MPI in Distributed Training

Matija Jakovac & Emanuel Pichler

Outline

Message Passing vs. Shared Memory

SHARED MEMORY

- One address space
  - Same RAM
- Synchronization with locks
- Cache line latency
- Scales only inside a single socket

MESSAGE PASSING

- Each process has its private RAM
- Exchange data via Send/Recv calls
- Network latency
- Scales across clusters

What is Message Passing Interface (MPI)?

- Open standard
- Language-independent
- Defines data exchange between processes

Category	Calls
Point-to-Point	MPI_Send, MPI_Recv
Collectives	MPI_Bcast, MPI_Allreduce, MPI_Scatter/Gather
Process Management	MPI_Init, MPI_Comm_rank, MPI_Comm_size, MPI_Finalize

Usage of MPI

- Multiple implementations of the standard
  - **OpenMPI**
  - **MPICH**
  - **Intel MPI**, **IBM Spectrum MPI**

Classic HPC workloads

- Weather & climate models
- Computational fluid dynamics

AI

- Deep-learning training with Horovod
- Data analytics with Dask-MPI

Concrete Installations

- Summit (US, 200 petaFLOPS) uses OpenMPI
- LUMI (FI, 531 petaFLOPS) uses Cray MPI

MPI vs. Other Collective-Communication Options

MPI
CPU & GPU clusters

Richest feature set

Gloo (Meta)
CPU clusters

Built into PyTorch Distributed

NCCL (NVIDIA)
GPU clusters

oneCCL (Intel)
CPU clusters

Runs fastest on Xeon clusters

Project Goals

Measure speed-up

Analyze communication cost

Validate model quality

Produce actionable visuals

Project Setup

Hardware: Lenovo ThinkPad P1 with an 8-core AMD Ryzen 5700

OS: Ubuntu 22.04

Software: OpenMPI (mpi4py), PyTorch, PyTorch Distributed (with Gloo backend)

Limitations: only one CPU is available, with 1 core per process. This does not fully include the network latency in a distributed computing cluster.

Dataset & Model Suite

- Dataset — MNIST
  - 60 000 training + 10 000 test images
  - 28 × 28 grayscale digits (0 – 9)
  - Fits easily in RAM ⇒ lets us focus on CPU scaling rather than I/O.
- Learning in 5 epochs with a batch size of 128.

Model	Parameters
MLP	235k
CNN	422k
RNN	82k

Sequential Baselines

	SEQ-1	SEQ-4	SEQ-8
Cores / Threads	1	4	8
Epoch Time - MLP	15 sec	14 sec	13 sec
Images / s - MLP	3800	4290	4500
Speed-up - MLP	1x	1.07x	1.15x
Epoch Time – CNN	65 sec	37 sec	30 sec
Images / s – CNN	930	1580	1970
Speed-up - CNN	1x	1.76x	2.2x
Epoch Time – RNN	33 sec	22 sec	20 sec
Images / s - RNN	1840	2711	2900
Speed-up - RNN	1x	1.5x	1.65x

