MPI-xCCL: A Portable MPI Library over Collective Communication Libraries for Various Accelerators

Chen-Chun Chen

Presented at Sixth Annual Workshop on Emerging Parallel and Distributed Runtime Systems and Middleware

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:

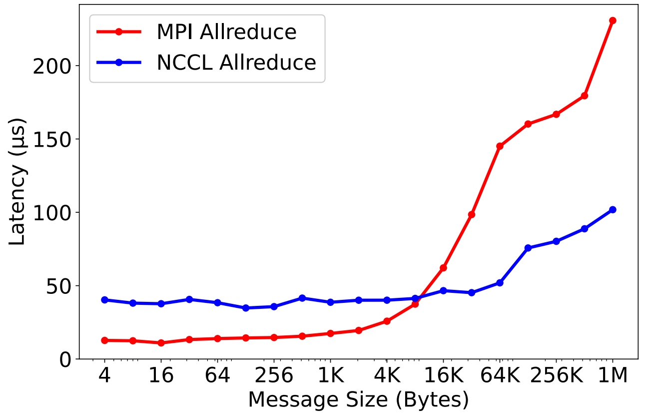
- MPI: communication paradigm used in HPC systems to enable communication across processes, modern GPUs, and new network interconnect.
  - GPU-aware MPI libraries: facilitate direct GPU-to-GPU data transfers
- Vendor-specific communication libraries: GPU vendors provide it to attain superior collective communication performance, particularly tailored for deep learning (DL) applications.
  - E.g.: NVIDIA Collective Communication Library (NCCL), ROCm Collective Communication Library (RCCL), Habana Collective Communications Library (HCCL), Microsoft Collective Communication Library (MSCCL)
- Application developers are often responsible for porting or updating their codes to utilize these vender-communication APIs.
  - It requires extensive knowledge and can lead to decreased productivity and potential inconsistencies across various hardware platforms.

Outline

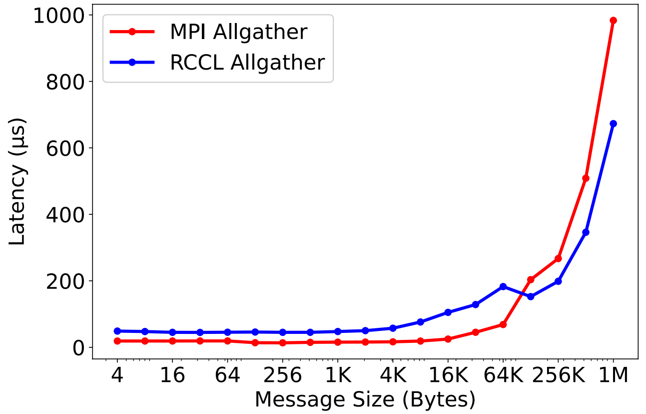
- Introduction
- Motivation
- Implementation
- Evaluation Results
- Conclusion and Future Work

Motivation

- Why not rely solely on traditional MPI libraries or vendor-specific communication libraries (CCLs)?
- CCLs have superior performance for **larger** message transfers
  - Higher overheads for small messages
- High efforts for porting previous designs to use CCLs
  - CCLs do not adhere to the MPI standard
  - Consider the emergence of new architectures and evolving communication libraries



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.

Motivation (cont'd)

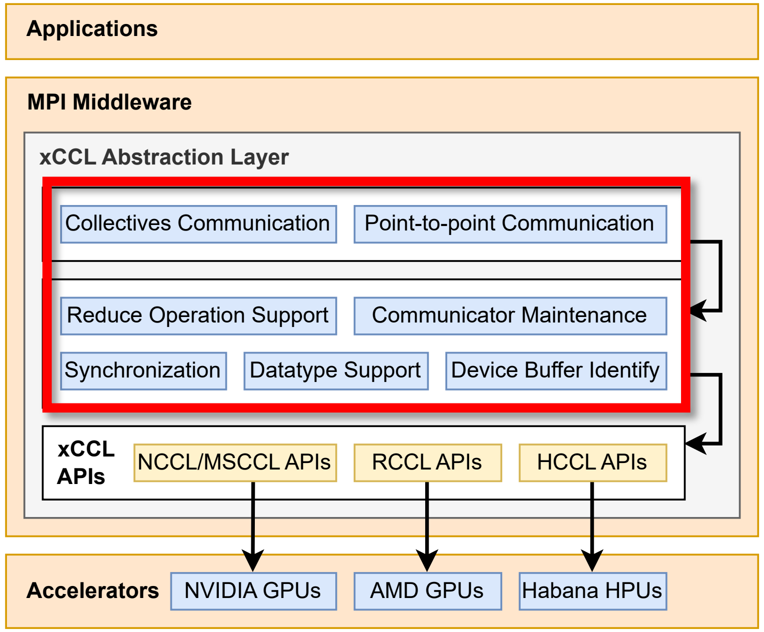
- MPI-xCCL: a unified, portable communication interface that supports various vendor accelerators, enabling users and developers to dynamically leverage the best features from diverse implementations in an application-transparent manner
- User perspective:
  - Utilize different CCLs across architectures without modifying their code and with standard MPI APIs.
  - Manage the complexities of underlying CCL APIs and logic, e.g.: stream handling
  - Support automatic error handling , e.g.: falling back to traditional MPI communication
  - Include common MPI non-blocking collective operations
  - Optimize performance across a wide range of message sizes by using the hybrid designs
  - Take advantage of the same features and functionalities provided by pure CCLs
- Developer perspective:
  - Provide a unified layer for implementing collective operations, eliminating the need to customize algorithms and functions for each new CCL
  - Provide a scalable design that can be easily extended to support upcoming architectures and CCLs

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
  - Hybrid Designs
- Evaluation Results
- Conclusion and Future Work

xCCL Abstraction Layer for GPU-aware MPI

- Given the success of NCCL library, other venders have proposed very similar communication APIs with compatible functionalities.
  - RCCL/MSCCL: keep using the prefix to `nccl`
  - HCCL: by changing the prefix to `hccl`
- 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: `ncclBroadcast/hcclBroadcast`
  - AllReduce: `ncclAllreduce/hcclAllreduce`
  - Reduce: `ncclReduce/hcclReduce`
  - ReduceScatter: `ncclReduceScatter/hcclReduceScatter`
  - AllGather: `ncclAllGather/hcclAllGather`
- Map these NCCL and HCCL APIs to our xCCL APIs and directly call those
  - E.g.: `xcclAllReduce` is created on top of `ncclAllReduce/hcclAllReduce`
- Checking mechanism for the supported datatype and reduce operations
  - Note that HCCL support less datatype currently (only support `float` currently)

