MCR-DL: Mix-and-Match Communication Runtime for Deep Learning

Quentin G. Anthony, Ammar Awan, Jeff Rasley, Yuxiong He, Aamir Shafi, Mustafa Abduljabbar, Hari Subramoni and Dhabaleswar K. Panda
{anthony.301, shafi.16, abduljabbar.1, subramoni.1, panda.2}@osu.edu, {ammar.awan, jeff.rasley, yuxhe}@microsoft.com

**DESIGN – MCR-DL**

We define a Mix-and-Match Communication Runtime for Deep Learning (MCR-DL)

• Unified interface for all communication calls based on PyTorch distributed module's API
• Backend-aware communication synchronization to avoid deadlocks
  • `MPI_Wait()` for MPI, CUDA events for NCCL/SCCL
• Easy to add communication features like compression, tensor fusion, and logging.

**PERFORMANCE – Applications**

Performance of DL applications using proposed designs on the Lassen@LLNL (V100 GPUs) and ThetaGPU@ALCF (A100 GPUs) systems

• Baseline: Single communication library for all operations
• MCR-DL: Coarse library mixing (NCCL AllReduce, MVAPICH2-GDR AlltoAll)
• MCR-DL-T: “Tuned” library mixing using a static table mapping message sizes to communication backend (see below table for details)

MCR-DL supports creating tuning tables (see right)

• Define table before DL applications are run
• Map messages to libraries based on message size
• One table per #GPUs and communication operation

MCR-DL’s reduced Python logic for communication operations improves small-message overhead (see below)

**CONTRIBUTIONS**

• Comprehensive profiling and evaluation of modern deep learning applications
• Introduce MCR-DL, a unified communication interface that supports mixing libraries so that the best library is used for a given communication operation
• Evaluate proposed designs compared to existing libraries using DeepSpeed Mixture-of-Experts (DS-MoE) on up to 256 V100 GPUs, and Deep Learning Recommendation Models (DLRM) on up to 32 A100 GPUs

**MOTIVATION**

• Deep Learning applications require significantly more communication than previous data-parallel applications
• Deep Learning workloads are no longer purely using AllReduce and Broadcast communication operations
  • Tensor Parallelism: AllReduce/AllGather
  • Pipeline Parallelism: Send/Recv
  • Data Parallelism: AllReduce/Broadcast
• Given that we need a mixture of communication operations, which communication library should we use?
  • Best AlltoAll is MVAPICH2-GDR, and best iAllReduce is NCCL

**PERFORMANCE – Secondary Results**

• MCR-DL reduces the proportion of time spent in tensor communication (see above)

• MCR-DL’s reduced Python logic for communication operations improves small-message overhead (see below)

**CHALLENGES**

How to use the best communication library for a given operation?

• How do we avoid deadlocks when mixing communication backends?
• Can we choose the best library dynamically based on the message size?
• How do we define a unified interface for all communication calls?