Microsoft

MSCCL++: Rethinking GPU Communication Abstractions for AI Inference

Changho Hwang, Peng Cheng, Roshan Dathathri, Abhinav Jangda, Saeed Maleki, Madan Musuvathi, Olli Saarikivi, Aashaka Shah, Ziyue Yang

Microsoft Research

Binyang Li, Caio Rocha, Qinghua Zhou, Mahdieh Ghazimirsaeed, Sreevatsa Anantharamu, Jithin Jose

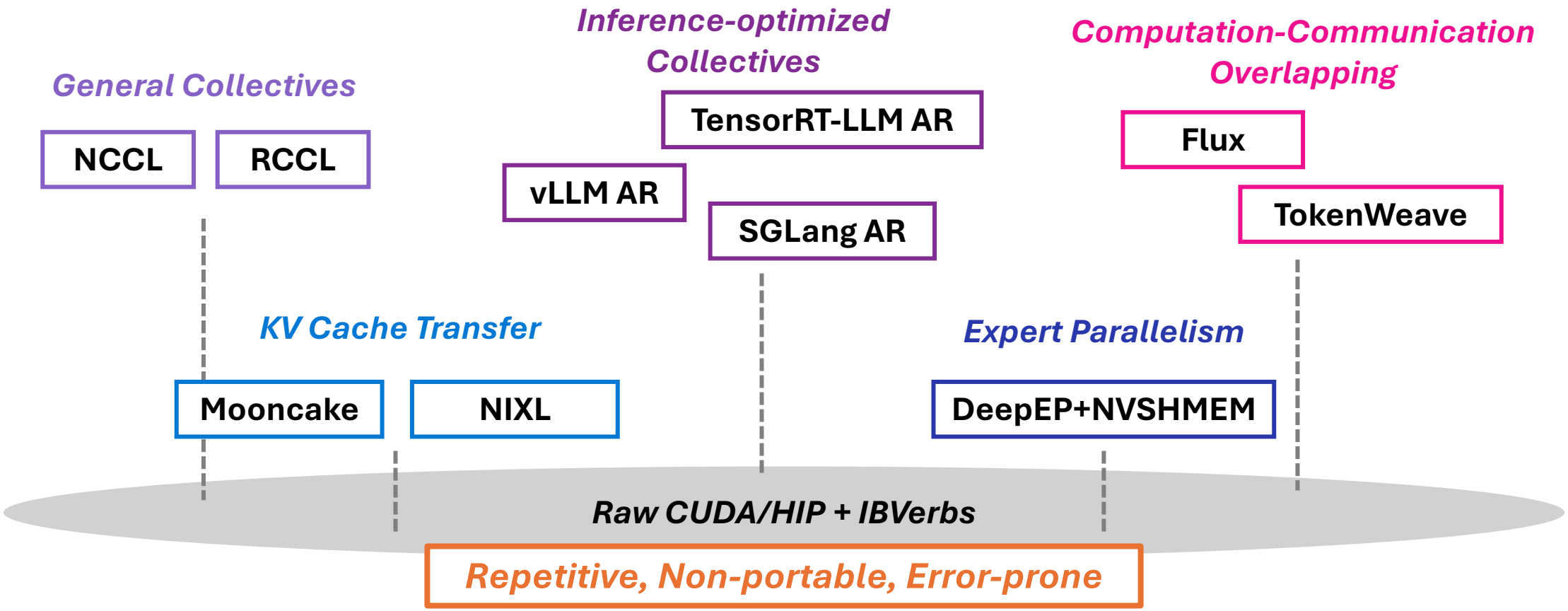
Microsoft Azure

AI Evolves with Advancing Interconnects

Model & System	*Interconnect*
Tensor parallelism	• On-package interconnect (NVLink) • In-network computation (NVLS)
PagedAttention & distributed KV cache	
Scale-out expert parallelism	• Low-latency direct access (IBGDA, GDS) • Scale-out high-bw interconnect (NVL72)
Distributed agentic inference?	• **Future Interconnects?**