Customized Send-recv-based Collective Functions

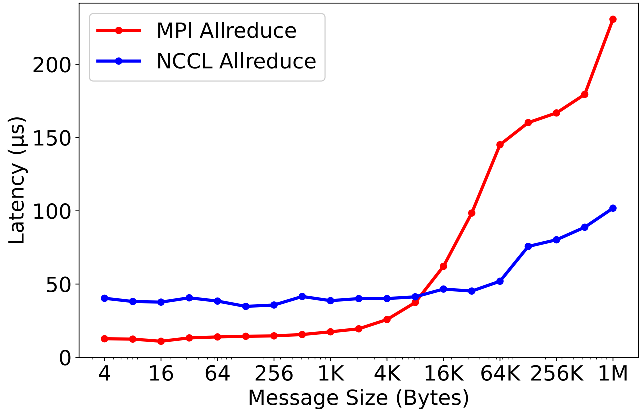
- The other collective calls are simple send-recv-based communications
  - E.g.: Gather, Scatter, Alltoall
- No vendor-optimized built-in implementation, typically users used to have to implement it on their own.
  - Use `ncclGroupStart`, `ncclGroupEnd`, `ncclSend`, and `ncclRecv` to implement by their 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

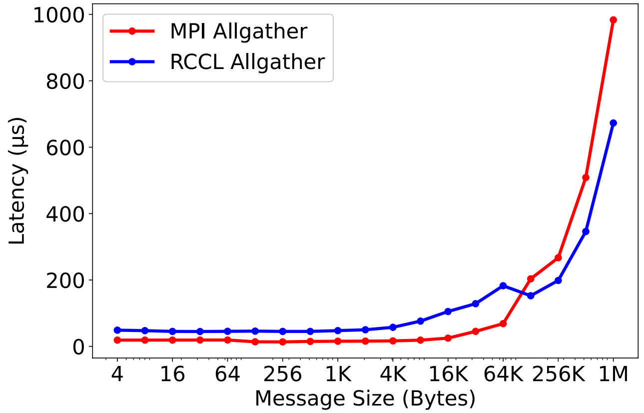
- Support `MPI_Isend` and `MPI_Irecv (MPI_Wait)`
- Support non-blocking collective operations
  - E.g.: `MPI_Iallreduce`
- The xCCL communication calls are non-blocking operations
  - Remove the synchronization and defer it until the `MPI_Wait` stage
  - Maintain the necessary information between the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

Hybrid Designs

- Hybrid designs enable users to leverage both vendor-optimized collective communication libraries and the existing MPI implementations.
  - In fact, modern MPI libraries utilize different protocols and algorithms according to different conditions, such as system architectures, MPI operations, and message sizes.
- With the xCCL Abstraction Layer designs, each operation can be easily encapsulated into one of the MPI algorithms and be called as one of the regular MPI implementations in the tuning tables.
- Tune the tuning tables offline, and during runtime, the hybrid designs select the most optimal solution from the tuning tables.



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

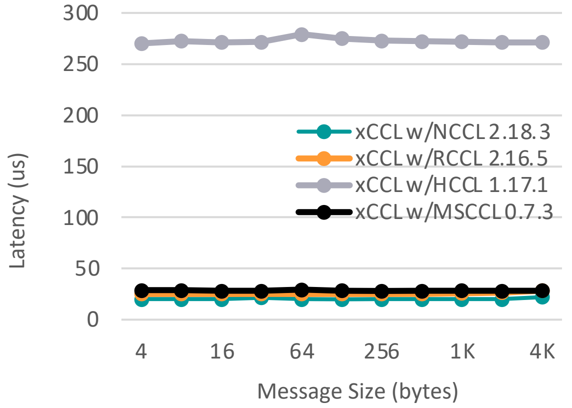
- Introduction
- Motivation
- Implementation
- Evaluation Results
- Conclusion and Future Work

Evaluation Results

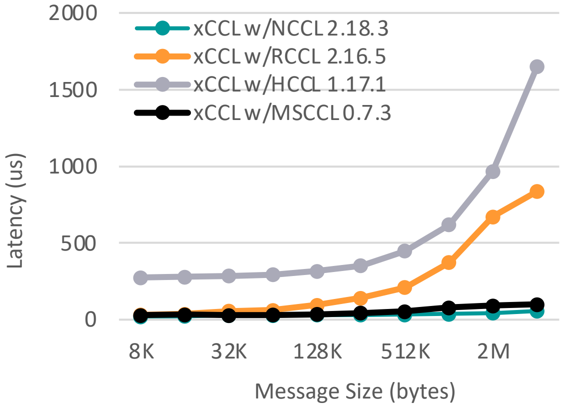
System Component	ThetaGPU (ALCF) NVIDIA	MRI (in-house cluster) AMD	Voyager (SDSC) Habana
CPU	AMD EPYC 7742	AMD EPYC 7713	Intel Xeon Gold 6336Y
Memory	1 TB DDR4	256 GB DDR4	512 GB DDR4
Sockets	2	2	2
Core/socket	64	64	24
Accelerator/node	8 NVIDIA DGX A100 GPUs	2 AMD MI100 GPUs	8 Habana Gaudi Processors
Device Memory/GPU(HPU)	40GB HBM2	32 GB HDM2	32 GB HDM2

- Micro benchmark:
  - OSU Micro-Benchmarks 7.2 for NVIDIA and AMD platforms
  - OSU Micro-Benchmarks 7.0 with extended features for Habana platform
    - Use Synapse AI Software Suite APIs to support the device buffer on Habana Gaudi
- Application-level benchmark: TensorFlow + Horovod

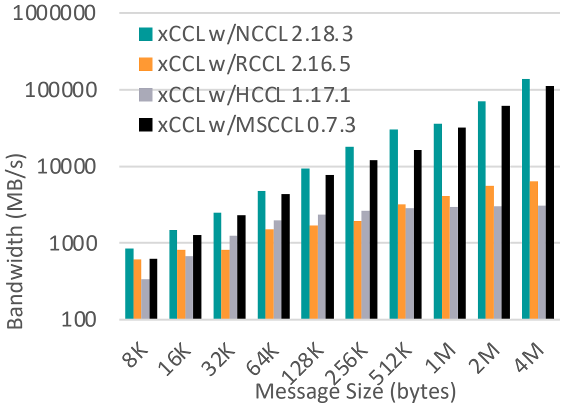
Micro-Benchmark Evaluation: Point-to-point



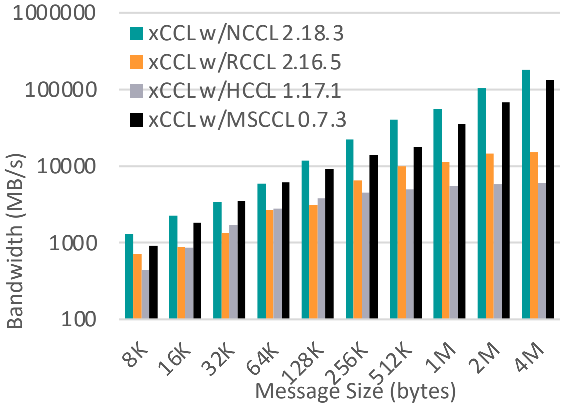
Intranode Point-to-Point Latency (Small Message)



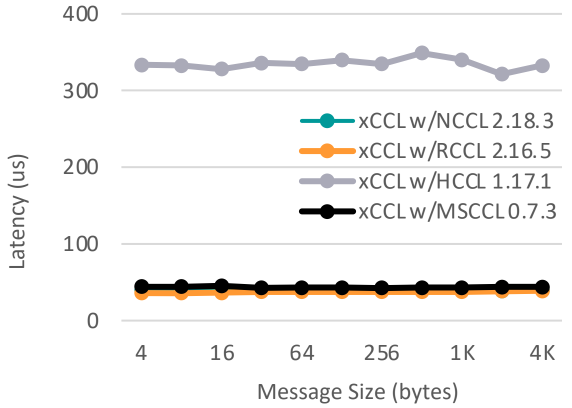
Intranode Point-to-Point Latency (Large Message)



