MVAPICH

MPI, PGAS and Hybrid MPI+PGAS Library

HiBD

High-Performance Big Data

HiDL

High-Performance Deep Learning

# Implementing an MPI Library over Collective Communication Libraries for Habana Accelerators

**Chen-Chun Chen**

**Presented at MUG 2023**

**E-mail: chen.10252@osu.edu**

# Outline

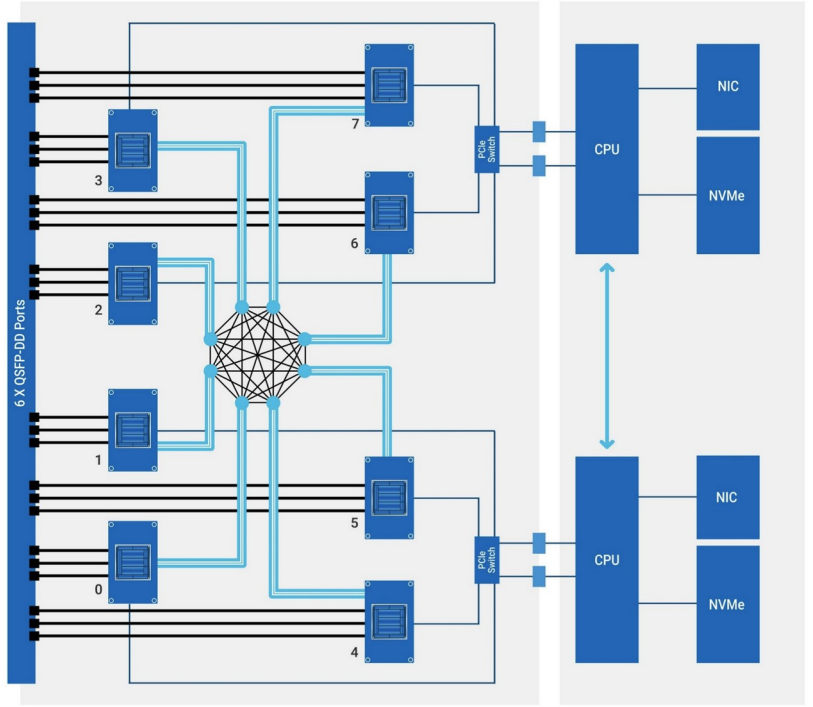
- Introduction
- Motivation
- Implementation
- Evaluation Results
- Conclusion and Future Work

# Introduction: Modern HPC for DL Frameworks

- The emergence of deep learning applications and frameworks
  - Early (2014) frameworks used a single fast GPU
  - Today, parallel training on multiple GPUs and multiple nodes is being supported by most frameworks
  - A lot of fragmentation in the efforts (Horovod, MPI, NCCL, Gloo, gRPC, etc.)
- The development of HPC supports
  - Multi-core/many-core technologies
  - Remote Direct Memory Access (RDMA)-enabled networking (InfiniBand, RoCE, and Slingshot)
  - Solid State Drives (SSDs), Non-Volatile Random-Access Memory (NVRAM), NVMe-SSD
  - Accelerators (NVIDIA GPGPUs, AMD GPUs, Habana Gaudi)