Fragmented Software Landscape

- Each evolution spawned *its own communication stack*



Rethinking the Abstractions

Easy-to-use

Slow HW adoption

Minimal; powerful optimization

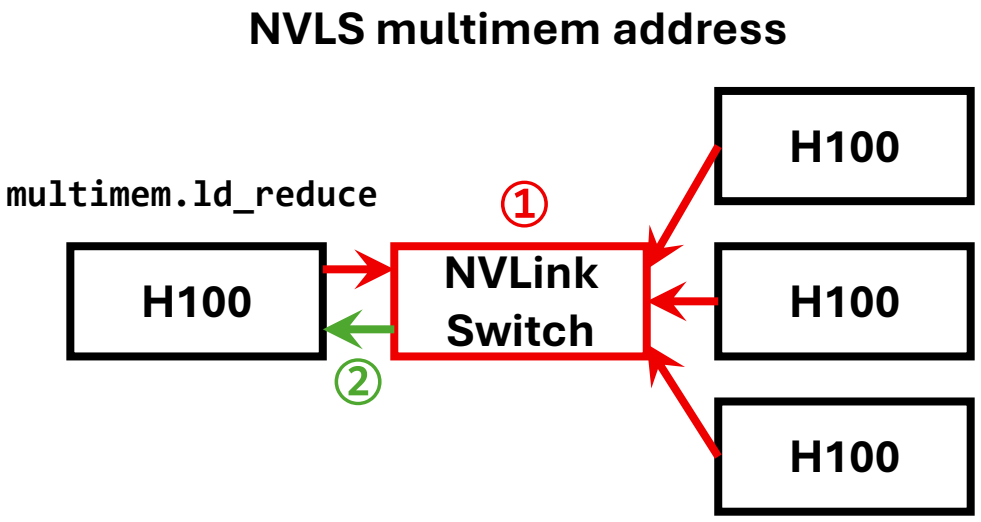
Less portable, difficult, error-prone

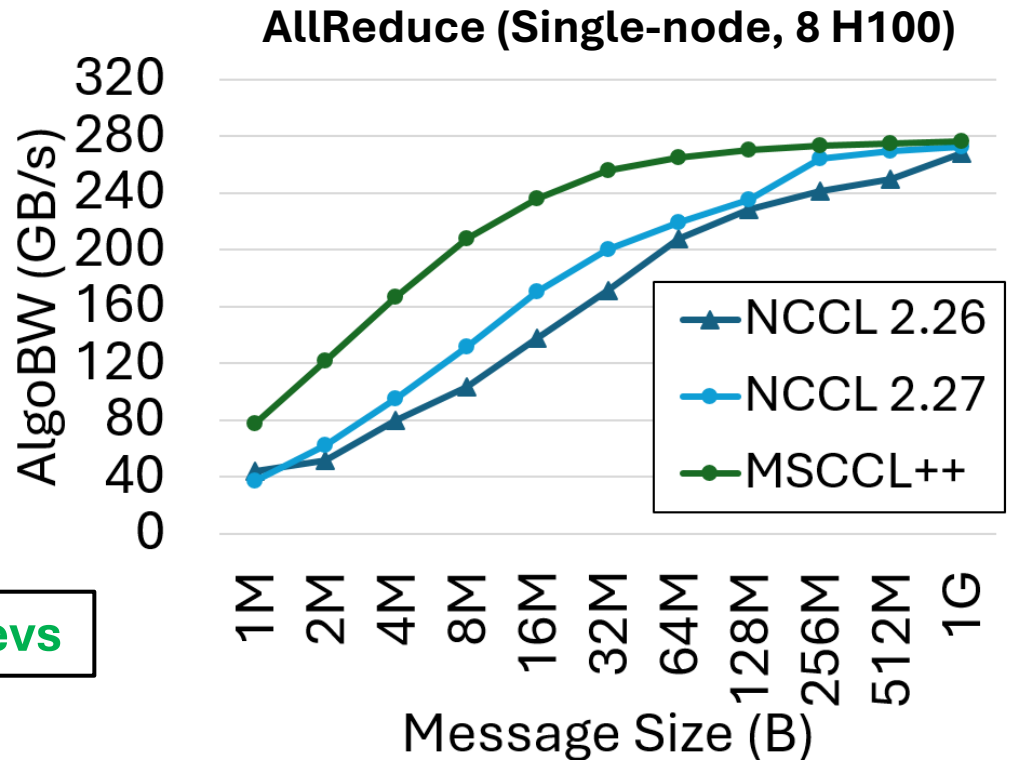
- **MSCCL++: multi-layer, portable abstraction** for GPU communication
  em-sickle-plus-plus
  - **Primitive API** for expert practitioners; provides **full control**
  - **DSL API** for app developers; **quick & easy optimization**
  - **Collective API** for agnostic users; provides **best-effort solutions**
  - *Separation of concerns*

Adopted in AMD RCCL, SGLang, and Microsoft Azure AI services

Easy-to-use Abstractions?

- Abstractions may hinder fast-evolving HW capabilities
  - Practitioners cannot wait long for adoption by third-parties
  - E.g., NVLink SHARP (NVLS) adoption by NCCL & PyTorch took ***years***



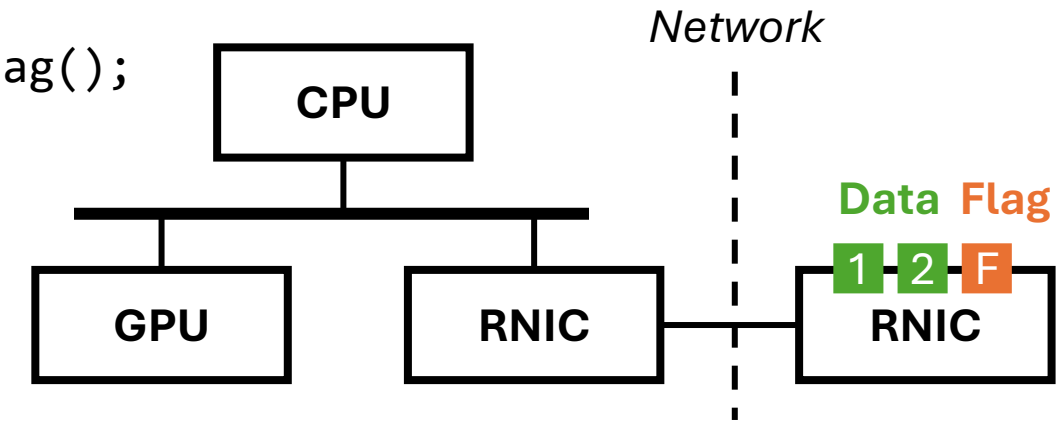


MSCCL++ (ours) NVLS support took **8 weeks of 2 devs**

Minimal Abstractions?

- Expose HW directly to users? [Exokernel, SOSP ‘95]
- Today’s AI platforms are nascent; less documented, less intuitive

```
for(;;) flag = poll_flag();
write_to_gpu(flag);
```

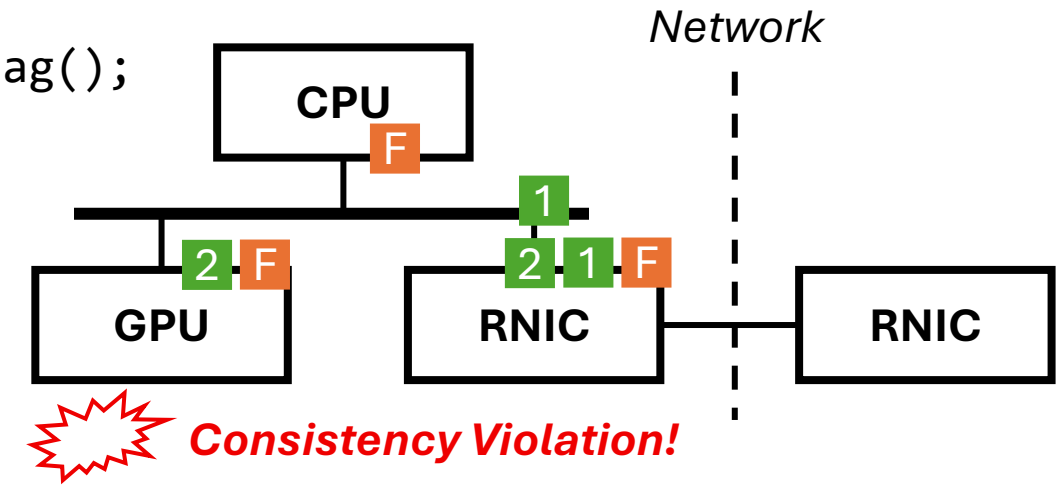



```
rdma_write_to_gpu(data);
rdma_write_to_cpu(flag);
```

Minimal Abstractions?