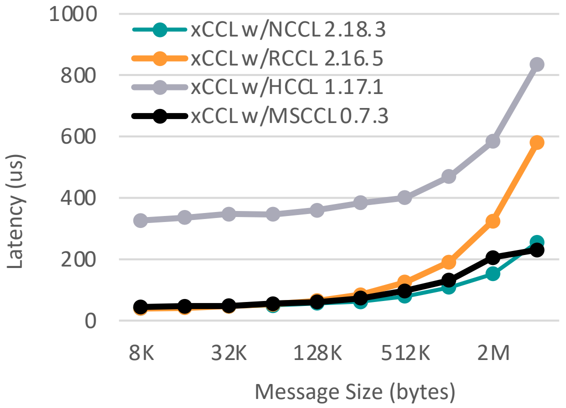
Intranode Point-to-Point Bandwidth



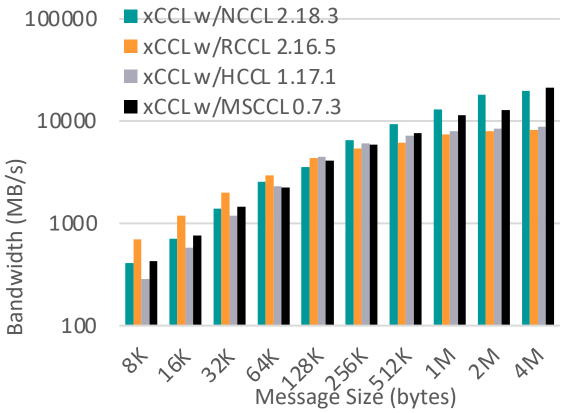
Intranode Point-to-Point Bi-Directional Bandwidth



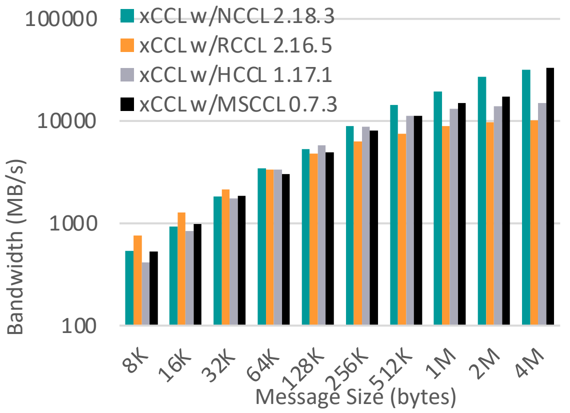
Internode Point-to-Point Latency (Small Message)



Internode Point-to-Point Latency (Large Message)



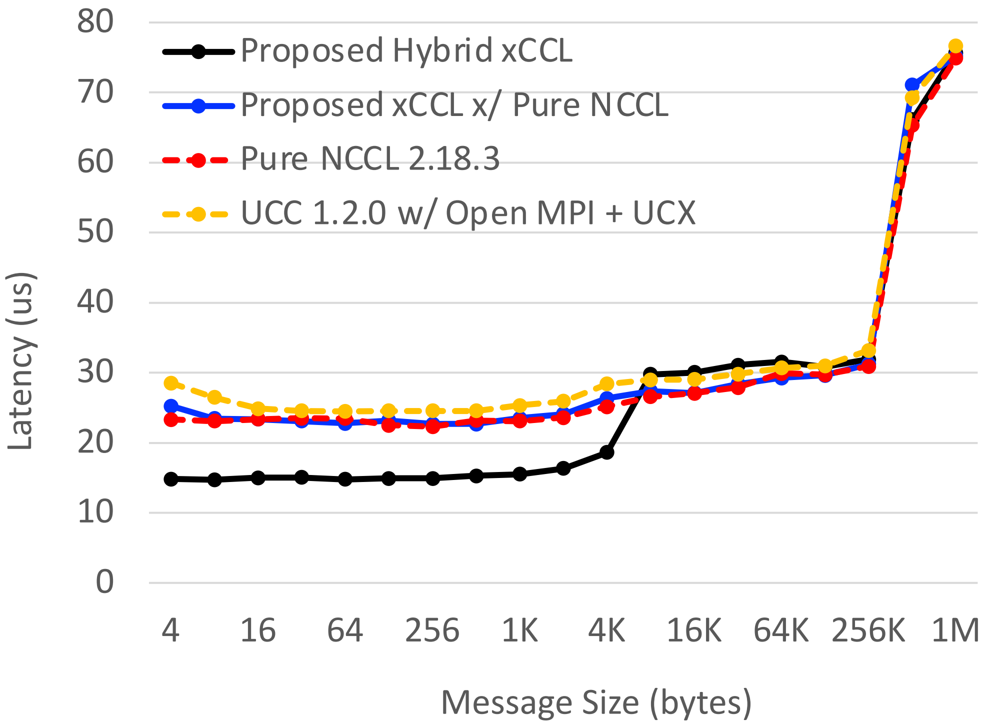
Internode Point-to-Point Bandwidth



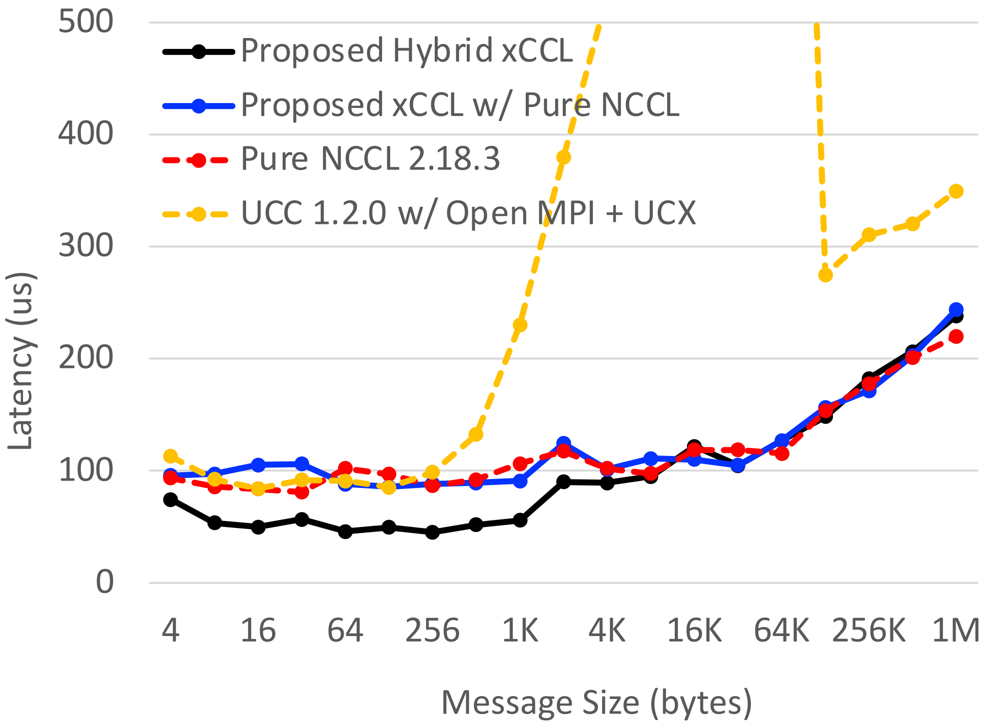
Internode Point-to-Point Bi-Directional Bandwidth

- Our designs can facility both intra-node and inter-node, blocking and non-blocking point-to-point communication with varies collective communication libraries on different platforms
- There is a similar trend of inter-node compared to the results of intra-node numbers.

