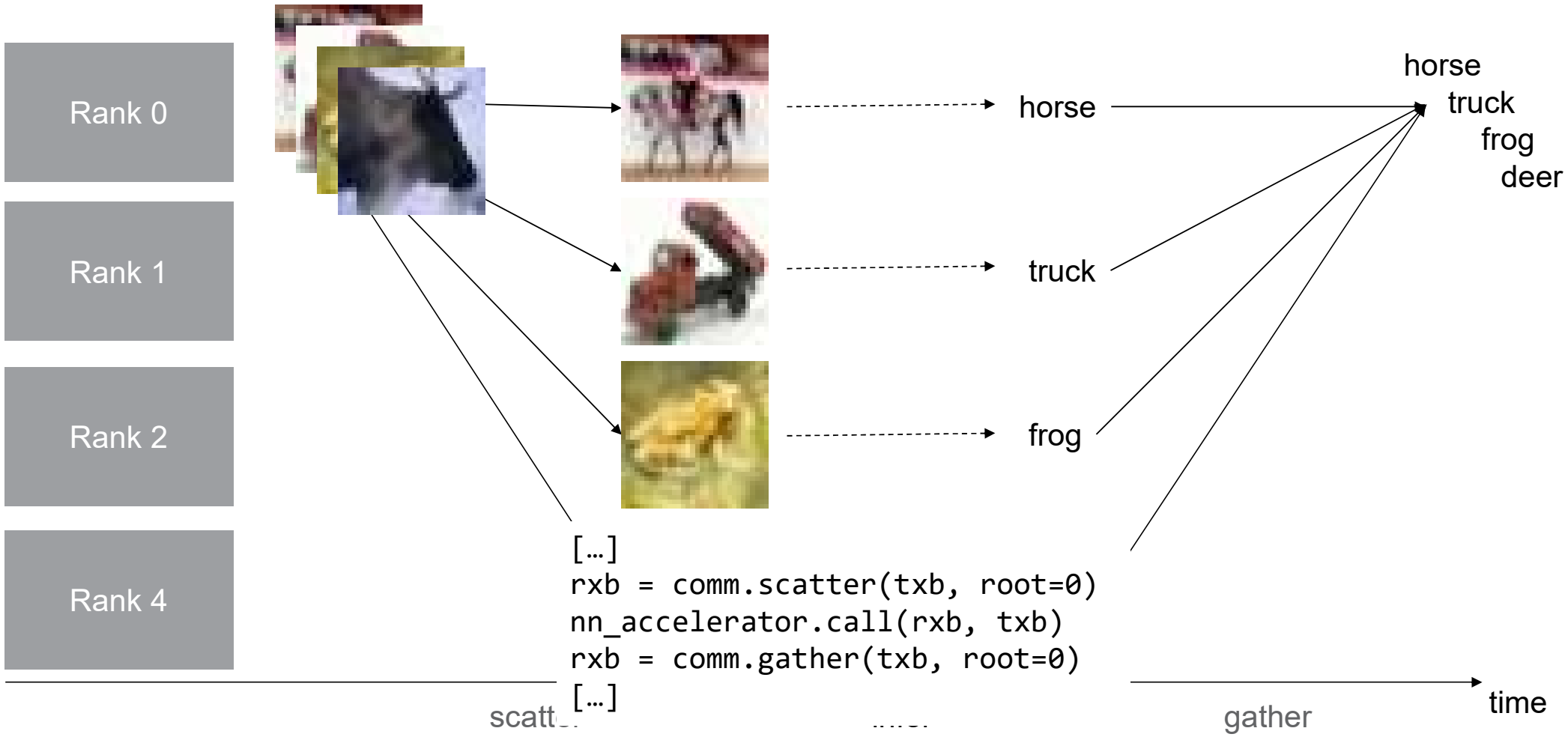
- Main repository: https://github.com/Xilinx/ACCL
  - Code and build automation for collectives orchestration kernel (CCLO) and all other kernels (including example plug-ins)
  - Simulator and Emulator
  - C++ bindings on top of XRT
  - HLS bindings and bus functional model of CCLO for HLS code to interact with
  - Tests and example designs for various Alveo cards
- Python bindings for ACCL: https://github.com/Xilinx/pyaccl
  - Same functionality as C++ bindings but from Python
  - Works on top of the Pynq library
  - Easier to install (via pip)
  - Comprehensive tests against hardware, simulator and emulator

Steps to build ACCL-enabled FPGA application

- Clone ACCL repo(s):
  - https://github.com/Xilinx/ACCL
  - https://github.com/Xilinx/pyaccl
- 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 Vitis
  - Against platform, protocol offload engine (POE), and any application kernels
- Deploy to FPGA



Toy App Example: distributed inference with mpi4py

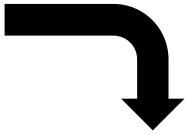


AMD

Toy App Example: distributed inference with pyaccl

mpi4py

```
[…]
rxb = comm.scatter(txb, root=0)
nn_accelerator.call(rxb, txb)
rxb = comm.gather(txb, root=0)
[…]
```



```
[…]
accl.scatter(txb, rxb, root=0)
nn_accelerator.call(rxb, txb)
accl.gather(txb, rxb, root=0)
[…]
```