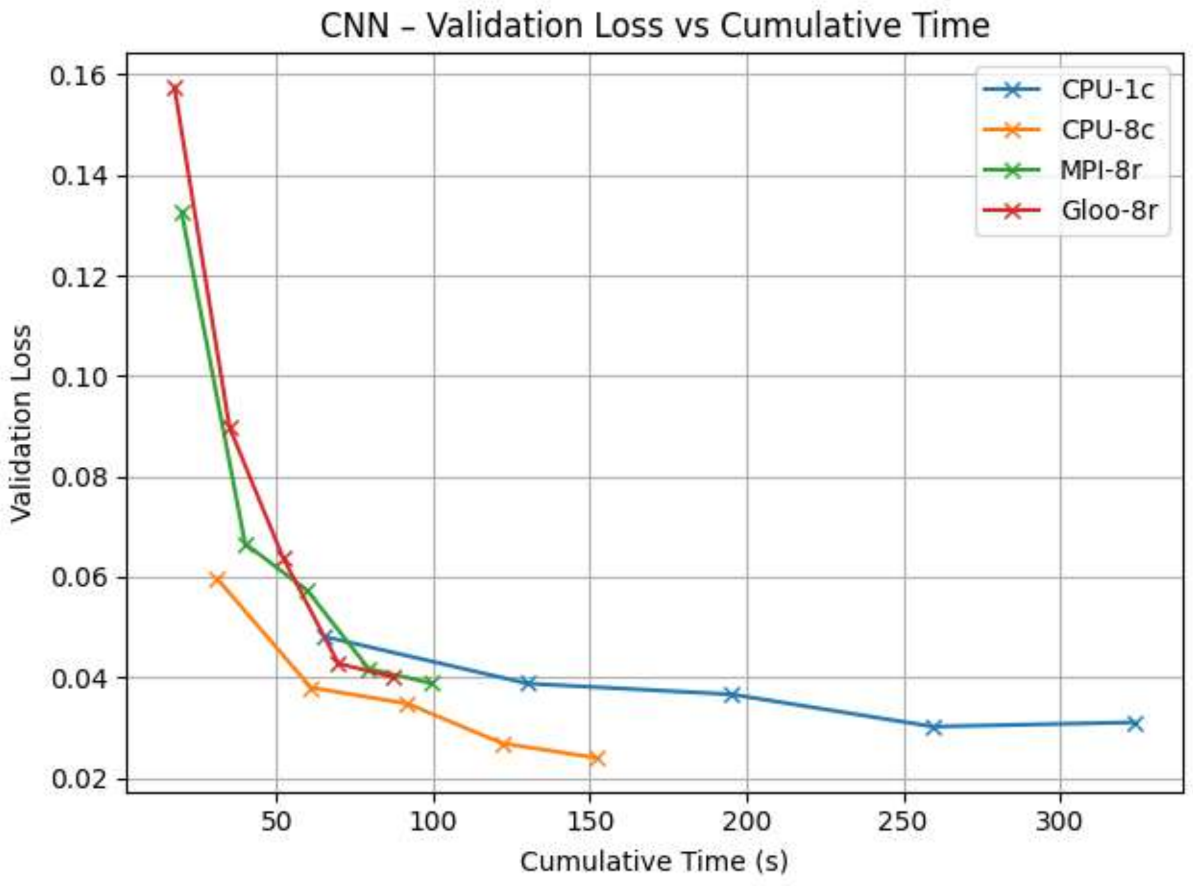
MPI Training Flow

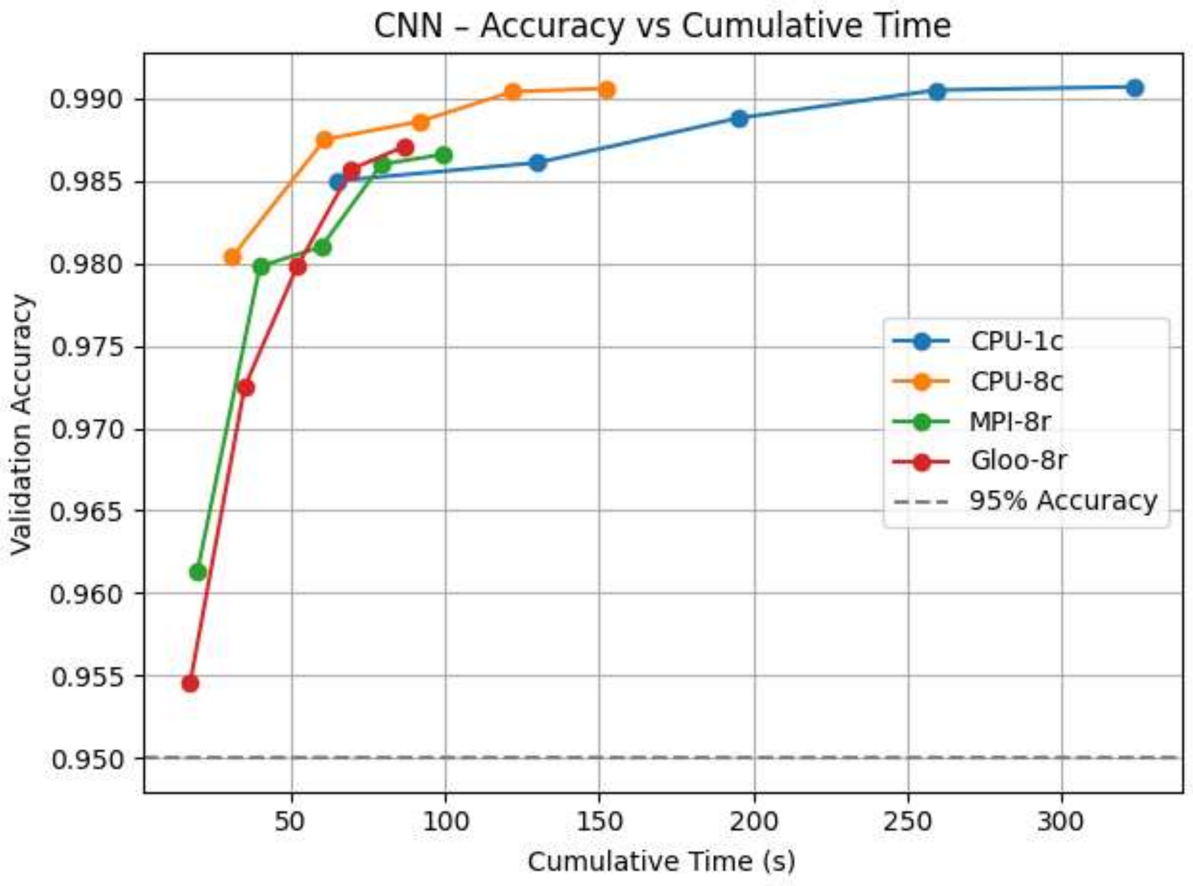
Phase	What happens on every rank	Key MPI call(s)
0 · Init	Discover rank ID and world-size.	MPI_Init, MPI_Comm_rank, MPI_Comm_size
1 · Broadcast Weights	Rank 0 sends model parameters so all ranks start identical.	MPI_Bcast
2 · Dataset Shard	Each rank loads only its slice.	No call needed, but slicing uses rank and world from step 0
3 · Forward Pass	Local computation on its mini-batch.	—
4 · Backward Pass	Compute local gradients g_r.	—
5 · Gradient Averaging	Sum grads across ranks, then divide.	MPI_Allreduce
6 · Parameter Update	Use the averaged grads → weights stay in sync.	—
7 · Epoch Barrier	Ensure all ranks finish epoch before timing / evaluation.	MPI_Barrier
8 · Gather Metrics	Rank-0 collects loss/accuracy from everyone for logging.	MPI_Gather
9 · Finalize	Clean MPI shutdown.	MPI_Finalize