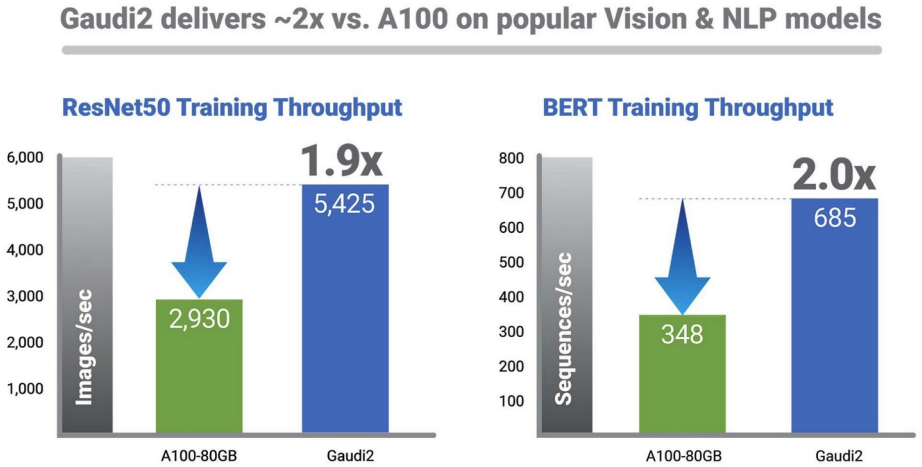
# Introduction: Habana Gaudi

- Founded in 2016 to develop purpose-built AI processors
- Acquired by Intel in 2019
- Designed to optimize AI performance, delivering higher AI efficiency than traditional CPUs and GPUs
  - GEMM engine (MME) excels at matrix multiplication
  - 32 GB of HBM2 memory
- Habana Gaudi within a node:
  - 8 Habana Gaudi accelerators
  - 24 x 100GbE RDMA RoCE for scale-out
  - Non-blocking, all-2-all internal interconnect across Gaudi AI processors
- Training cost saving on Gaudi 2




courtesy: https://extremecomputingtraining.anl.gov/wp-content/uploads/sites/96/2022/11/ATPESC-2022-Track-1-Talk-5-Pandit-Habana-Gaudi.pdf

**Example of Integrated Server with eight Gaudi AI processors, two Xeon CPU and multiple Ethernet Interfaces**




courtesy: https://extremecomputingtraining.anl.gov/wp-content/uploads/sites/96/2022/11/ATPESC-2022-Track-1-Talk-5-Pandit-Habana-Gaudi.pdf

# Introduction: Habana Gaudi (cont'd)

- Hardware:
  - Habana Gaudi, Habana Gaudi 2
  - Cluster with 8 accelerators per node
- Software:
  - SynapseAI
  - A modified TensorFlow or PyTorch image with new implemented computing kernels
  - Habana Collective Communication Library (HCCL)
- Parallel training:
  - Use HCCL for communication between HPUs
  - A modified Horovod with HCCL communication layers

# Outline

- Introduction
- Motivation
- Implementation
- Evaluation Results
- Conclusion and Future Work

# Motivation

- Besides HCCL, are there other communication approaches for HPUs?
  - Traditional GPU-aware MPI libraries
- If we have HCCL, why we need MPI libraries to support HPUs?
- Can we run traditional HPC applications on HPUs?
  - Can the GEMM engine also offer benefits to applications beyond TensorFlow and PyTorch?
- Is HCCL enough?
  - Vender-specific collective communication libraries usually have significant overheads, especially at small messages
  - Can we use other optimization, such as the hybrid designs, to provide better performance over all message sizes?

**Comparison of MPI and NCCL Allreduce latency using 32 GPUs (4 nodes) on a DGX A100 system.**

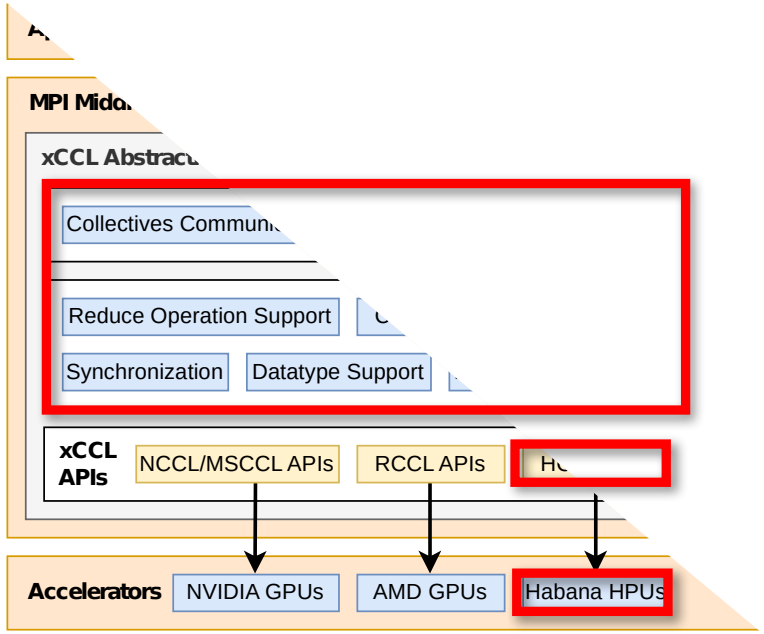
**Comparison of MPI and RCCL Allgather latency using 8 GPUs (4 nodes) on an AMD GPU system.**