ACCL
(quick)

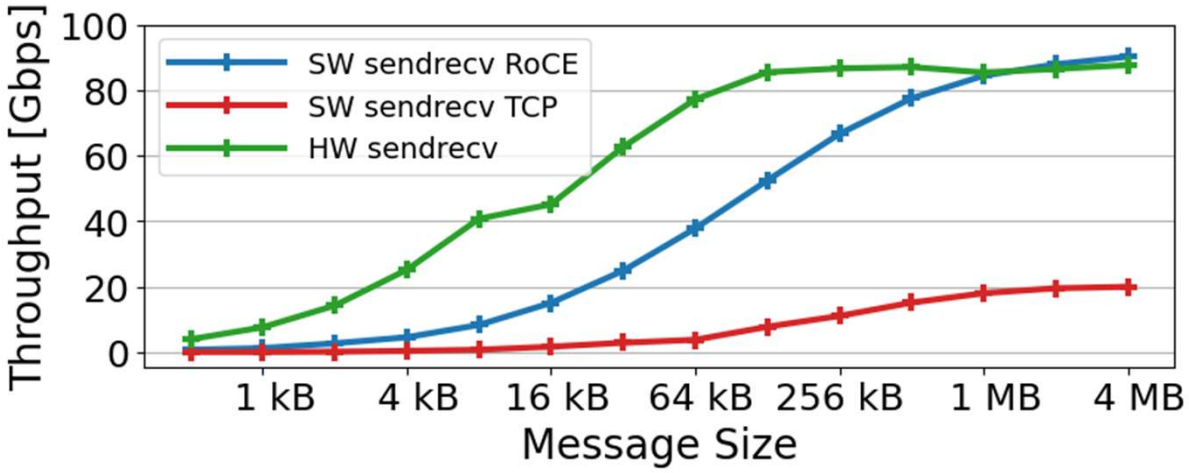
ACCL
(optimized)

```
[…]
ch = accl.scatter(txb, rxb, root=0, to_fpga=True)
ch = nn_accelerator.call(rxb, txb, from_fpga=True, to_fpga=True)
ch = accl.gather(txb, rxb, root=0, async=True, from_fpga=True)
[…]
```

Benchmarking ACCL

ACCL Send & Recv throughput

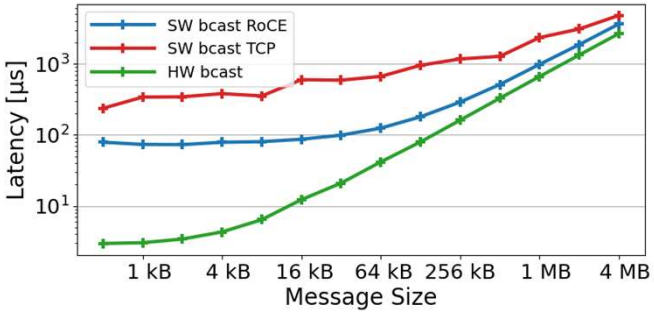
- Evaluation at ETHZ HACC:
  - ACCL dev + Alveo U55C/U280/U250 using PL invocation
  - MPICH 4.0.2, OpenMPI 4.1.3 + OpenUCX 1.13.1 + Mellanox Connex X5 100G
- Performance on U280, U55C, and U250 is similar (design is **portable**)
- ACCL achieves **higher throughput** than MPICH over TCP, comparable to OpenMPI over RDMA:



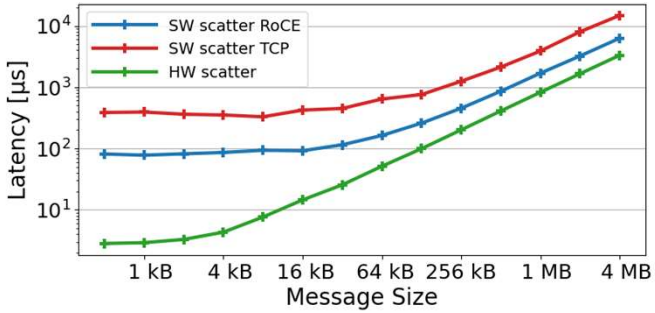
ACCL Collectives Performance

- Latency of Stream-to-Stream Collectives at ETHZ HACC, 8 ranks
- ACCL: PL kernel initiates collective, ACCL orchestrates, data exchanged through AXI Streams
- SW MPI: host initiates and orchestrates, streams flushed and moved host – FPGA via XRT

