Evaluating All-Reduce Operations in DDP Training with FABRIC

Framework for Architecture, Backend, and Reduction Interchange Comparison

Matilde Restelli [1,2], Ke Fan [3], Aadirupa Saha [1], John K. Holmen [2], and Sidharth Kumar [1]

[1] University of Illinois Chicago, [2] Oak Ridge National Laboratory, [3] Temple University

Motivation and Challenges

Goal: A systematic framework to study how communication strategies and backends affect Distributed Data Parallel (**DDP**) training.

Motivation: Enable investigation of communication strategies inside real PyTorch DDP training, where all-reduce algorithms can be easily integrated and evaluated.

Challenges:

- Communication algorithms are hidden inside backends.
- Differences across backends, systems, and GPU vendors further complicate fair and portable evaluation.

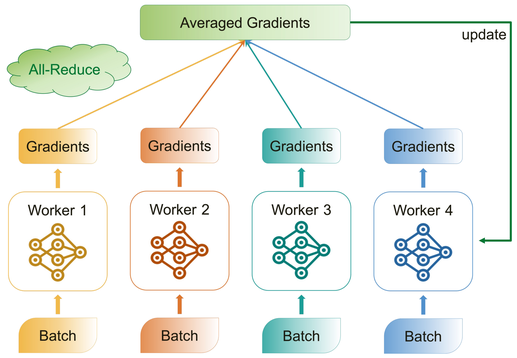


Figure 1. All-reduce gradient synchronization across GPUs in DDP training.

Systems

- Developed and tested on two contrasting HPC platforms:
  - **Frontier** (OLCF): AMD system, 9,856 compute nodes, each with one AMD EPYC CPU and four AMD Instinct MI250X GPUs.
  - **Polaris** (ALCF): NVIDIA system, 560 compute nodes, each with one AMD EPYC CPU and four NVIDIA A100 GPUs.
- PyTorch built from source with fixed GPU-aware MPI detection for **Cray MPICH** to support the MPI backend.
- To exploit **ZFP compression kernels**, we integrated ZFP with CUDA and HIP support and added stream-aware compression/decompression launches.



Figure 2. *Left:* Polaris (ALCF). *Right:* Frontier (OLCF).

Acknowledgments:

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Advanced Scientific Computing Research programs in the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725. This research used resources of the Argonne Leadership Computing Facility, which is a U.S. Department of Energy Office of Science User Facility operated under contract DE-AC02-06CH11357.

Framework

- Exposes the gradient synchronization step using **communication hooks**.
- DDP primitives invoke the selected backend: **MPI**, **NCCL**, or **RCCL**.
- Experiment setup through a Python interface or a batch/job script.
- Enables interchangeable selection of the **ML frontend** (model, dataset, optimizer, …), **backend**, and **communication algorithm** within the same training code.
- New communication algorithms can be added by implementing and registering a communication hook.

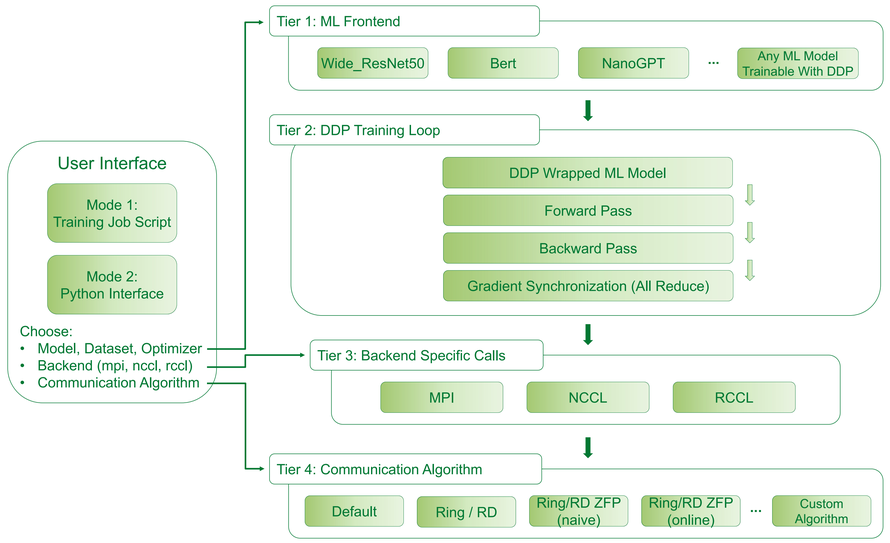


Figure 3. Four-tier FABRIC architecture, from ML frontend to communication algorithm.