- Expose HW directly to users? [Exokernel, SOSP ‘95]
- Today’s AI platforms are nascent; less documented, less intuitive

```
for(;;) flag = poll_flag();
write_to_gpu(flag);
```

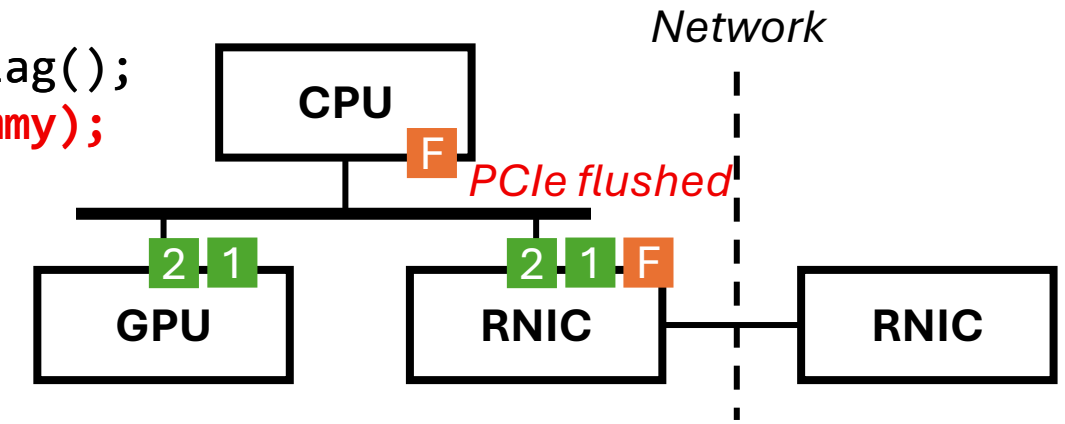



```
rdma_write_to_gpu(data);
rdma_write_to_cpu(flag);
```

Minimal Abstractions?

- Expose HW directly to users? [Exokernel, SOSP ‘95]
- Today’s AI platforms are nascent; less documented, less intuitive

```
for(;;) flag = poll_flag();
rdma_read_from_gpu(dummy);
write_to_gpu(flag);
```

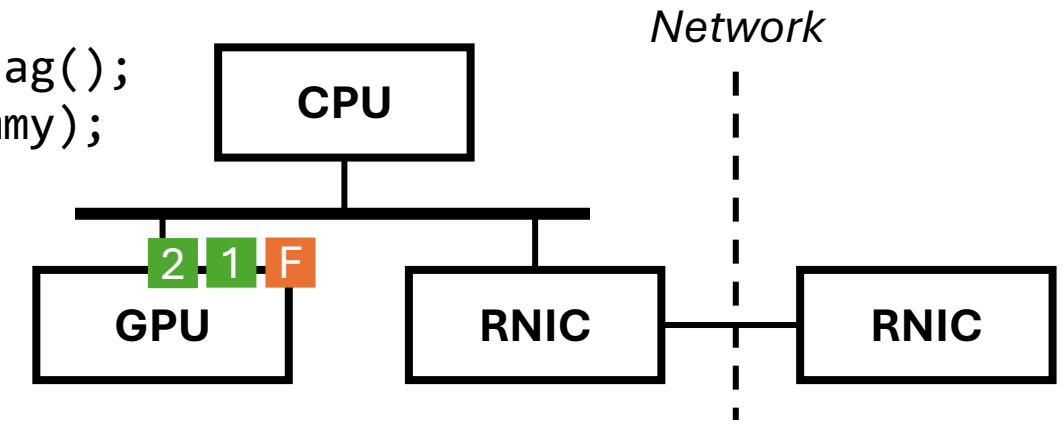



```
rdma_write_to_gpu(data);
rdma_write_to_cpu(flag);
```

Minimal Abstractions?

- Expose HW directly to users? [Exokernel, SOSP ‘95]
- Today’s AI platforms are nascent; less documented, less intuitive

```
for(;;) flag = poll_flag();
rdma_read_from_gpu(dummy);
write_to_gpu(flag);
```

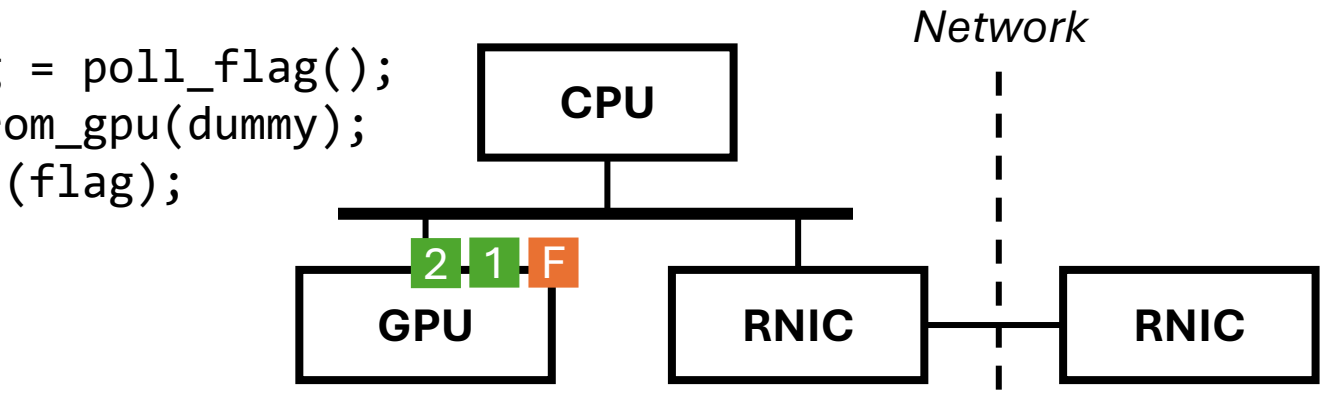



```
rdma_write_to_gpu(data);
rdma_write_to_cpu(flag);
```

Minimal Abstractions?