PyTorch Distributed (Gloo) Training Flow

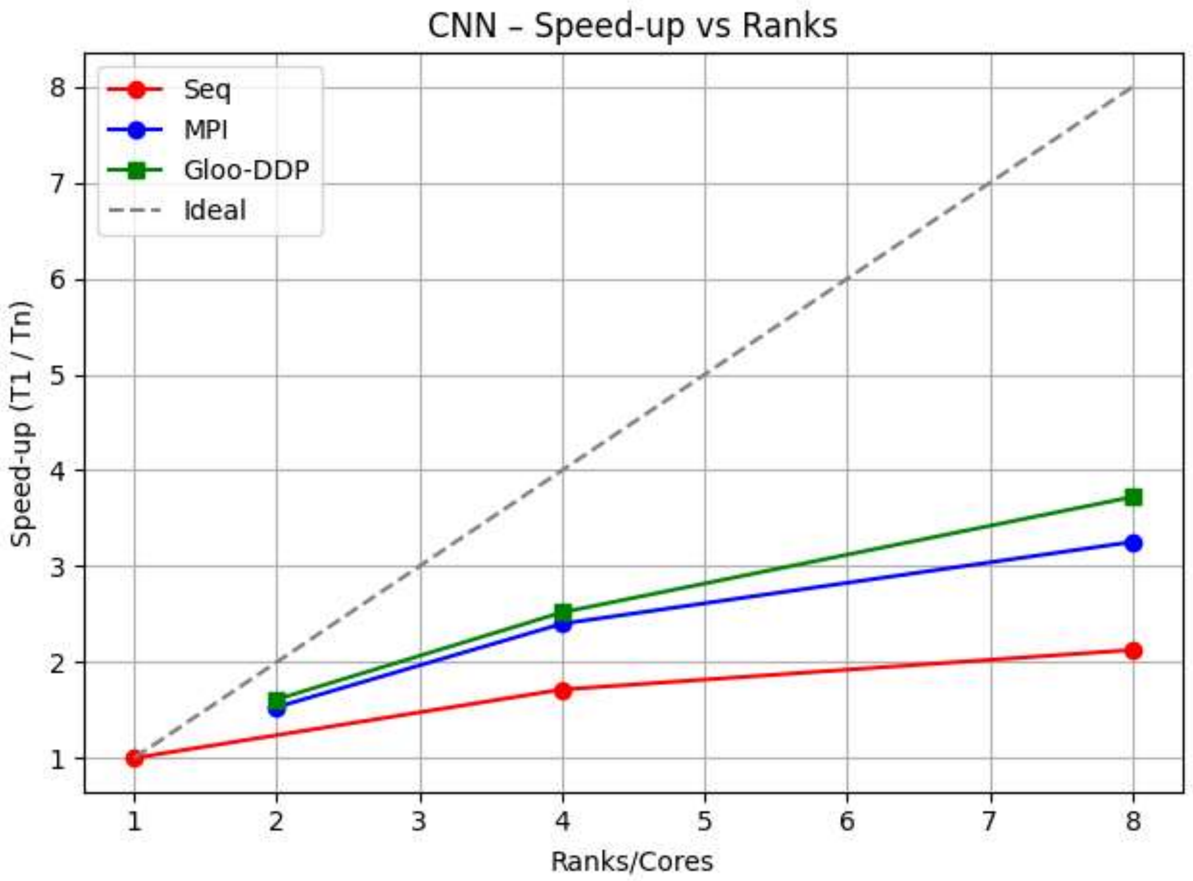
Phase	What happens on every rank	Key DDP / Gloo call(s)
0 · Init	Environment variables set by torchrun give RANK and WORLD_SIZE.	dist.init_process_group(backend="gloo")
1 · Wrap Model	Model copied to CPU.	model = DistributedDataParallel(model)
2 · Dataset Shard	PyTorch sampler gives each rank non-overlapping batches.	DistributedSampler(dataset, num_replicas=WORLD_SIZE, rank=RANK)
3 · Forward Pass	Local computation on its mini-batch.	—
4 · Backward Pass	Local gradients are computed.	—
5 · Gradient Synchronisation	For every parameter DDP launches an asynchronous Gloo Allreduce and averages grads across all ranks.	(internal) allreduce_coalesced via Gloo
6 · Parameter Update	Optimiser uses the averaged grads, so weights stay identical on all ranks.	—
7 · Implicit Barrier	DDP waits for all Allreduces to finish before exiting backward().	—
8 · Metrics Collection	Rank 0 computes/prints loss & accuracy; other ranks optionally skip eval.	—
9 · Shutdown	Clean up distributed resources.	dist.destroy_process_group()