# Outline

- Introduction
- Motivation
- Implementation
  - xCCL Abstraction Layer for GPU-aware MPI
  - Built-in Collective Communication Functions
  - Customized Send-recv-based Collective Communication Functions
  - Non-blocking Designs
- Evaluation Results
- Conclusion and Future Work

# xCCL Abstraction Layer for GPU-aware MPI

- Given the success of NCCL library, Habana has proposed very similar communication APIs with compatible functionalities.
  - By changing the prefix to `hccl`
- MVAPICH2-GDR supports NCCL, and MVAPICH-Plus supports xCCL, such as NCCL and RCCL.
- xCCL abstraction layer enables us to use a single API to access third-party libraries.
- A high level of adaptability and productivity by aggregating existing APIs




| Category | NCCL/RCCL/MSCCL | HCCL |
|---|---|---|
| Communicator Creation | ncclCommInitRank | hcclCommInitRank |
| | ncclCommDestroy | hcclCommDestroy |
| Collective Communication | ncclBroadcast | hcclBroadcast |
| | ncclAllReduce | hcclAllreduce |
| | ncclReduce | hcclReduce |
| | ncclReduceScatter | hcclReduceScatter |
| | ncclAllGather | hcclAllGather |
| Group Calls | ncclGroupStart | hcclGroupStart |
| | ncclGroupEnd | hcclGroupEnd |
| Point-to-point Communication | ncclSend | hcclSend |
| | ncclRecv | cclRecv |
| Types | ncclComm_t | hcclComm_t |
| | ncclDataType_t | hcclDataType_t |
| | ncclRedOp_t | hcclRedOp_t |
| Datatypes | ncclFloat | hcclFloat |
| | ncclInt32 | hcclInt32 |
| | ncclUint8 | hcclUint8 |
| Reduce Operations | ncclSum | hcclSum |
| | ncclProd | hcclProd |
| | ncclMax | hcclMax |

# Built-in Collective Functions

- 5 built-in collective communication functions:
  - Broadcast: `hcclBroadcast`
  - AllReduce: `hcclAllreduce`
  - Reduce: `hcclReduce`
  - ReduceScatter: `hcclReduceScatter`
  - AllGather: `hcclAllGather`
- Map these HCCL APIs to our xCCL APIs and directly call those
  - E.g.: `xcclAllReduce` is created on top of `hcclAllReduce`
- Checking mechanism for the supported datatype and reduce operations
  - HCCL support less datatype currently (only support `float` currently)

# Customized Send-recv-based Collective Functions

- The other collective calls are simple send-recv-based communications
  - E.g.: Gather, Scatter, Alltoall
- No vendor-optimized built-in implementation, typically users used to have to implement it on their own.
  - Use `ncclGroupStart`, `ncclGroupEnd`, `ncclSend`, and `ncclRecv` to implement by their own
- We implemented those functions with our high-level xCCL APIs and provided hooks in MPI runtimes
  - Alltoall, Alltoallv, Gather, Gatherv, Scatter, Scatterv, and Allgatherv

# Non-blocking Designs

- Support `MPI_Isend` and `MPI_Irecv (MPI_Wait)`
- Support non-blocking collective operations
  - E.g.: `MPI_Iallreduce`
- The xCCL communication calls are non-blocking operations
  - Remove the synchronization and defer it until the `MPI_Wait` stage
  - Maintain the necessary information between the non-blocking and wait operations