Micro-Benchmark Evaluation: Collective - NCCL



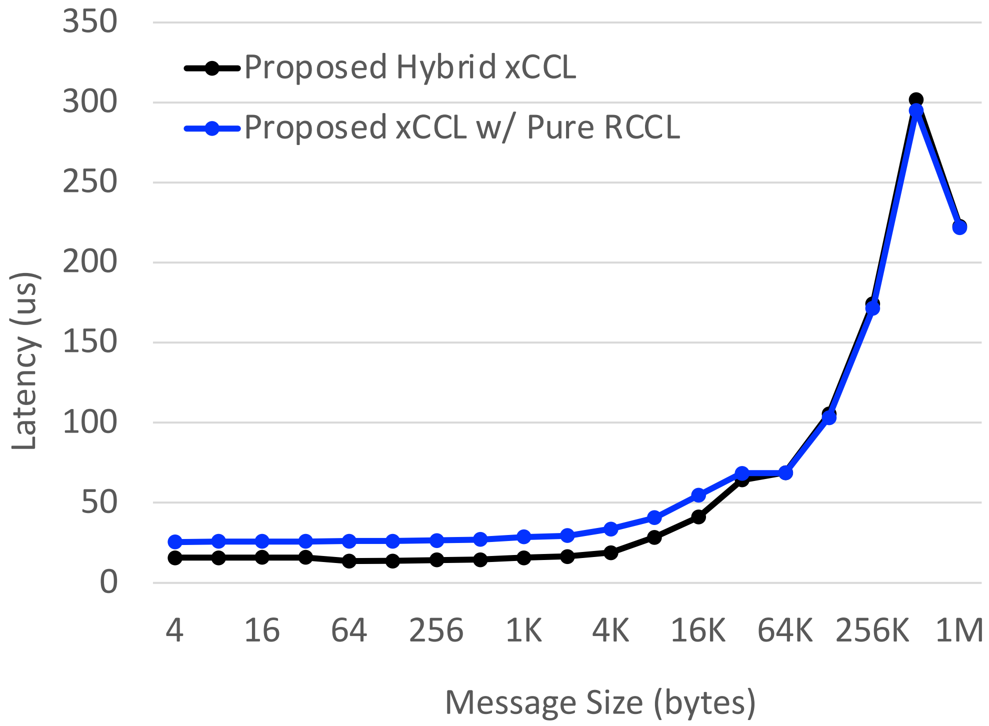
Reduce w/ NCCL (1 Node, 8 GPUs)



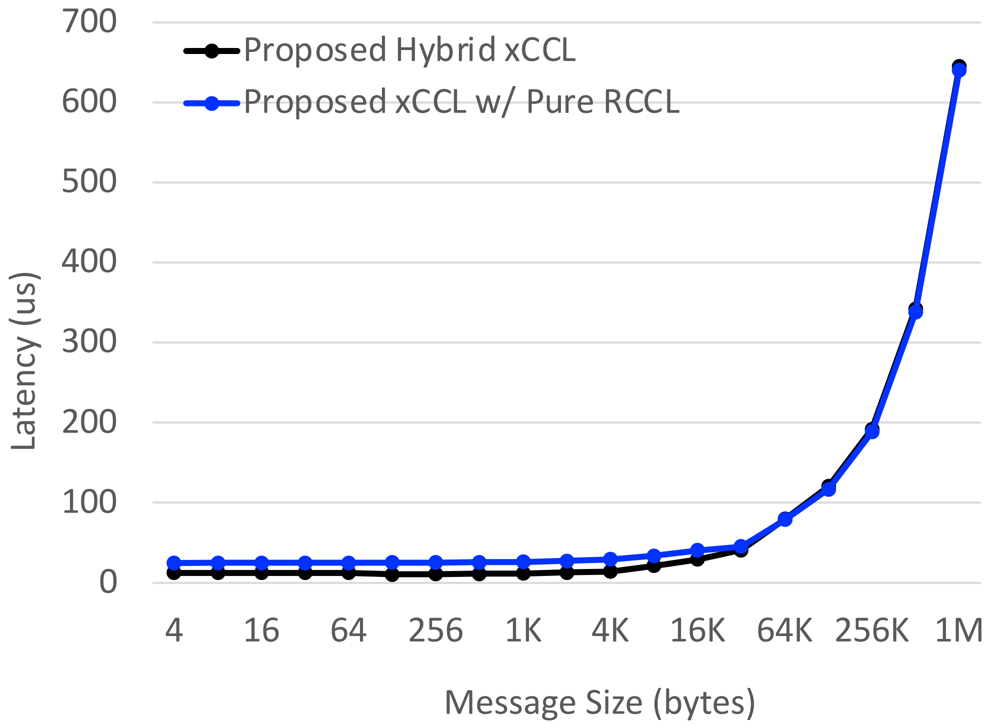
Allreduce w/ NCCL (16 Node, 128 GPUs)

- The performance of proposed pure xCCL designs (blue lines) mirrors that of original vendor-specific xCCL (dotted red lines), highlighting minimal overhead in our implementation.
- The proposed hybrid xCCL (black lines) achieves even lower small message latency.
- Reduce latencies shrink from 23 to 14 µs for small messages (<8KB) in 1-node case.
- Compared our results to Open MPI + UCX + UCC also reveals our designs’ reduced overhead.