- Expose HW directly to users? [Exokernel, SOSP '95]
- Today's AI platforms are nascent; less documented, less intuitive

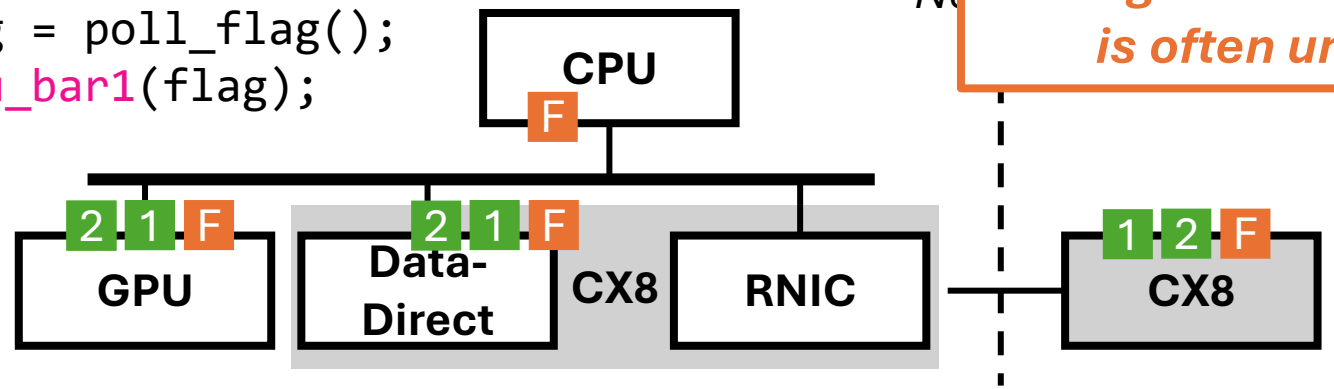
```
for(;;) flag = poll_flag();
rdma_read_from_gpu(dummy);
write_to_gpu(flag);
```




```
rdma_write_to_gpu(data);
rdma_write_to_cpu(flag);
```

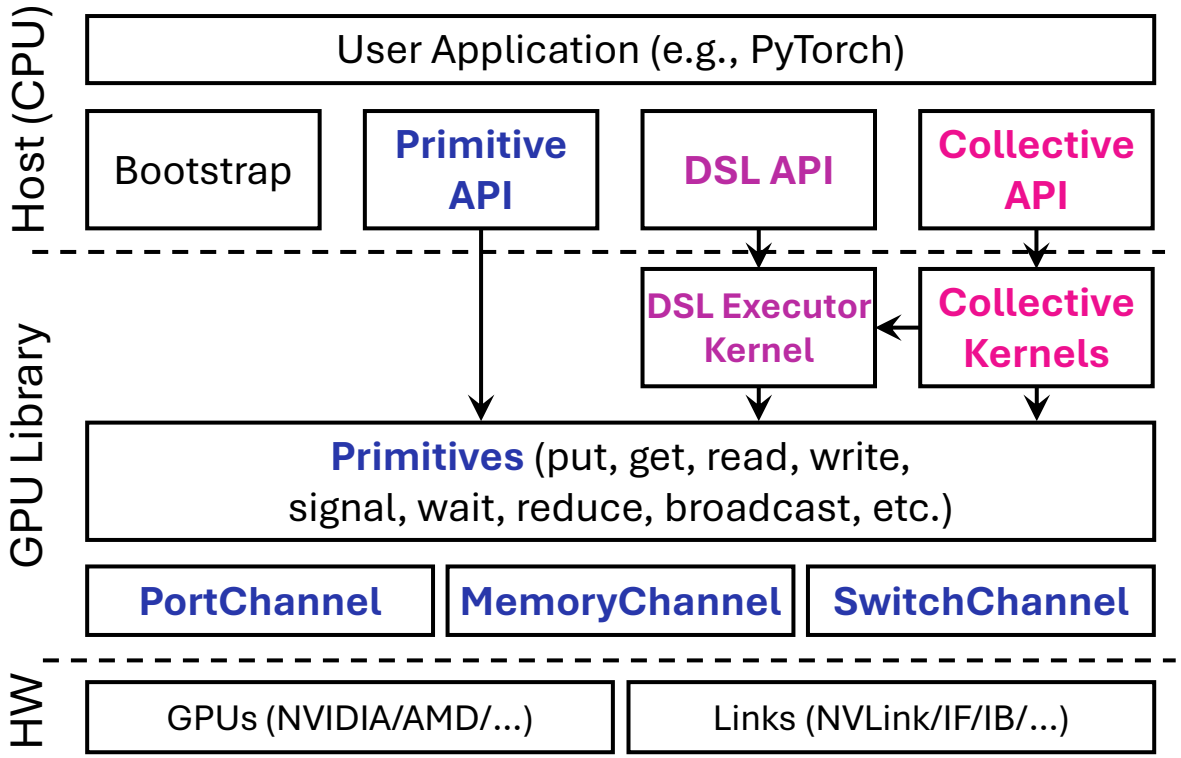
Using latest HW efficiently & correctly is often uneasy even for experts

```
for(;;) flag = poll_flag();
write_to_gpu_bar1(flag);
```



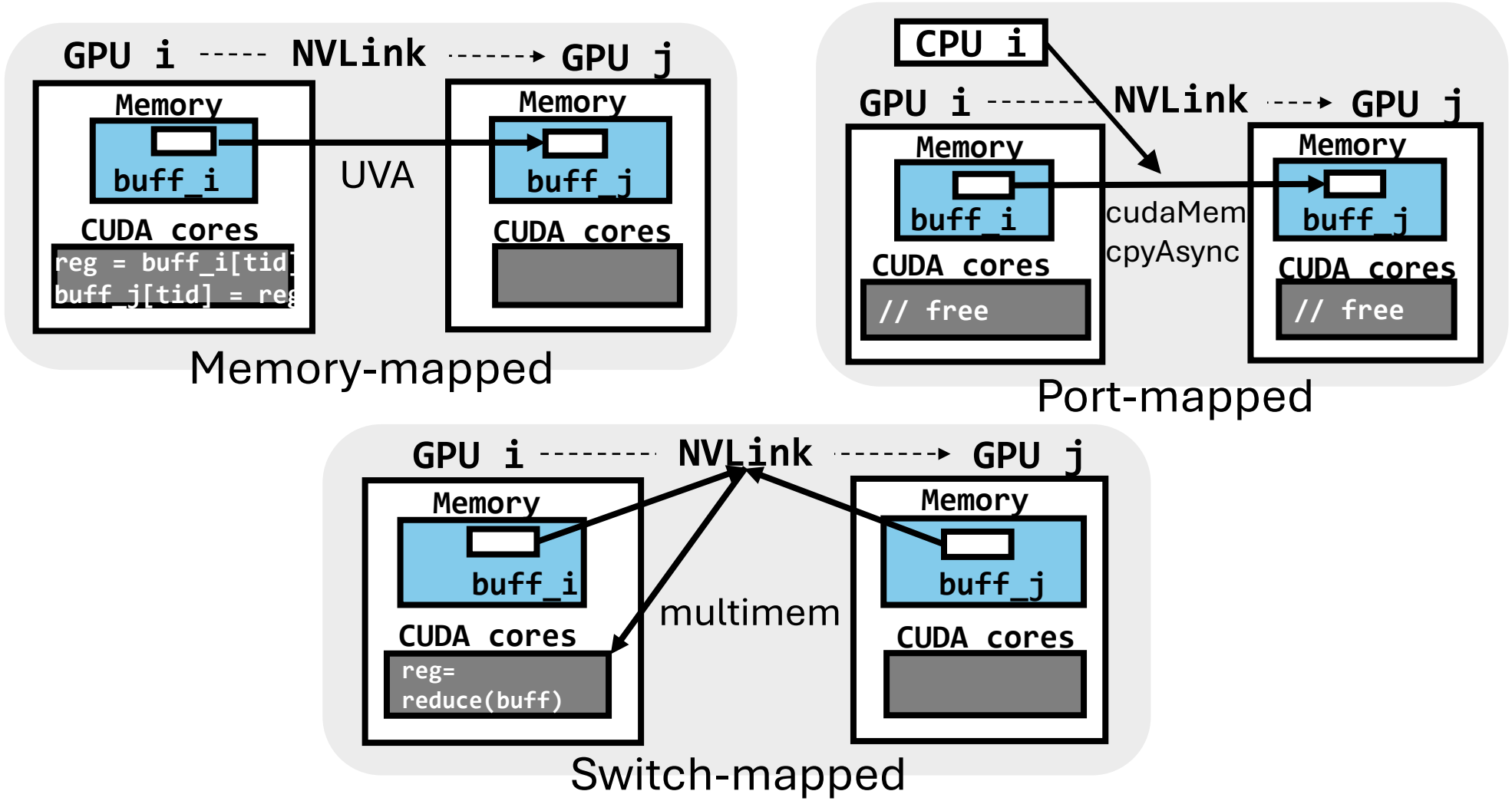

```
mlx5dv_write_to_gpu(data);
mlx5dv_write_to_cpu(flag);
```