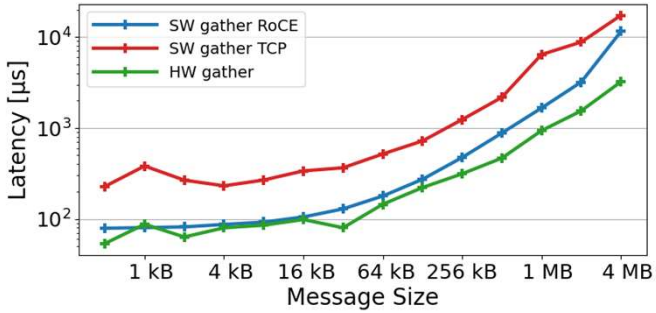
Broadcast



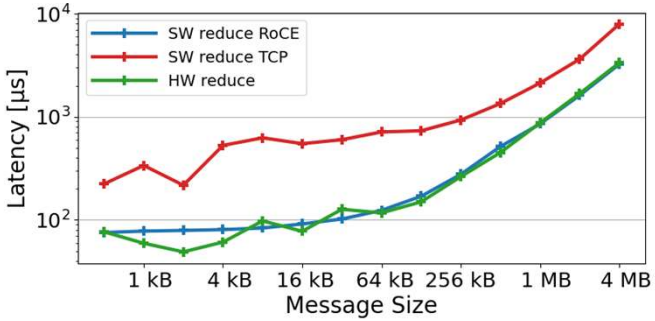
Scatter



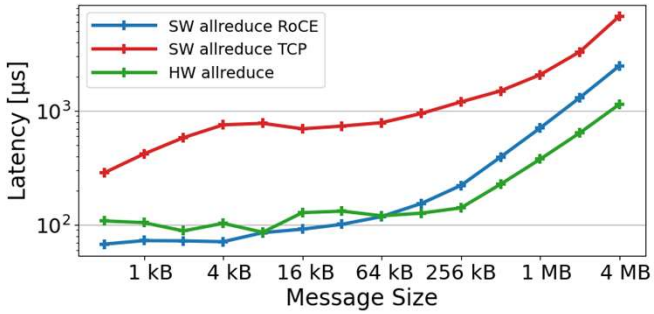
Gather



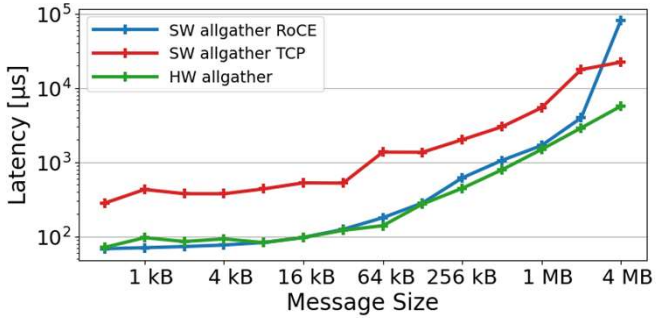
Reduce



All-Reduce

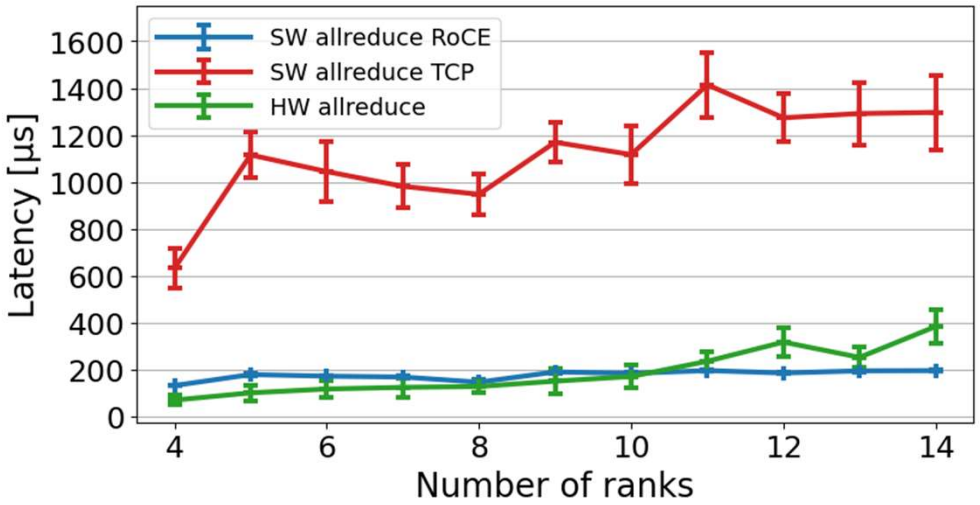


All-Gather



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
- Comparable with RDMA, but tuning still needed



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

Our R&D Roadmap / Wishlist

Active, dynamic project looking for contributions!

Current work threads:

- Full interoperability between communication layers and seamless availability at HACCs
- Tuning (tree collectives, DMP scheduling)
- Versal support and invocation from PS/AIE
- Transparent FPGA-driven data movement through programming framework integration
- More sophisticated (variable rate) compression, adding encryption

Open to suggestions!

Disclaimer & Attribution

Timelines, roadmaps, and/or product release dates shown in these slides are plans only and subject to change.

The information contained herein is for informational purposes only and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD's products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale.


AMD