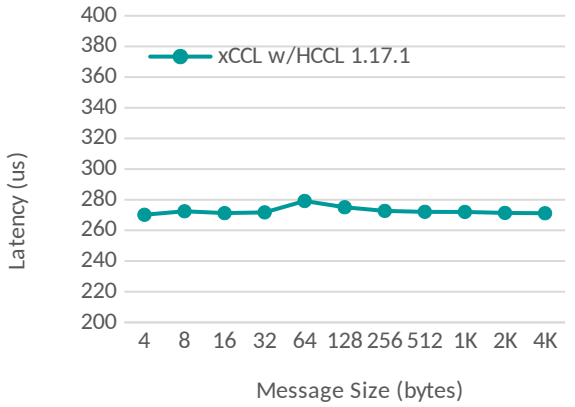
# Outline

- Introduction
- Motivation
- Implementation
- **Evaluation Results**
- **Conclusion and Future Work**

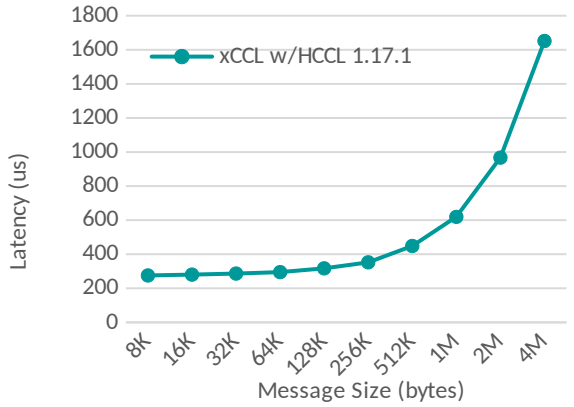
# Evaluation Results

- Platform: **Voyager** at San Diego Supercomputer Center (SDSC)
  - CPU: Intel Xeon Gold 6336Y
  - Memory: 512 GB DDR4
  - Sockets: 2
  - Core/sockets: 24
  - Accelerator/node: **8 Habana Gaudi Processors**
  - Device Memory/HPU: 32 GB HDM2
  - Interconnection: 400 Gbps interconnect from Arista
- Voyager is designed to support research in science and engineering, especially for artificial intelligence and deep learning computing.
- Micro benchmark: OSU Micro-Benchmarks 7.0 with extended features
  - Use Synapse AI Software Suite APIs to support the device buffer on Habana Gaudi
- Application-level benchmark: TansorFlow + Horovod

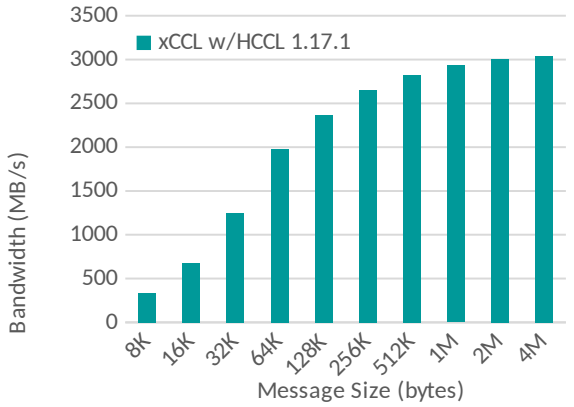
# Micro-Benchmark Evaluation: Point-to-point