MSCCL++ Overview: Hierarchical API Layers



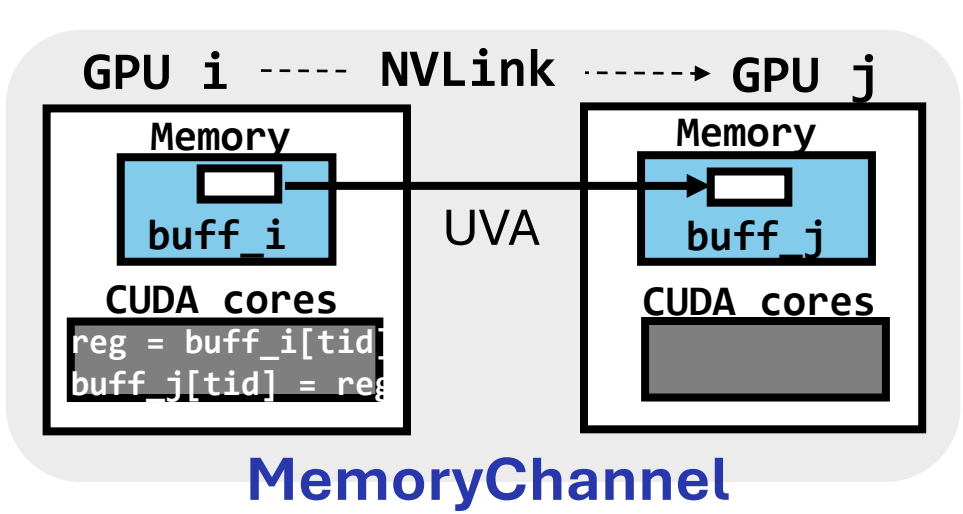
Primitive API: Channels

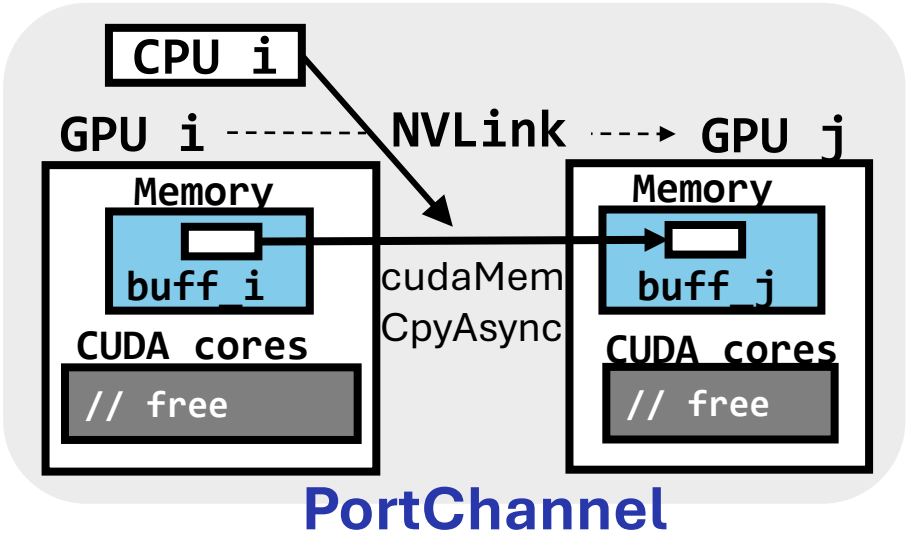
- Interfaces matching HW-level operations *on GPU*