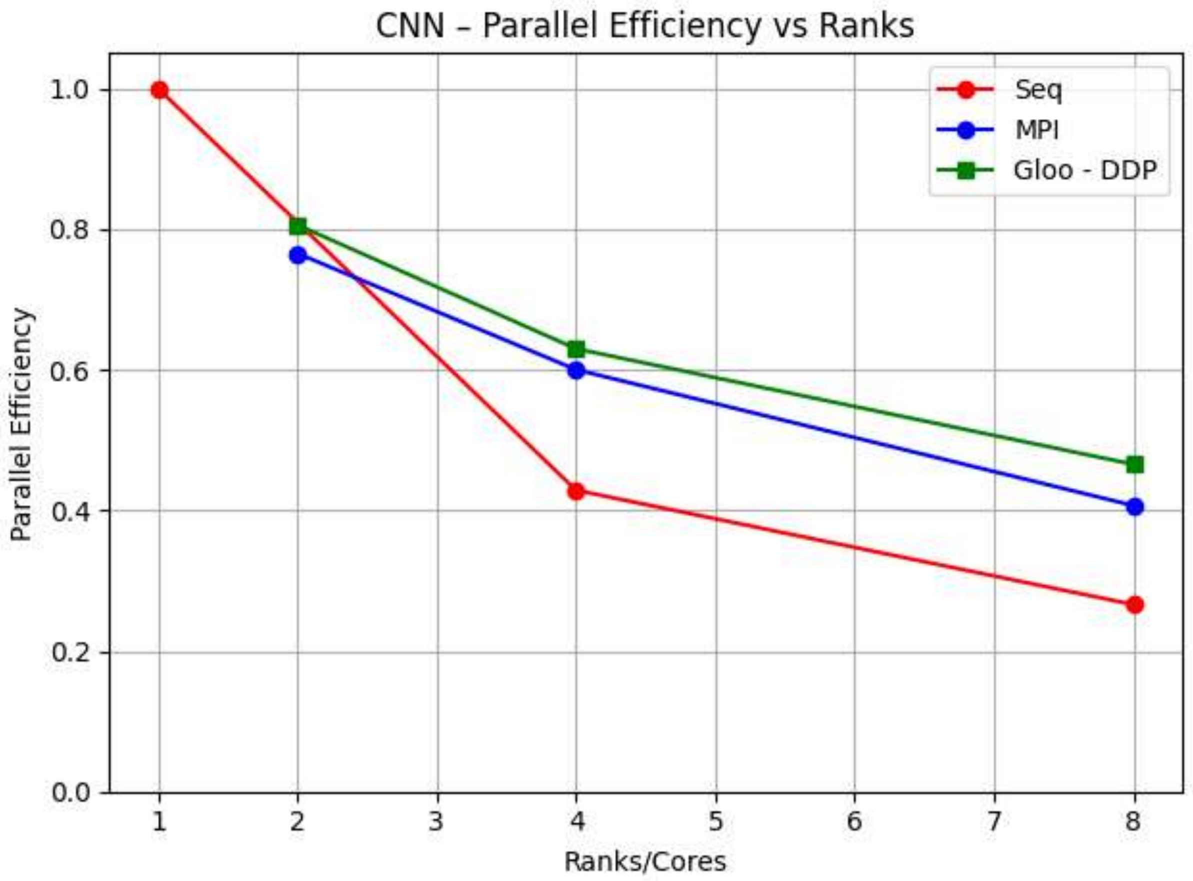
Results – Wall-clock Learning Curves