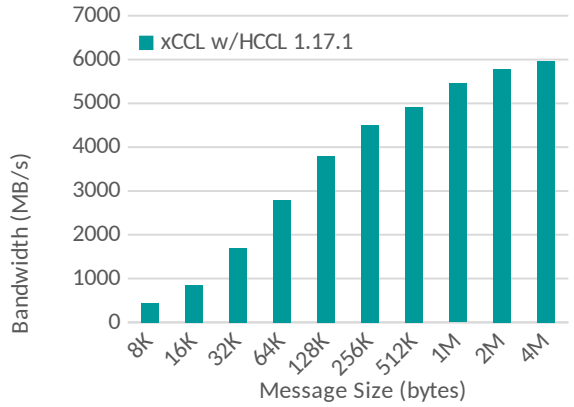

**Intranode Point-to-Point Latency (Small Message)**



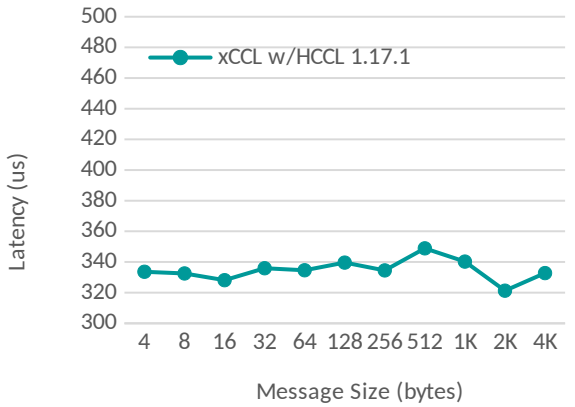

**Intranode Point-to-Point Latency (Large Message)**



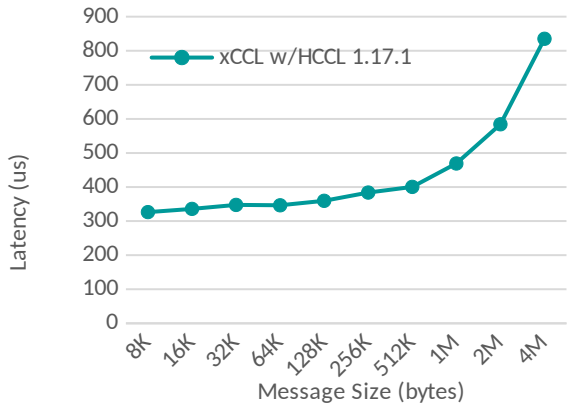

**Intranode Point-to-Point Bandwidth**



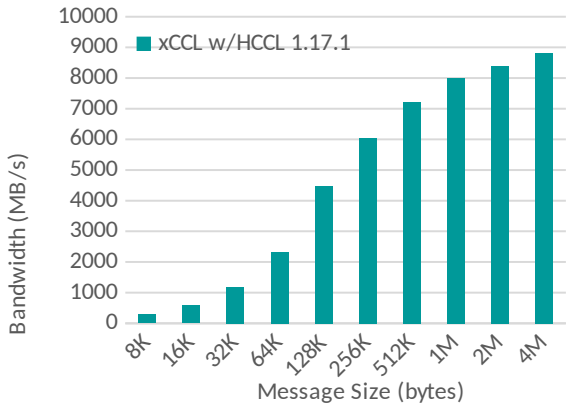

**Intranode Point-to-Point Bi-Directional Bandwidth**



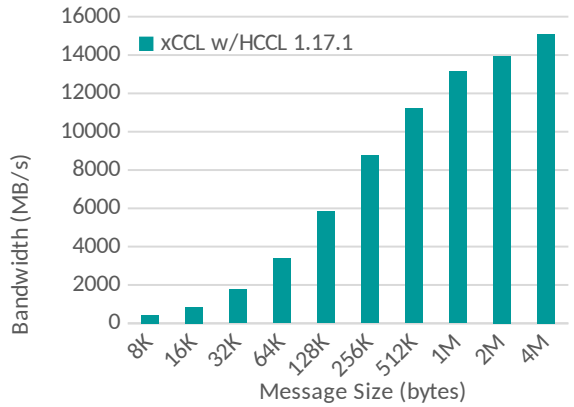

**Internode Point-to-Point Latency (Small Message)**




**Internode Point-to-Point Latency (Large Message)**



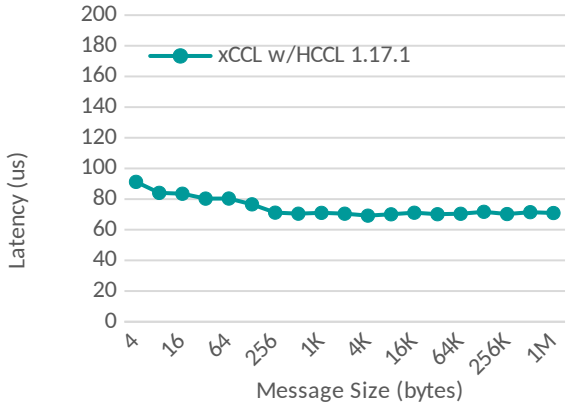