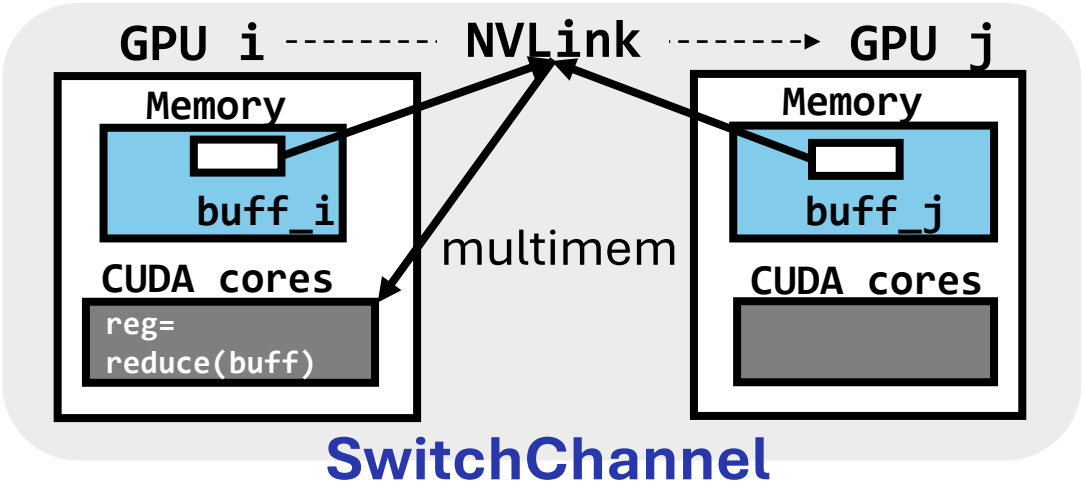


Primitive API: Channels

- Interfaces matching HW-level operations *on GPU*

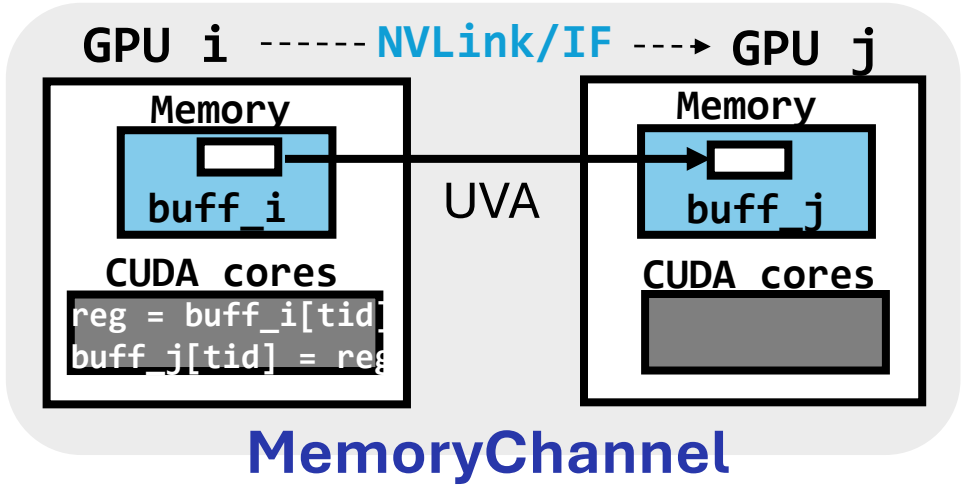


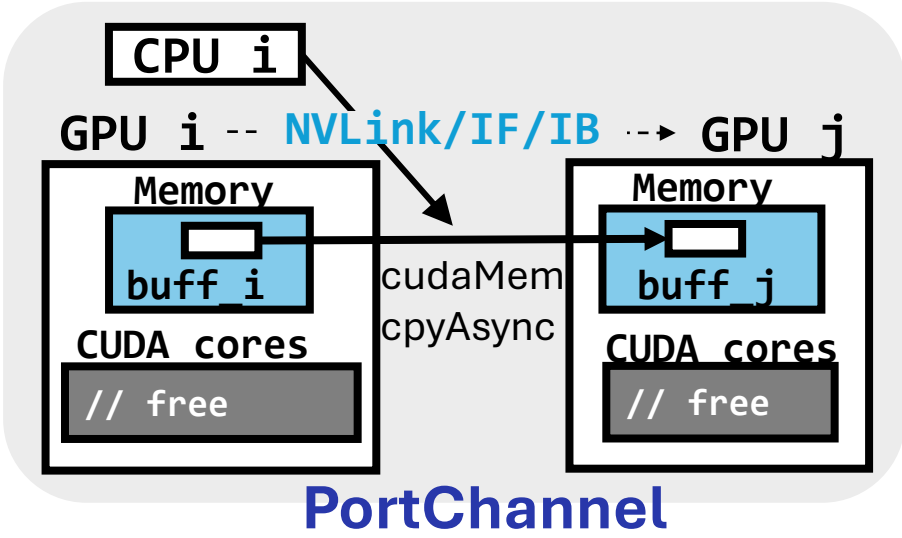


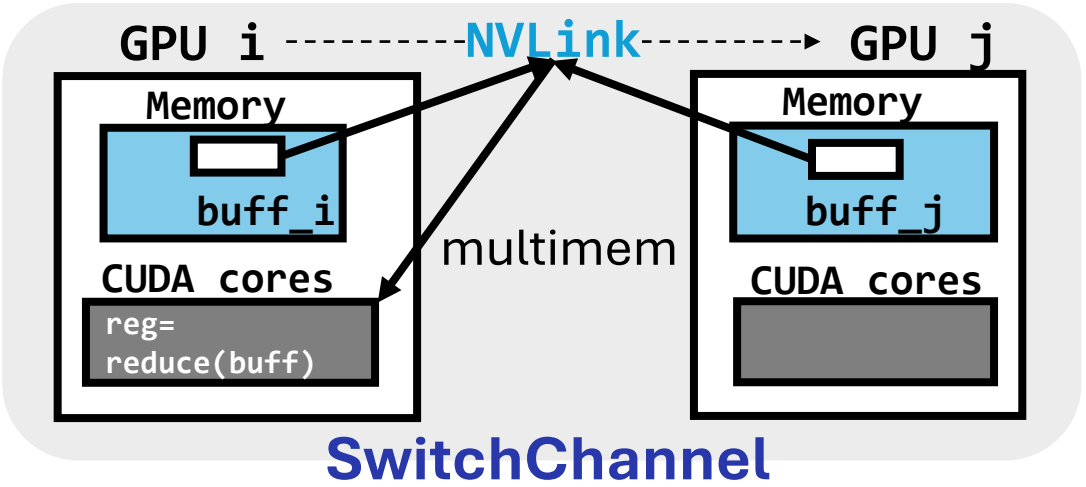


Primitive API: Channels

- Interfaces matching HW-level operations *on GPU*



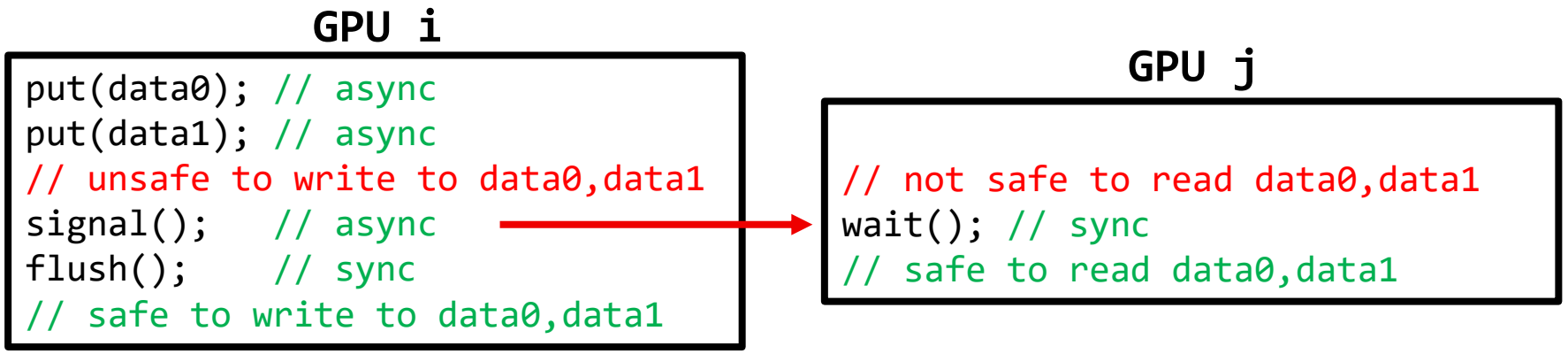




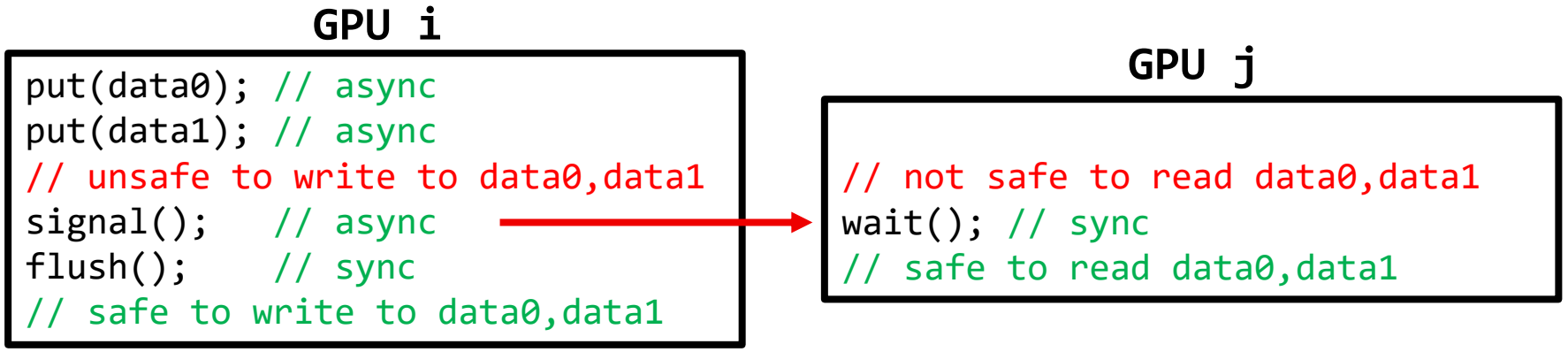
Portable across diverse link types

Primitive API: Benefits

- Near-hardware-peak performance
  - Minimal abstractions: zero-copy, one-sided, asynchronous
  - Tightly coupled & overlapped with GPU computation inside kernels
- Abstracted yet extensible by design
  - Channels are defined to match hardware



Primitive API: Methods Overview




```
class PortChannel
   void put(ulong dstOff,
            ulong srcOff,
            ulong sz);
   void signal();
   void wait();
   void flush();
```

```
class MemoryChannel
   void put(ulong dstOff,
            ulong srcOff,
            ulong sz,
            uint tid,
            uint tids);
   T read(ulong off);
   void write(ulong off, T elem);
   void signal();
   void wait();
   void flush();
```

```
class SwitchChannel
   T reduce(ulong dstOff,
            ulong srcOff,
            ulong sz);