Micro-Benchmark Evaluation: Collective - RCCL



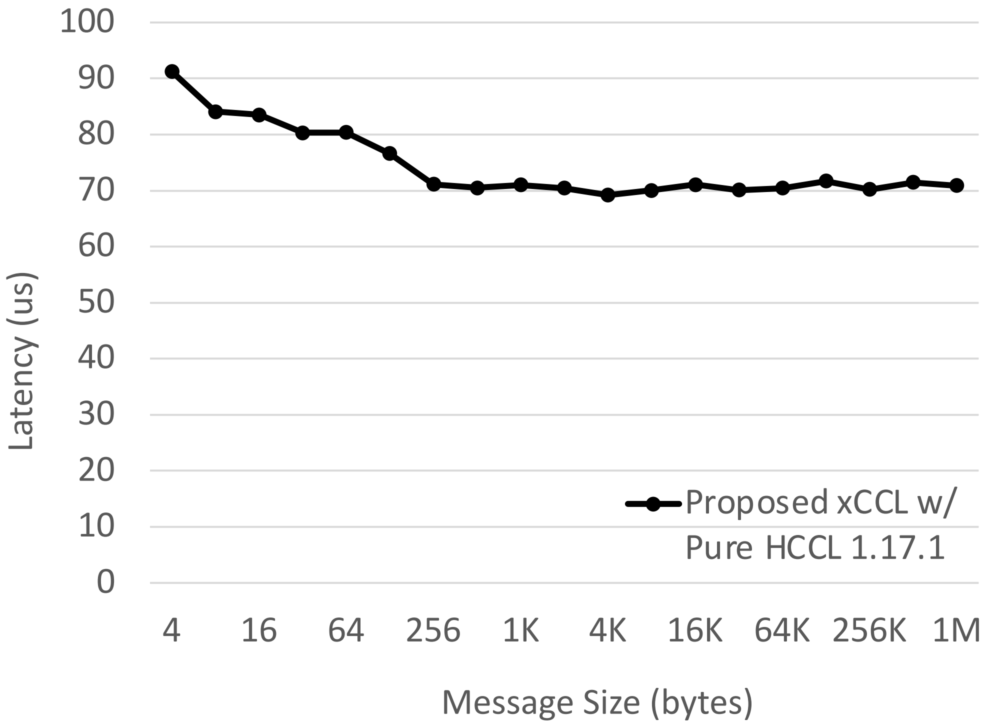
Alltoall w/ RCCL (1 Node, 2 GPUs)



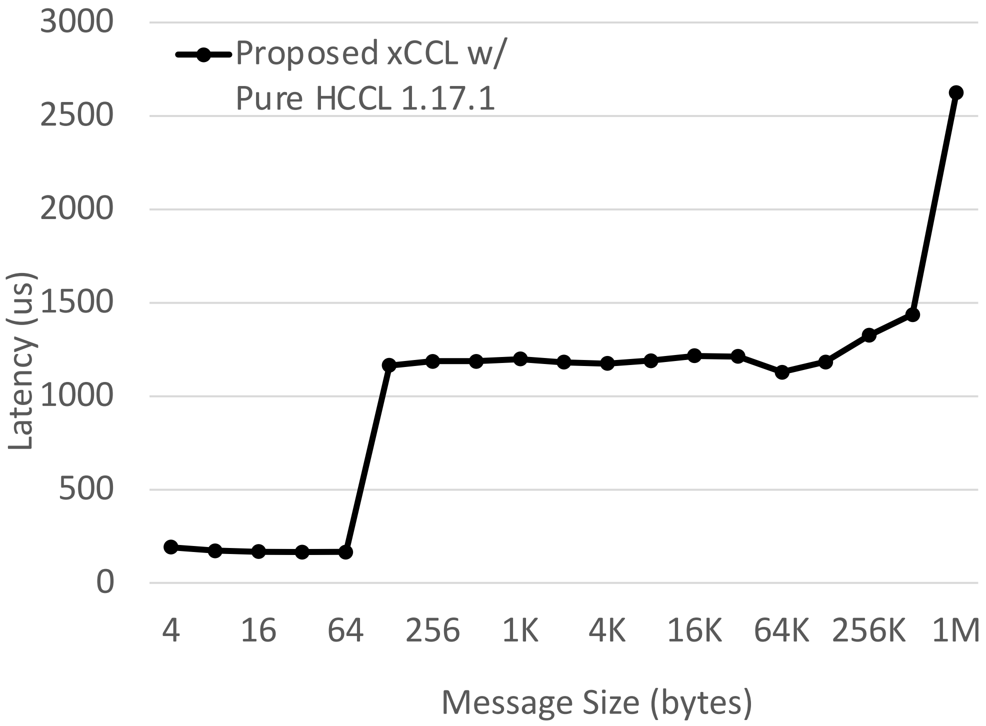
Reduce w/ RCCL (8 Node, 16 GPUs)

- While no RCCL benchmarks exist for AMD architectures, outcomes suggest our designs' adaptability to new architectures.
- Better performance for small messages (<32KB) with the proposed hybrid designs in both 1-node and 8-node cases.

Micro-Benchmark Evaluation: Collective - HCCL



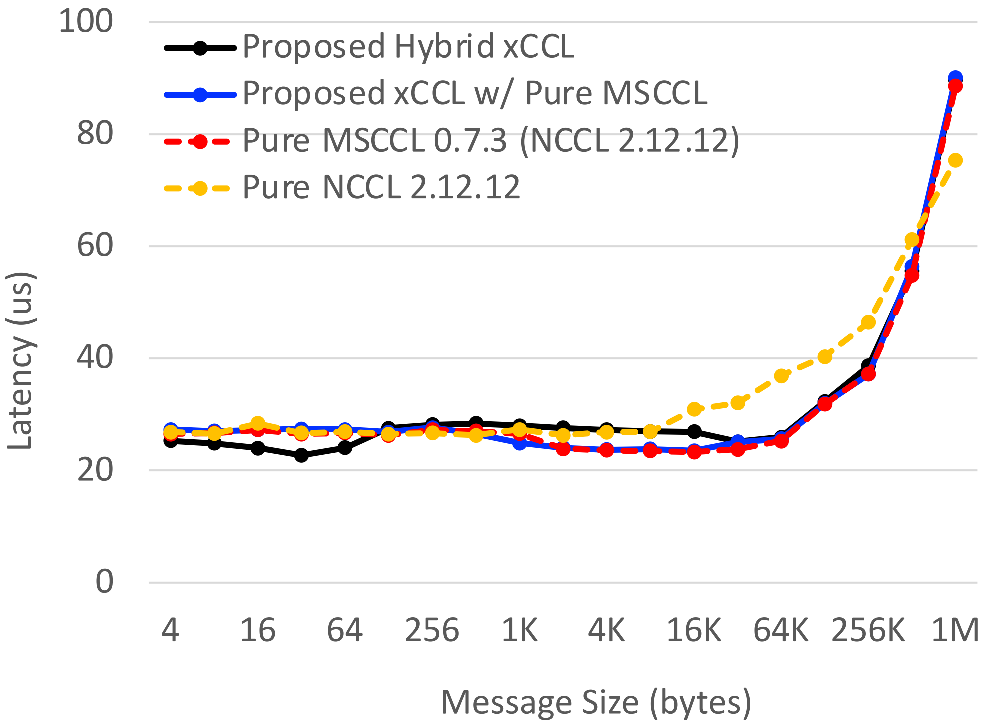
Allreduce w/ HCCL (1 Node, 8 HPUs)