**Internode Point-to-Point Bandwidth**



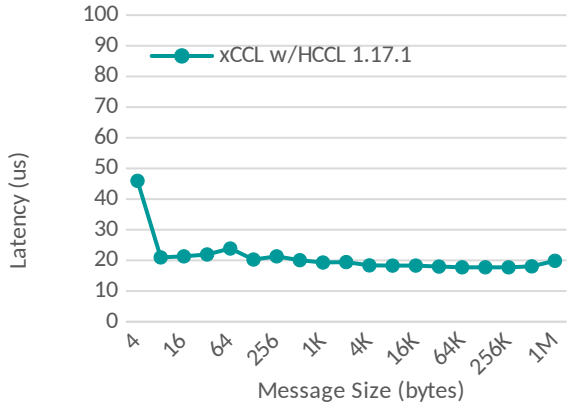

**Internode Point-to-Point Bi-Directional Bandwidth**

- Intranode overhead for latency around 270 µs, internode overhead for latency around 330 µs.
- However, lower latency and higher bandwidth for internode point-to-point operations.

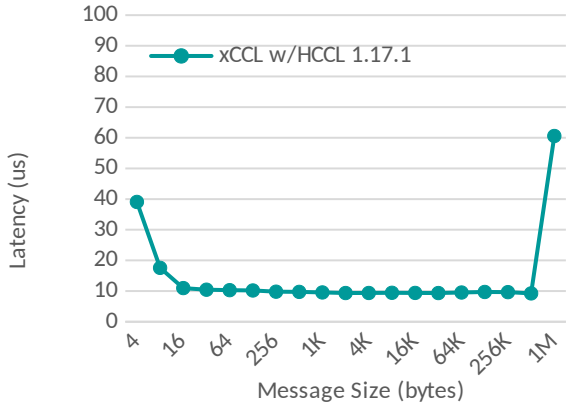
# Micro-Benchmark Evaluation: Collective



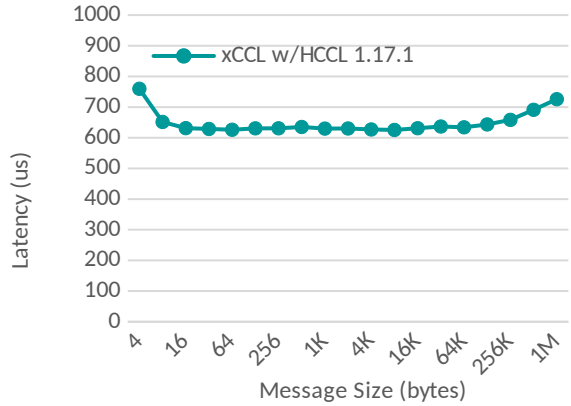

Allreduce w/ HCCL (1 Node, 8 HPUs)



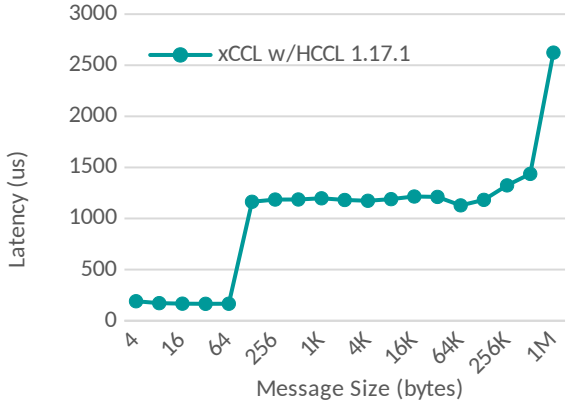

Reduce w/ HCCL (1 Node, 8 HPUs)



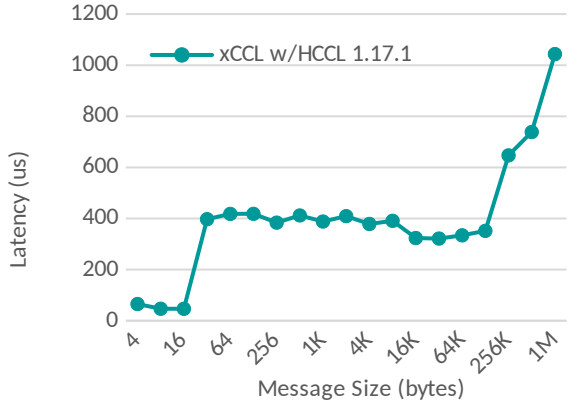

Bcast w/ HCCL (1 Node, 8 HPUs)