   void broadcast(ulong dstOff,
                  ulong srcOff,
                  ulong sz);
```

Methods are simplified for readability

DSL API

- Write collective algorithms with a ***global view*** *(threadblocks and ranks)*
  - Python-native language
  - Channel interfaces + Buffer / MultimemBuffer abstractions

- Quickly draft & test new algorithms
  - Plugged into Collective APIs

```
class MemoryChannel
    def __init__(self,
                 dst_rank : int,
                 src_rank : int)

    def put(self,
            dst_chunk : Buffer,
            src_chunk : Buffer,
            tb : int,
            tb_group : TBGroup)

    ...
```

Evaluations

- MSCCL++ has been open-sourced for 2.5y+
- Evaluated in real practices for:
  - High-performance collectives for LLM inference
  - Quick HW adoption & low development efforts
  - Reusability in various application scenarios (e.g., DeepEP)

Evaluations

- MSCCL++ has been open-sourced for 2.5y+
- Evaluated in real practices for:
  - **High-performance collectives for LLM inference**
  - **Quick HW adoption & low development efforts**
  - Reusability in various application scenarios (e.g., DeepEP)

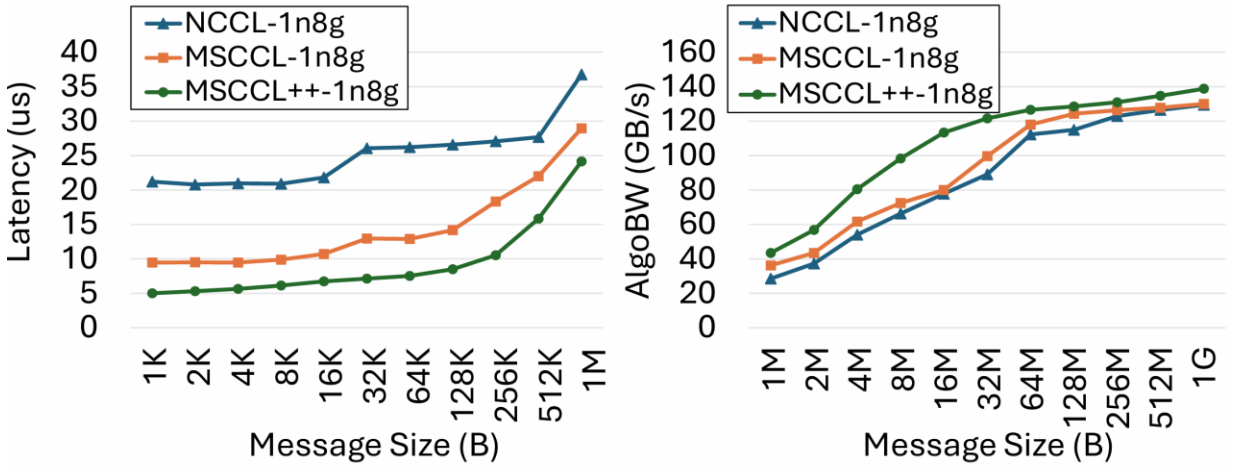
Please check details out from our paper!