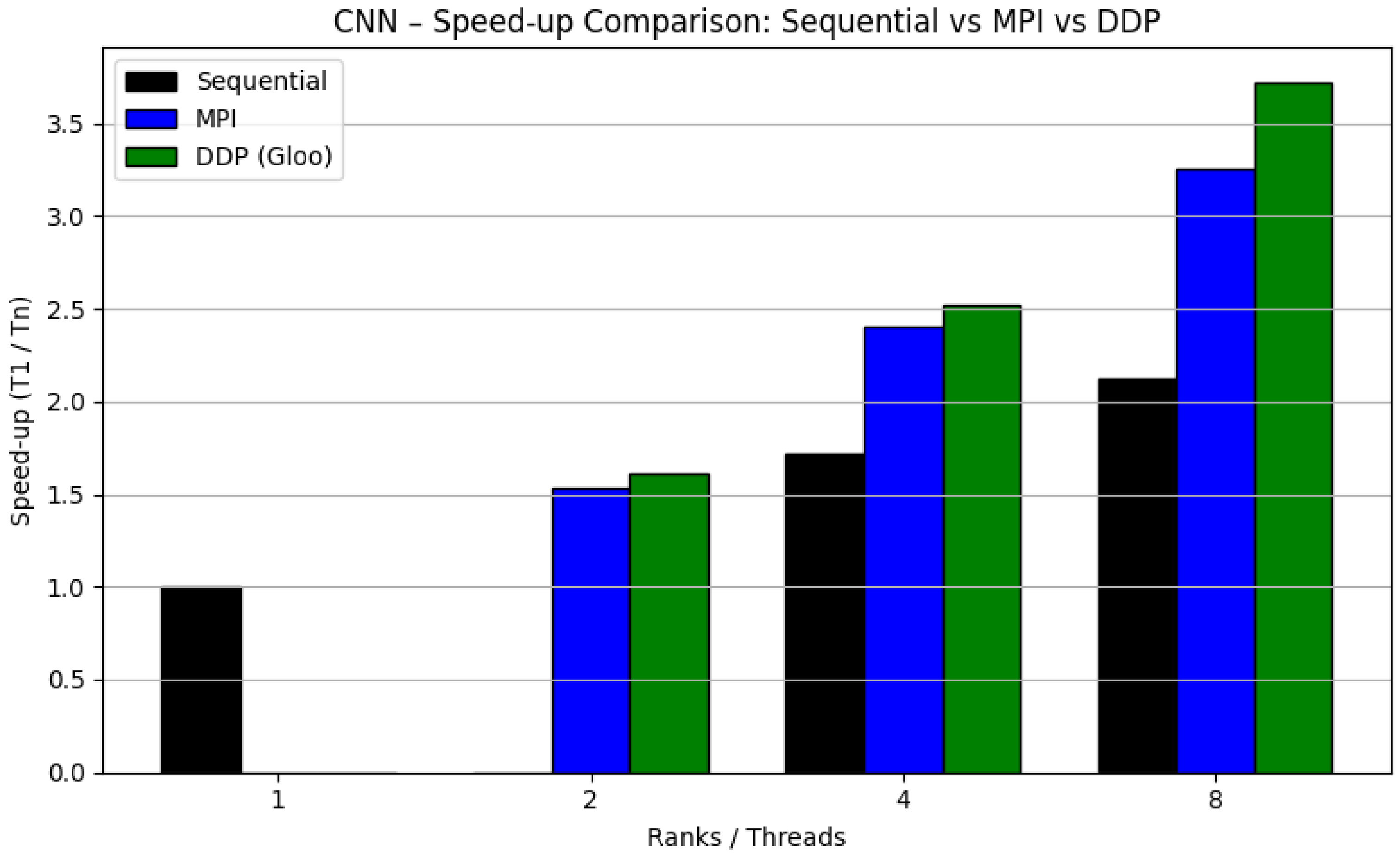




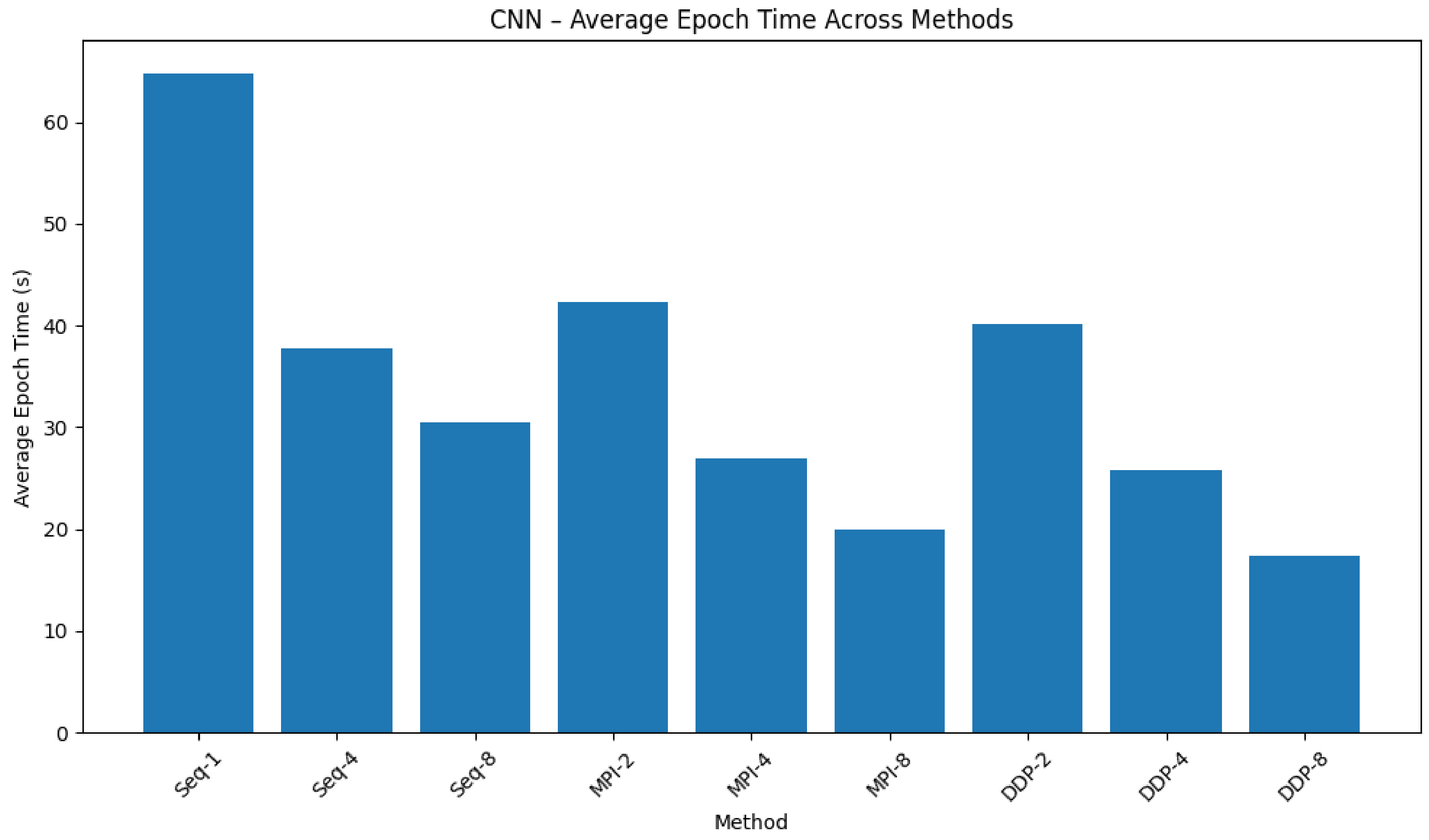
Results – Speedup & Efficiency





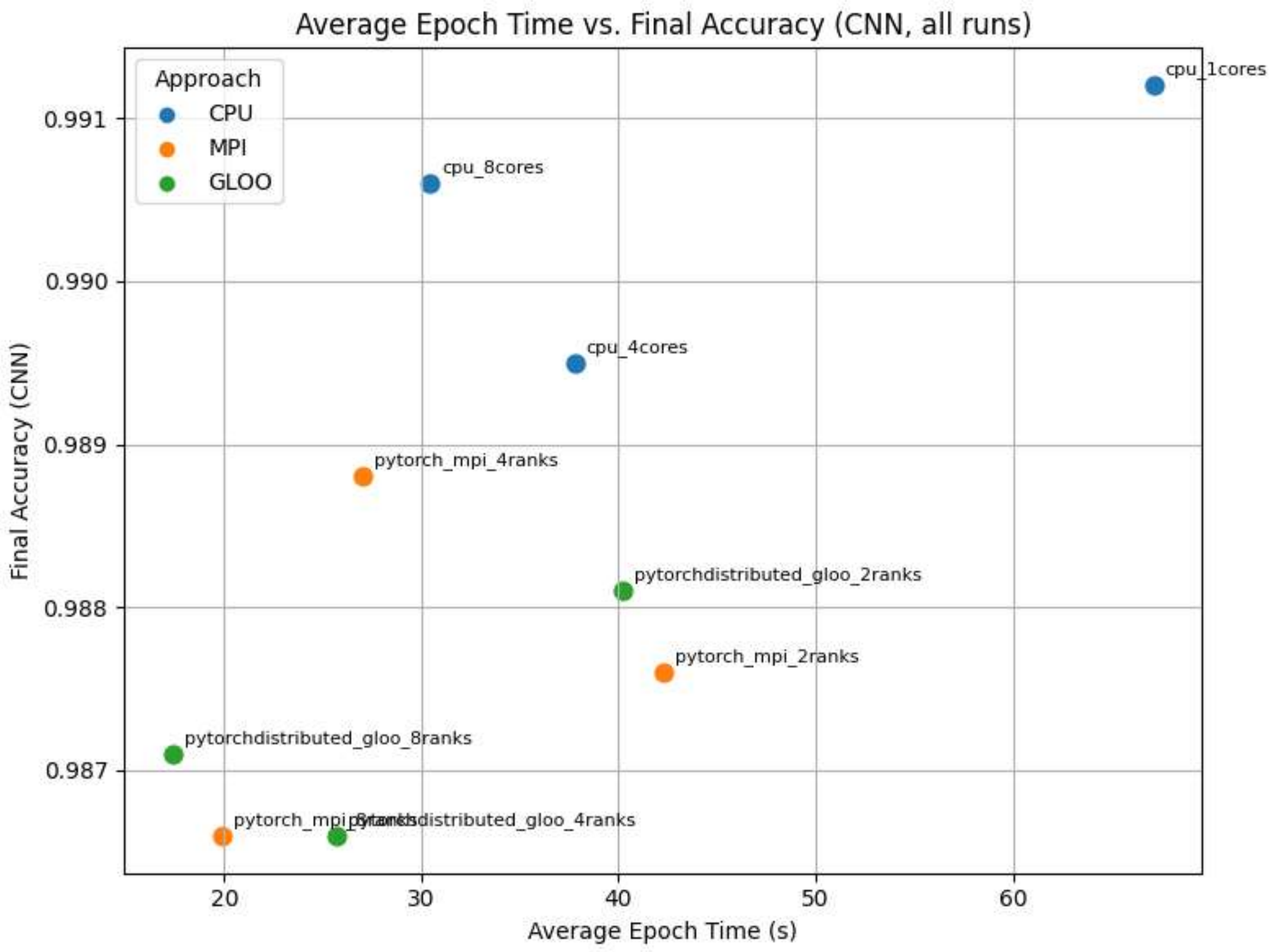


Results – Speed-up cont.