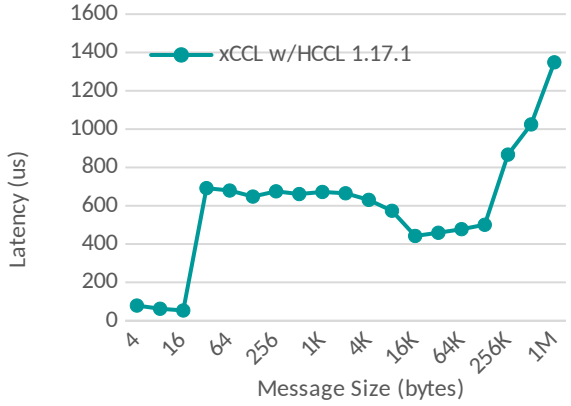

Alltoall w/ HCCL (1 Node, 8 HPUs)



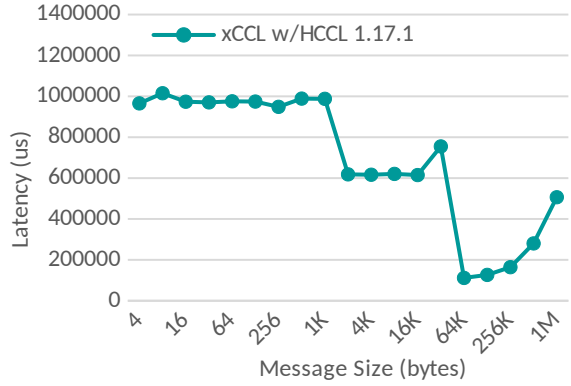

Allreduce w/ HCCL (4 Nodes, 32 HPUs)




Reduce w/ HCCL (4 Nodes, 32 HPUs)



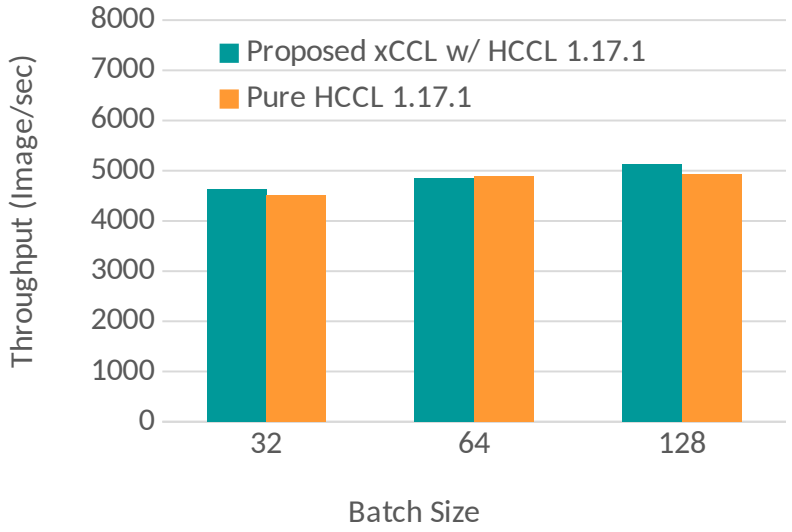

Bcast w/ HCCL (4 Nodes, 32 HPUs)



