ModSim 2026 Workshop

AMD
together we advance_

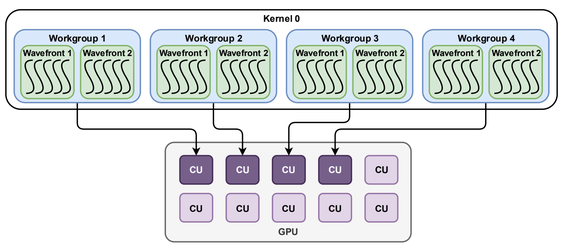
ASTRA-sim 3.0: Taking Distributed Machine Learning Simulation to the Next Level via Fine-Grained GPU Modeling

William Won, Tuan Ta, Moumita Dey, Ruchi Shah, Conor Green, and Bradford M. Beckmann
AMD Research and Advanced Development

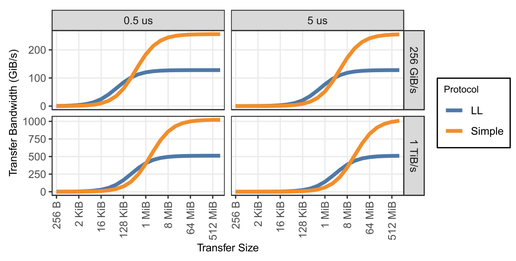
Primary Research Area: Recent Advances in ModSim Implementation

Introduction and Motivation

- Distributed machine learning (ML) modeling needs is in **inference**
  - **Small-sized** collective communications → **latency-sensitive**
- **Faithful latency modeling** is a must
  - **Data Path**: Cache-line-sized `load` and `store` transactions
  - **Control Path**: barriers, semaphores, resource constraints, etc.
- Trade-off between **fidelity** and **speed**
  - ASTRA-sim: fast, but coarse-grained: cannot accurately model latency
  - GPU simulators: high-fidelity, but slow and not flexible



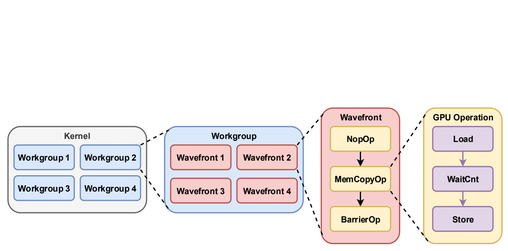
Compute unit (CU) runs **workgroup (WG)** = multiple **`load/store`** wavefronts (WFs)



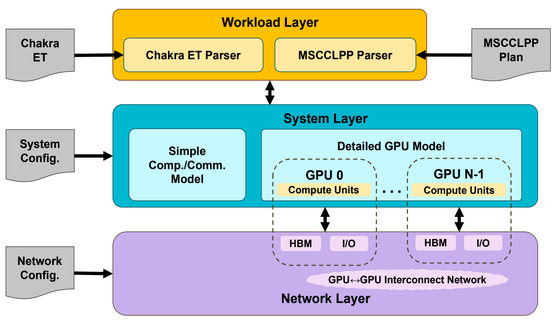
Small latency changes result in different system implications

ASTRA-sim 3.0

- **Strike the right balance** between fidelity and speed with flexibility
  - Model **cache-line-sized** `load/store` transactions, but not full GPU assembly
  - New detailed **GPU Model** that simulates execution at CU-level granularity
  - Support **arbitrary collective algorithm** (we parse MSCCL++)