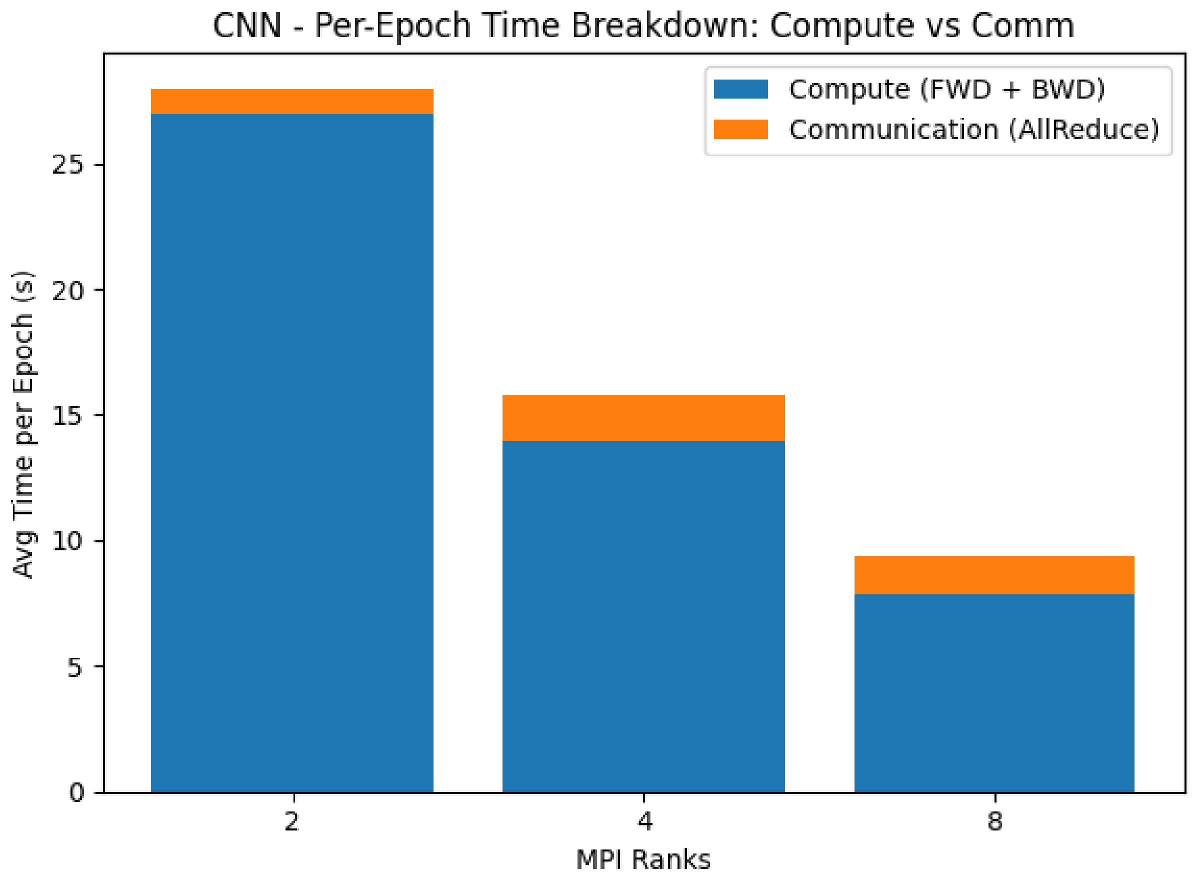


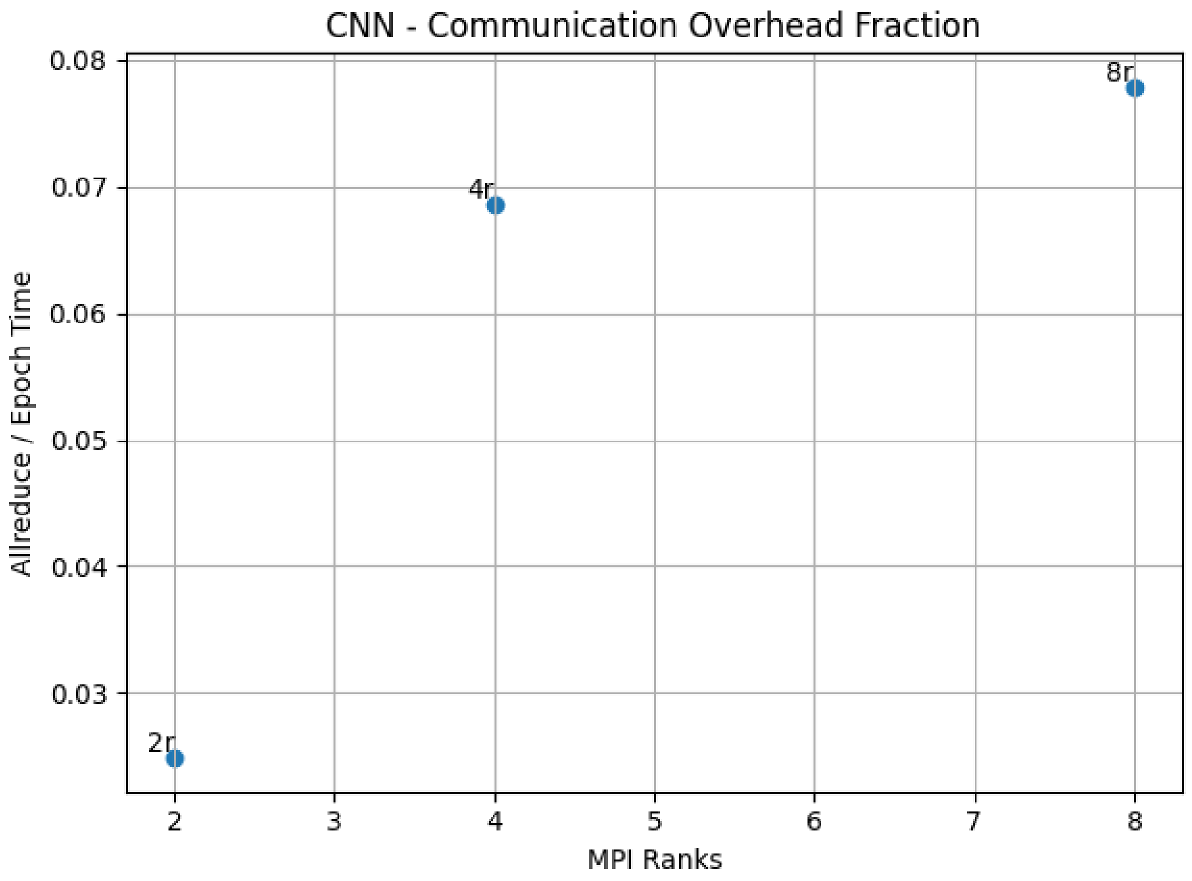
Results – Epoch Time

Results – Epoch Time vs. Final Accuracy



Results – Communication Overhead





Key Findings

- Accuracy decreases with a higher degree of distribution.
  - The sequential approach on 1 thread reaches 99.1%
  - The best distributed accuracy was achieved with MPI (4-rank), 98.88%
- MPI and Gloo outperform the sequential approach
  - even when running the sequential approach with 8 threads on the local machine.
  - Gloo outperformed MPI by a very small magnitude only.

Lessons Learned

- Communication Wall shows up fast.
  - With increasing ranks, the Allreduce time gets large relative to the mini-batch time.
  - This leads to efficiency dropping.
- Oversubscribing the CPU
  - The increase in performance stagnates if too many ranks and threads are used.
  - This is due to the increased overhead and oversubscribing the limited hardware available.
- The accuracy decreases slightly with ranks increasing

Future Work

- Scale Up the Hardware
  - Multi-node test on university HPC cluster to measure real network latency.
  - Exchange DDP backend to NCCL and compare CPU vs. GPU scaling for a GPU variant.
- Alternative Communication Libraries
  - Horovod (MPI & Gloo)

Conclusion

- Introduction to inter-process communication techniques in HPC
- Compared and analyzed distributed AI training techniques
  - Sequential, Message Passing Interface (MPI), Gloo
  - Limitations due to single 8-core CPU only
- Distributed Learning improved the time per epoch by up to 3.9x
  - Best timing results with 8-rank Gloo
    - 17 sec per epoch with Gloo
    - 19 sec per epoch with MPI
    - 67 sec per epoch with 1-core CPU training, 30 sec per epoch with 8-core CPU training
  - Best validation results with non-distributed CPU training
    - 99.1 % with non-distributed CPU training
    - 98.7 % with MPI and Gloo

Questions