Experiments

- **Configuration:** DDP hyperparameters were tuned once per model and then fixed across backends, algorithms, and both systems.
- **Scaling: Strong scaling** fixed the global batch size. **Weak scaling** fixed the per-GPU batch and grew LR with GPU count.
- **Metrics:** Primarily epoch time and iteration time. Framework instrumentation logs **hook work time** and **communication tail time** to better isolate communication overhead, as shown here:

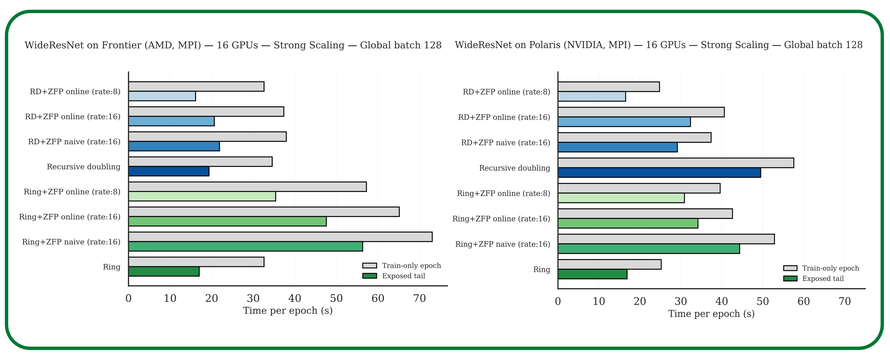


Figure 4. Communication tail time, 16 GPUs, strong scaling, global batch 128. *Left:* Frontier. *Right:* Polaris.

Results

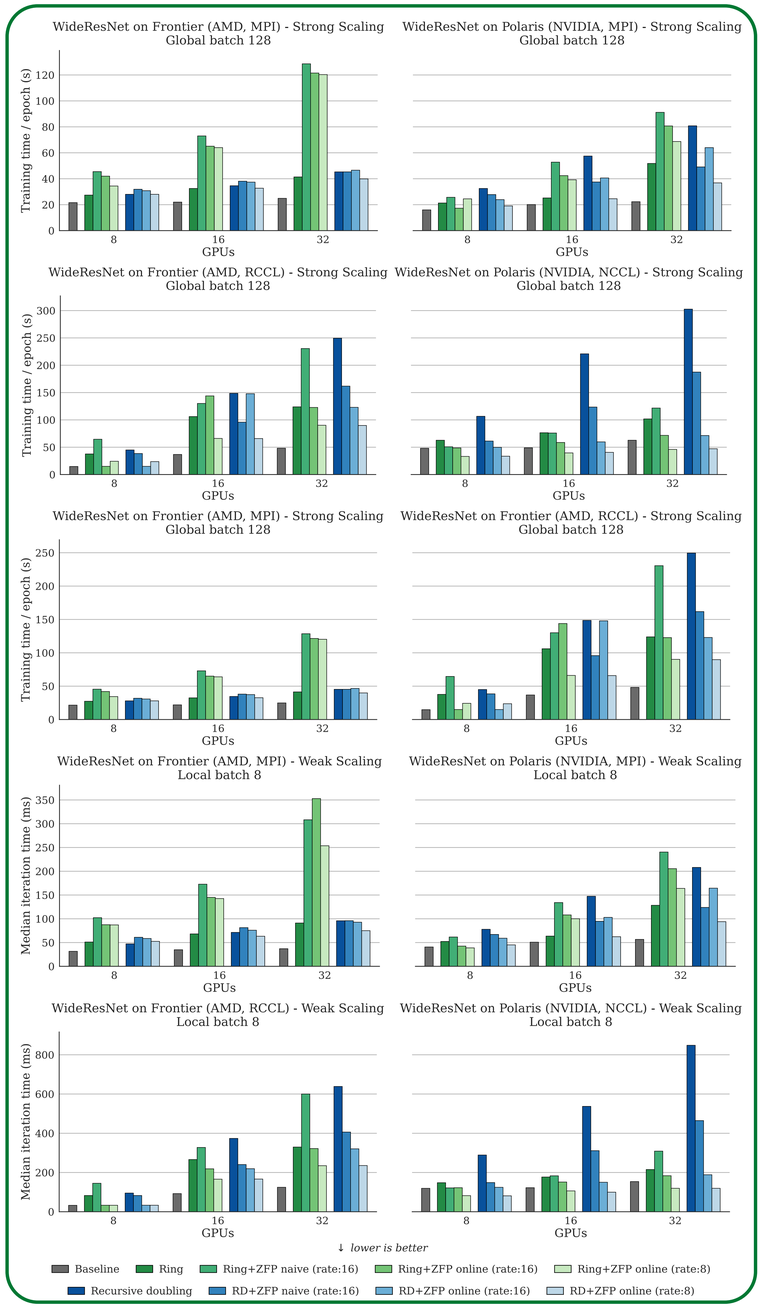


Figure 5. WideResNet training time across scaling type, system, and backend.

Conclusions and Future Work

- Backend and algorithm choices strongly affect DDP performance across systems, models, and scaling regimes.
- **Deeper analysis** of communication can reveal bottlenecks and guide **custom communication algorithm** design.
- Future evaluation: **larger models**, compute-intensive workloads, and **hierarchical communication** for HPC.