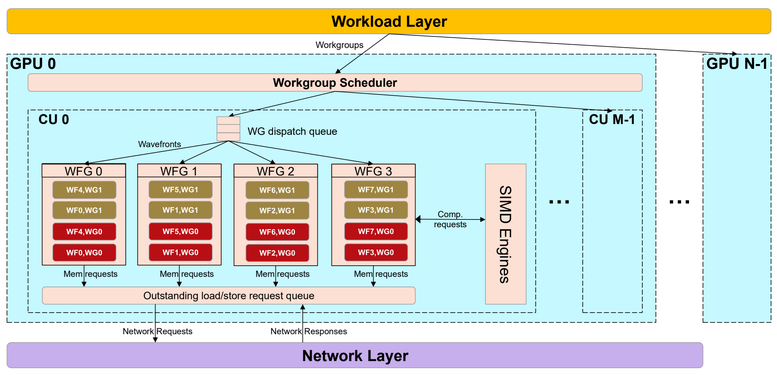
Fine-grained GPU simulation model



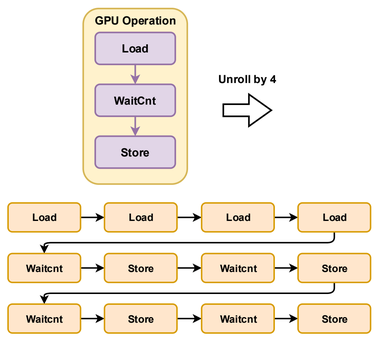
ASTRA-sim 3.0 **overview**

Detailed GPU Model

- Simulates **data** operations (`load/store`), **control** (`barrier/semaphore`)
- Data path modeling: dispatching cache-line-sized (128B) requests

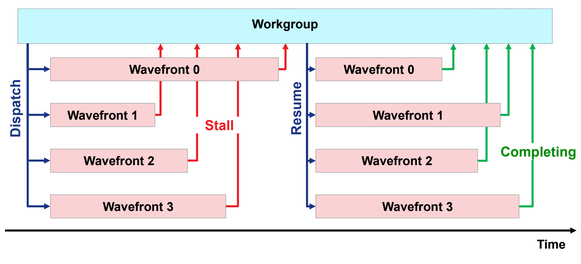


GPUs **execute WGs/WFs:** each WF generates cache-line (128B) network requests

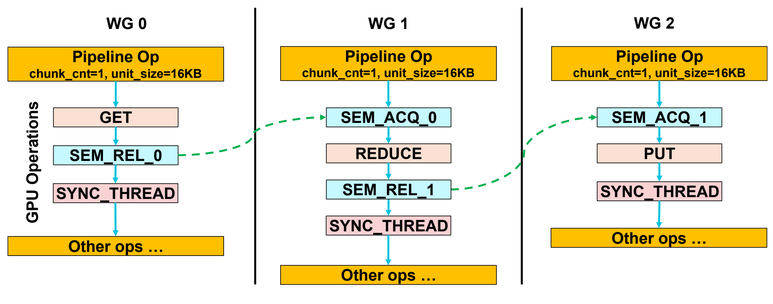


Loop unrolling enables **instruction-level parallelism**

- Control path modeling: barrier (intra-WG) and semaphore (inter-WG)



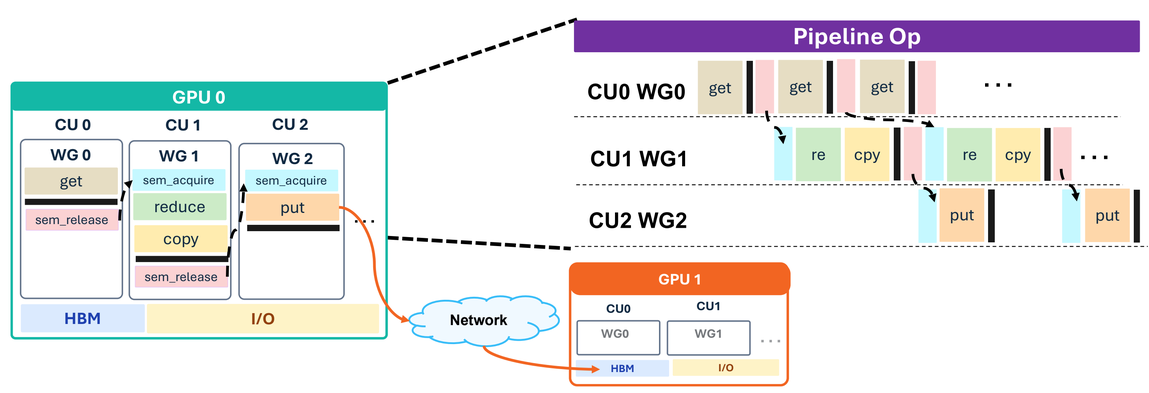
Barrier halts WF execution, synchronizes all WFs within a WG