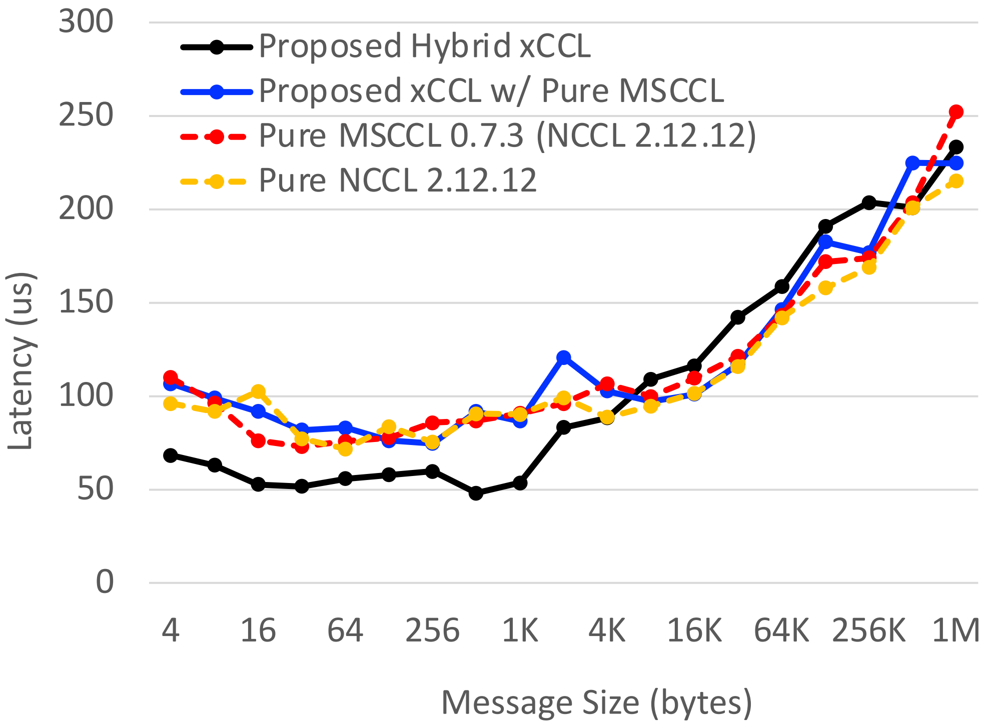
Allreduce w/ HCCL (4 Node, 32 HPUs)

- While no HCCL benchmarks exist for Habana architectures, outcomes suggest our designs' adaptability to new architectures.
- Observe overheads for small messages, especially at the beginning stage, but mostly good on a single node.
- For Allreduce on multiple nodes, we observe degradations shown by a step curve around 64 bytes.

Micro-Benchmark Evaluation: Collective - MSCCL



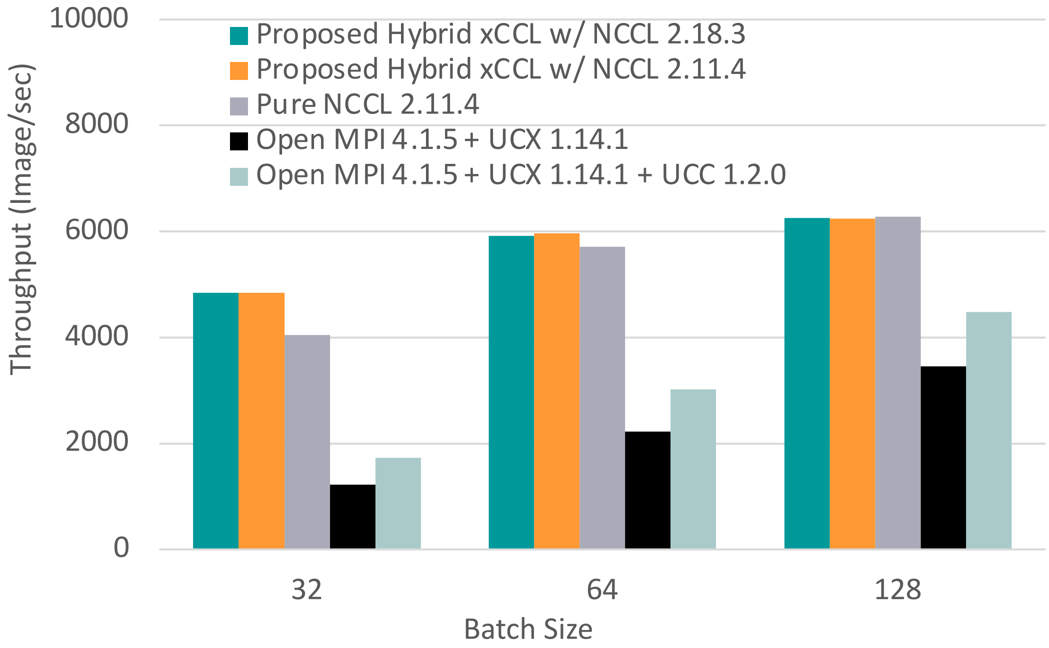
Allreduce w/ MSCCL (1 Node, 8 GPUs)



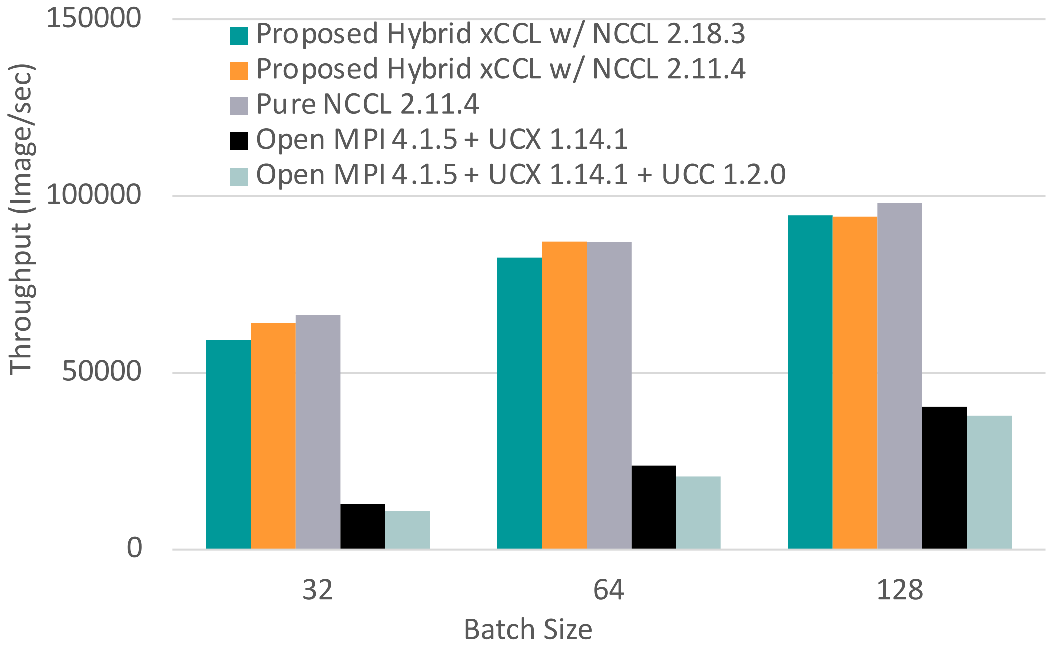
Bcast w/ MSCCL (2 Node, 16 GPUs)

- MSCCL uses NCCL 2.12.12 as the backend, we use pure NCCL 2.12.12 as the baseline.
- MSCCL outperforms NCCL for medium messages (256B - 256KB), and our performance mirrors MSCCL.
- The proposed hybrid designs have better performance for small messages (<64B, <1KB) due to hybrid designs.