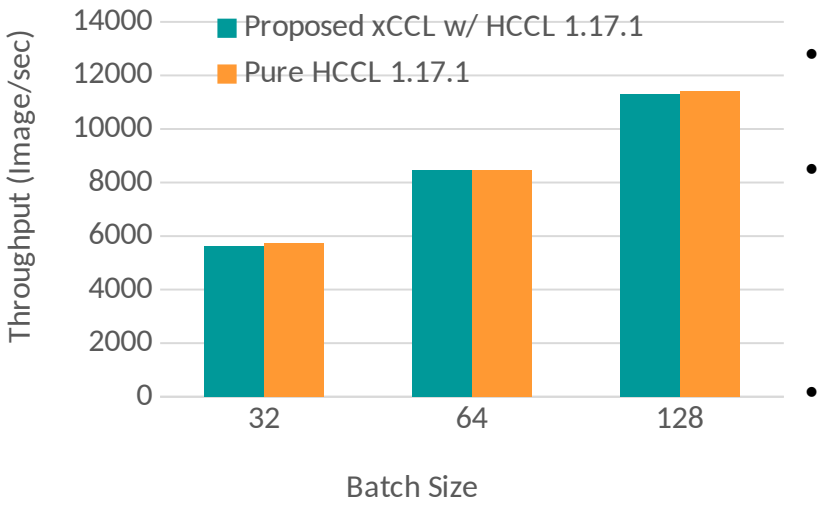

Alltoall w/ HCCL (4 Nodes, 32 HPUs)

- Observe overheads for small messages, especially at the beginning stage, but mostly good on a single node.
- For Allreduce, Reduce, and Bcast on multiple nodes, we observe degradations shown by a step curve around 16 and 64 bytes.
- Poor performance for Alltoall

# Application-Level Evaluation




**1 Node, 8 HPUs**




**4 Nodes, 32 HPUs**

- The container image has already contained prebuilt Habaha TensorFlow and Horovod.
- The computing kernel is replaced by Habana ops through TensorFlow custom ops, the com munication layer in Horovod is implemented by HCCL directly.
- Modify the Horovod communication by replacing all `hcclAllreduce` calls with `MPI_Allreduce` operations.

- On single node, xCCL provides 5139 img/sec throughput with batch size 128, and it is close to the throughput of 4936 img/sec using pure HCCL (4% overhead).
- On multiple nodes (4 nodes), both xCCL and pure HCCL reach the throughput of 11300 img/sec where the overhead is less than 1%
- This evaluation proves that our xCCL designs can be easily extended to new architectures and collective communication libraries with negligible overheads.

# Outline

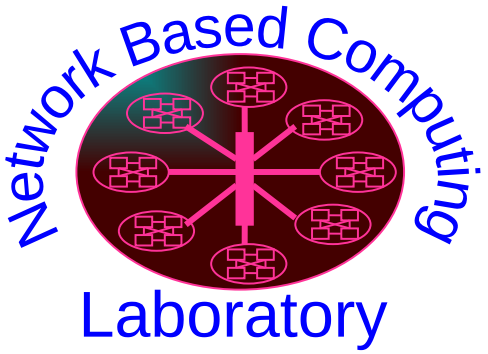
- Introduction
- Motivation
- Implementation
- Evaluation Results
- **Conclusion and Future Work**

# Conclusion and Future Work

- To support Habana Gaudi accelerators deployed on top supercomputers for HPC and Deep Learning applications, it is pertinent to have comprehensive support in the communication libraries for optimal and enhanced communication-level performance.
- Early experience in exploring the novel and state-of-the-art Habana Gaudi Accelerators.
- Support HCCL as the backend with GPU-aware MPI library runtimes in MVAPICH-Plus.
- Support blocking and non-blocking, point-to-point and collective MPI operations.
- Application-level evaluations only show few overheads.
- In the future, we intend to continue this work to develop and provide better strategies for small messages, such as staging approaches, and propose hybrid designs to leverage the best performance over all message sizes.

# Thank You!

chen.10252@osu.edu




*Follow us on*

**https://twitter.com/mvapich**

Network-Based Computing Laboratory
http://nowlab.cse.ohio-state.edu/

MVAPICH
MPI, PGAS and Hybrid MPI+PGAS Library

The High-Performance MPI/PGAS Project
http://mvapich.cse.ohio-state.edu/

HiBD
High-Performance
Big Data

The High-Performance Big Data Project
http://hibd.cse.ohio-state.edu/

HiDL
*High-Performance*
*Deep Learning*

The High-Performance Deep Learning Project
http://hidl.cse.ohio-state.edu/