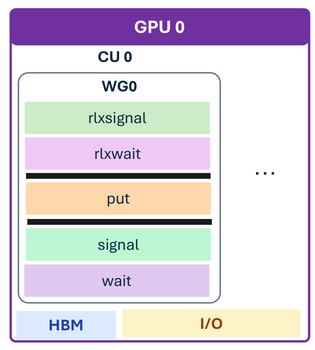
Semaphore halts WG execution, controls inter-WG control flow

Custom Collective Algorithm Support via MSCCL++

- Simulator needs a **flexible abstraction** to represent what GPUs do
  - **MSCCL++** is a domain-specific representation for collective communications



- MSCCL++ enables to represent flexible GPU communication behavior




```
"gpus": [{
  "id": 0,
  "input_chunks": 1,
  "threadblocks": [{
    "id": 0,
    "ops": [
      {"name":"rlxsignal","channel_type":"memory"},
      {"name":"rlxwait", "channel_type":"memory"},

      {"name":"nop"},

      {"name":"put",
       "src_buff":[{"index":0,"size":1}],
       "dst_buff":[{"buffer_id":0,"size":1}],
       "channel_type":"memory"},

      {"name":"nop"}

      {"name":"signal","channel_type":"memory"},
      {"name":"wait",  "channel_type":"memory"},]
}] }]
```

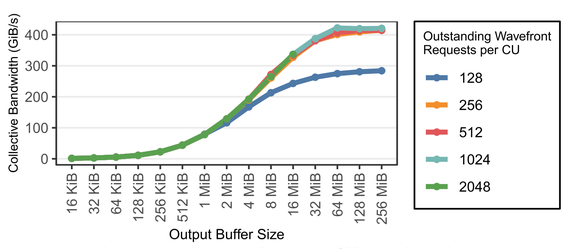
- MSCCL++ lowers to **JSON**, input to the **ASTRA-sim** 3.0
  - ASTRA-sim 3.0 parses and constructs **GPU instructions** to simulate

Preliminary Results

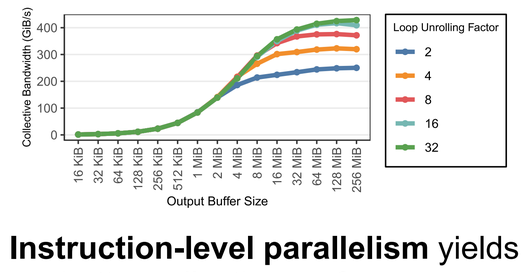
- ASTRA-sim 3.0 **enables** interesting design space explorations

Saturating scale-up bandwidth needs **more workgroups per GPU**

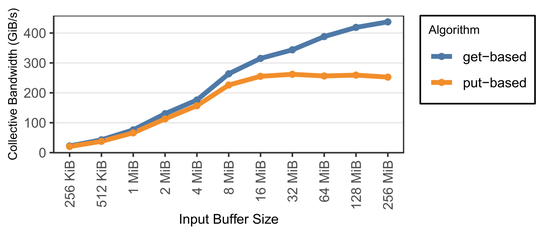
Simulation time scales **linearly to the comm size**



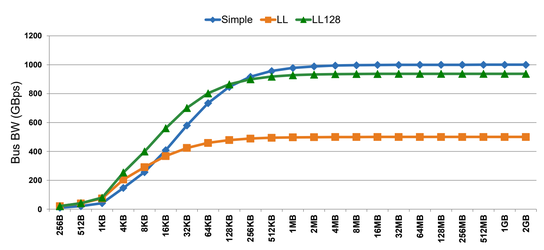
Larger register file yields better collective performance



Instruction-level parallelism yields better collective performance



`get` is better than `put` for Reduce-Scatter



Collective size impacts **communication protocol** preference

Disclaimer