Application-Level Evaluation: NCCL



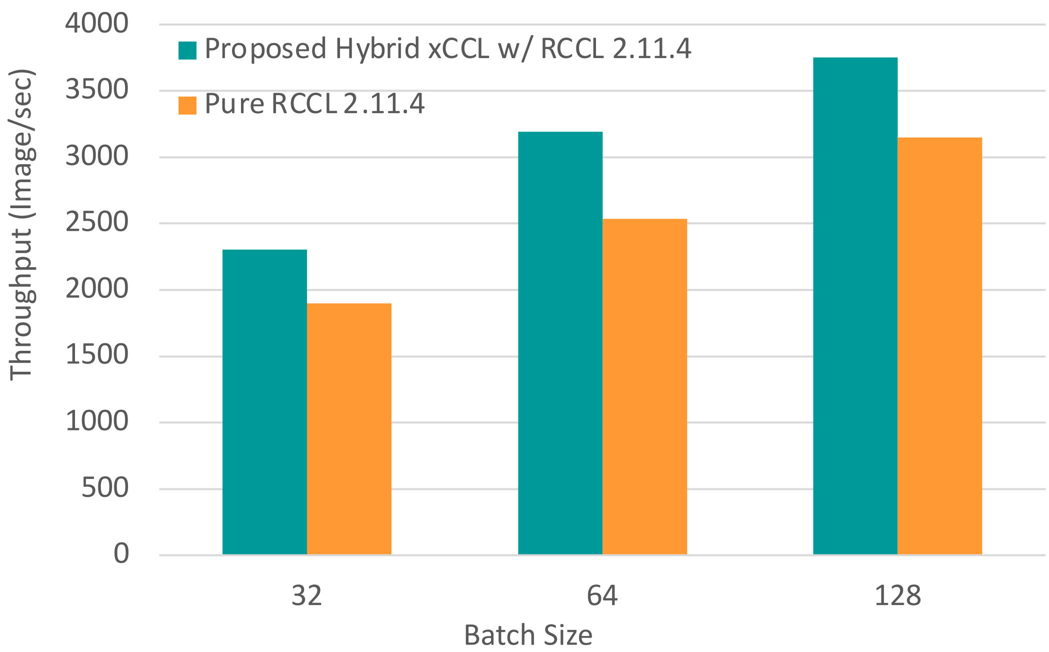
1 Node, 8 GPUs



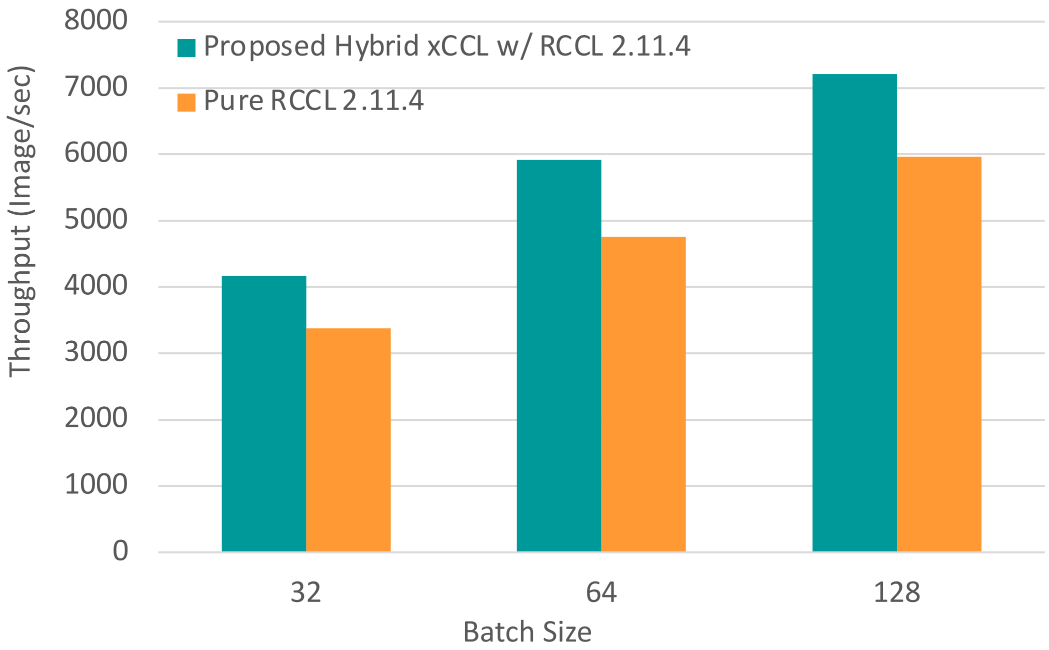
16 Node, 128 GPUs

- Our xCCL designs (with NCCL 2.18.3 or 2.11.4) either match or surpass pure NCCL performance, it achieves 4850 img/sec compared to pure NCCL's 4050 img/sec at batch size 32.
- Traditional MPI runtimes (Open MPI + UCX or advanced designs with UCC) yield 3450 or 4480 img/sec at a batch size of 128, 44% or 28% below our designs.
- On multiple nodes, xCCL's 94600 img/sec throughput is 1.35x and 1.5x higher than Open MPI + UCX and UCC with a batch size of 128.

Application-Level Evaluation: RCCL



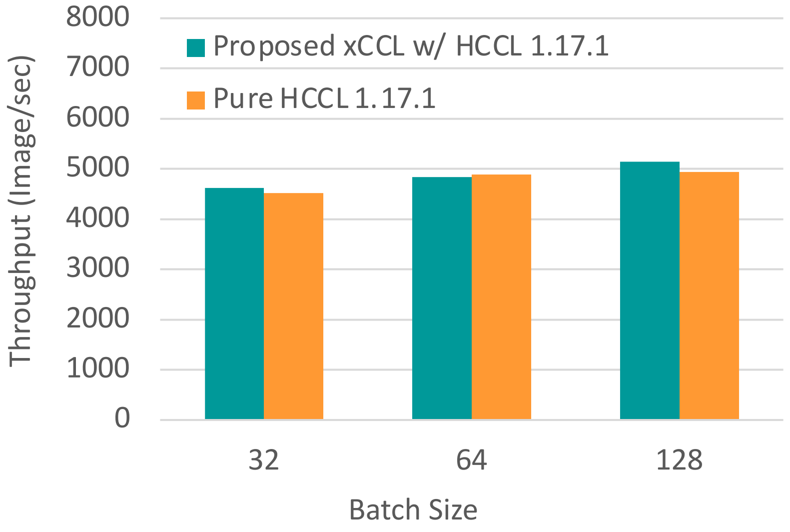
4 Node, 8 GPUs



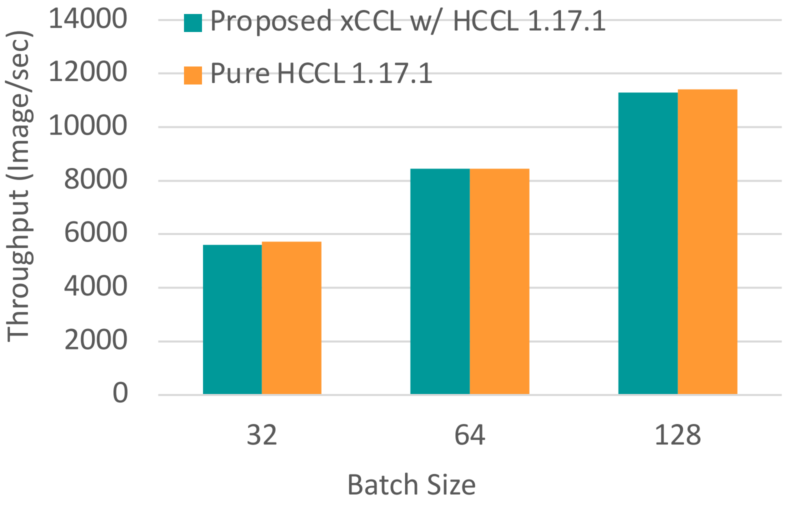
8 Node, 16 GPUs

- Our xCCL designs achieve a throughput of 3192 img/sec with a batch size of 64 on 8 AMD GPUs, which is a 25% improvement over pure RCCL.
- On multiple nodes, it shows a throughput of 7210 img/sec with a batch size of 128 on 16 AMD GPUs, which is a 20% improvement over pure RCCL.

