ECS 289D Project Proposal : Tutorial on RDMA programming in the GPU era

Kaiyue Li, Yanfeng Ma, Qianqian Tan, Zhuoli Huang, Yiqiao Lin

November 5, 2025

1 Problem Selection

As GPUs become the main computing units in modern high-performance computing and AI workloads, large-scale data transfer between nodes has become a major bottleneck for overall system performance. RDMA is widely adopted for its low latency and low CPU overhead, making it a key communication technology in current datacenter and AI cluster designs.

Recent works, such as UCCL, focus on improving RDMA performance and scalability for distributed machine learning systems. However, there is still a lack of clear, hands-on tutorials that demonstrate how RDMA can be implemented and optimized in GPU-based environments. This project aims to build a tutorial that explains and demonstrates RDMA programming in the GPU era, helping users understand the design, challenges, and optimization techniques for GPU-direct communication.

2 Motivation

While the idea of RDMA and its performance are well-known for all, the actual RDMA programming that utilizes libibverbs to achieve more complex communications and data transfer. Extending RDMA to GPU-optimized communication introduces additional challenges such as memory registration and synchronization between devices. To address this, we aim to build a clear and practical tutorial that helps users understand both CPU and GPU data-transfer workflows.

3 Related works

3.1 RDMA tutorial

Although many open-source examples exist for basic RDMA programming, few resources provide clear, structured explanations of the underlying concepts and code flow. To address this gap, our tutorial will follow and extend the structure presented in Introduction to Programming Infiniband RDMA , which offers a practical walkthrough of low-level RDMA programming with InfiniBand.

We will use this reference as a foundation to explain each component of the RDMA workflow, including device initialization, queue pair creation, memory registration, and connection setup. The tutorial will emphasize how these primitives interact through the verbs API and provide annotated code examples to help readers understand key operations such as RDMA Read/Write and Send/Recv. This serves as the basis for our GPU-optimized RDMA discussion in later sections.

3.2 UCCL

Scalable software transport layer called UCCL (Ultra-CCL) is designed to address the growing demand for network transmission technologies from rapidly evolving machine learning (ML) workloads and the difficulty in evolving the host network transmission on RDMA network cards. The core idea of UCCL is to decouple the data path and control path of the existing RDMA network card and efficiently run the control path transmission logic on the host CPU. This software scalability brings transmission innovations that hardware cannot achieve, such as multi-path transmission for resolving flow conflicts, thereby increasing the communication performance of ML collective operations by 4.5 times compared to existing RDMA network cards.

UCCL provides a good example of RDMA implementation, especially in its optimization for GPU memory. We would use that as our example when talking about optimization in our tutorial.

4 Plan and methodologies

We will implement our tutorial in two phases:

Phase 1: CPU-side RDMA Fundamentals

In this phase, we would provide the general tutorial for RDMA

- Introduce RDMA verbs programming model and libibverbs API.
- Implement basic examples: RDMA Write, RDMA Read, Send/Recv.

Phase 2: GPU-optimized RDMA

In this phase, we would discuss about optimizations for GPU with reference to the implementation of RDMA in uccl library.

- Discuss optimizations that could apply to RDMA for GPU.
- Provide example with related code in uccl library.

5 Resource needed

Access to two virtual machines that have NICs that support RDMA in order to test our example and code.