Application-Level Evaluation: HCCL



1 Node, 8 HPUs

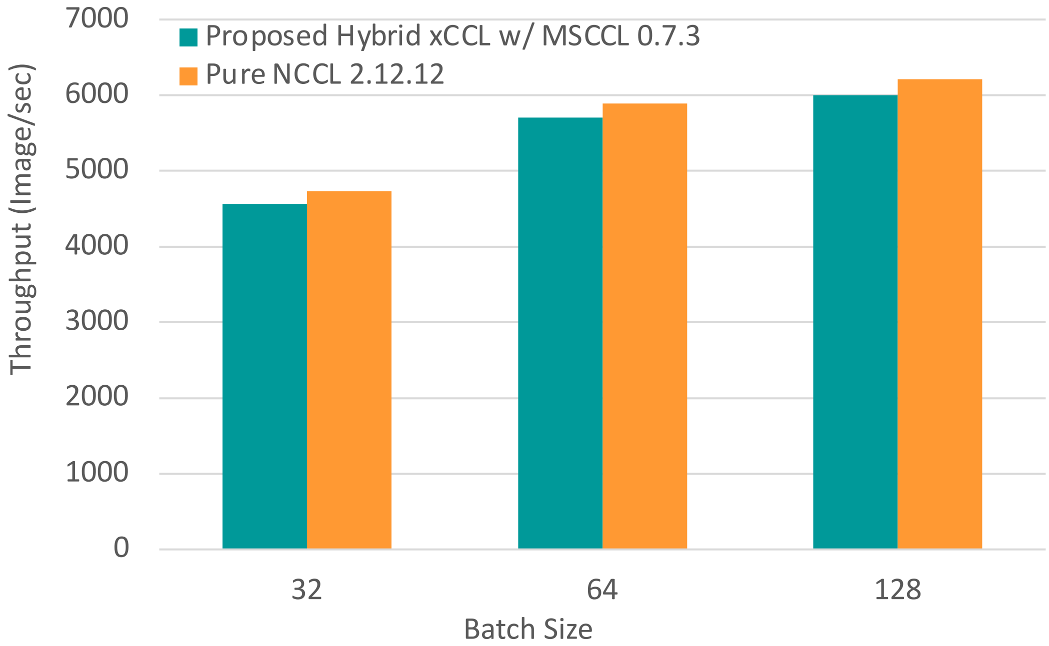


4 Nodes, 32 HPUs

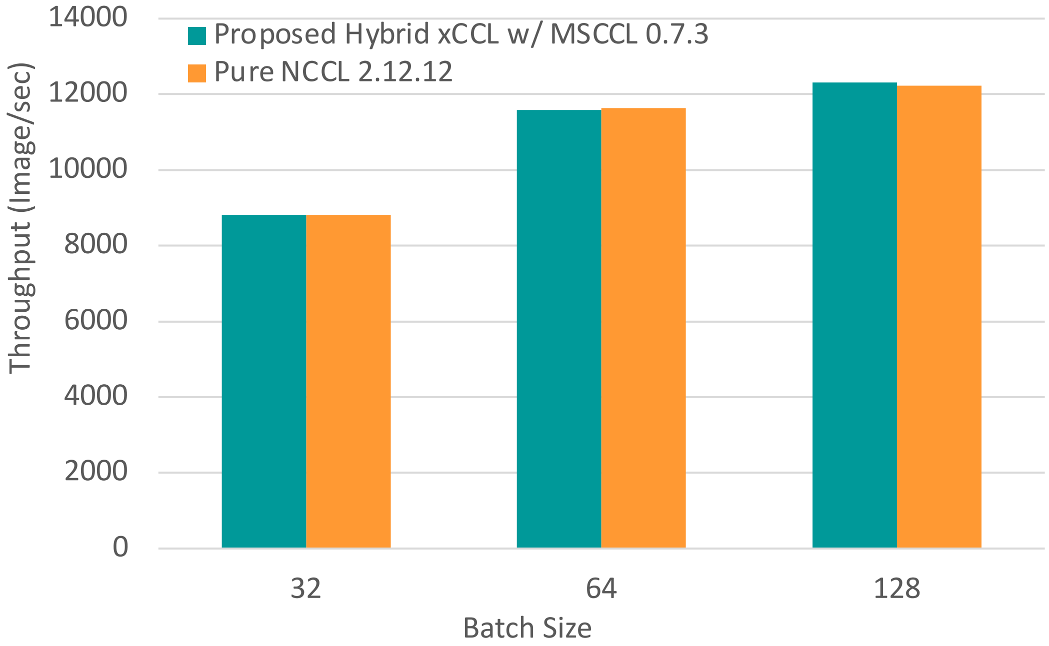
- The container image has already contained prebuilt Habaha TensorFlow and Horovod.
- The computing kernel is replaced by Habana ops through TensorFlow custom ops, the com munication layer in Horovod is implemented by HCCL directly.
- Modify the Horovod communication by replacing all `hcclAllreduce`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

Application-Level Evaluation: MSCCL



1 Node, 8 GPUs



2 Node, 16 GPUs

- Our xCCL's performance with the MSCCL backend, mirroring the NCCL trend, with xCCL achieving 12300 img/sec at batch size 128 on 2 nodes.
- This evaluation proves that our xCCL designs can be easily extended to new collective communication libraries with negligible overheads.

Outline

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

Conclusion and Future Work

- Introduction of xCCL communication runtime for optimal communication-level performance in HPC and Deep Learning on supercomputers.
- Abstraction layer covering NCCL, RCCL, HCCL, and MSCCL APIs, enabling dynamic selection of hardware-specific API calls.
- Performance evaluation on ThetaGPU, MRI, and Voyager clusters with NVIDIA GPUs, AMD GPUs, and Habana HPUs.
- Comprehensive assessment of intra-node and inter-node communication, as well as collective operations across single and multiple GPU nodes using four communication backends.
- Application-level designs achieving substantial throughput gains over UCC and RCCL by 4.6x and 1.25x.
- Pioneering communication-level performance evaluation for upcoming Habana Gaudi Processors.
- Future work aims to extend support to additional hardware such as Intel GPUs or FPGAs and new vendor-specific libraries like oneCCL.

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/