High-performance Collectives

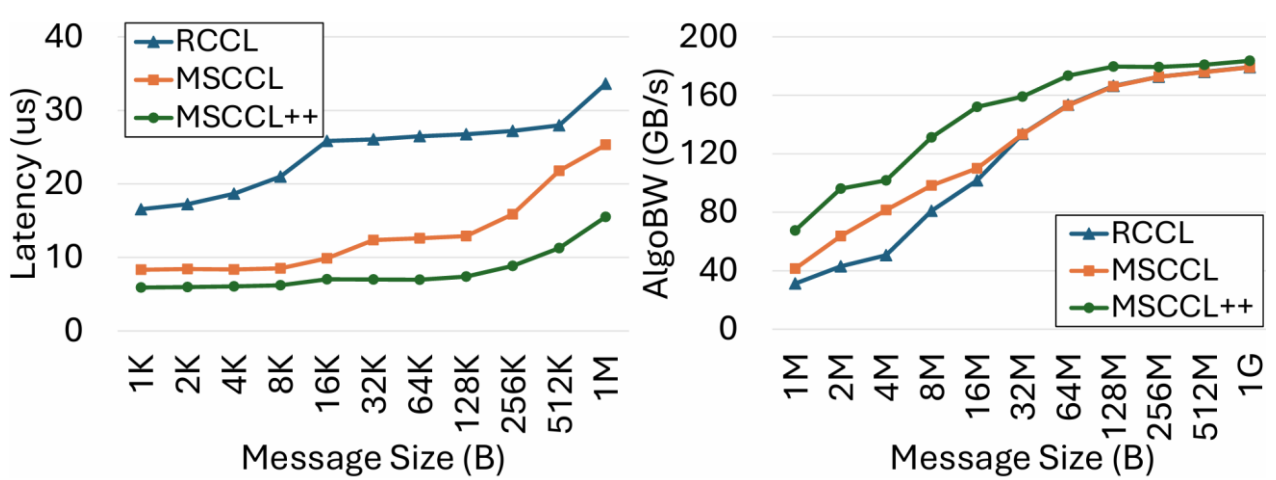
- AllReduce & AllGather
  - Geomean **1.7x** (up to 5.4x) speedup
  - Compared to:
    - NCCL/RCCL
    - MSCCL (the same HW abstractions as NCCL/RCCL)

MSCCL++ Channel abstractions enable optimized algorithm & high performance

AllReduce, 8 A100

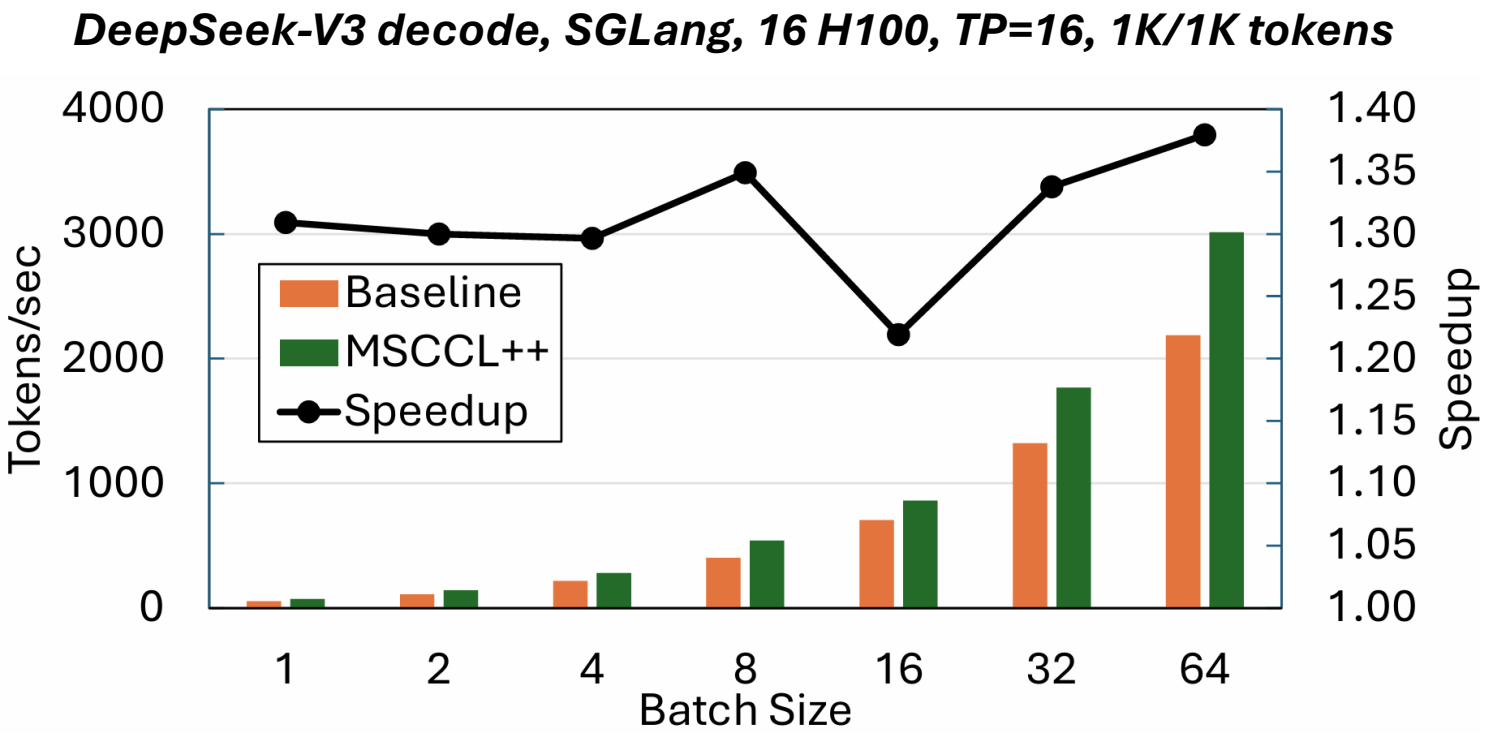


AllReduce, 8 MI300x



LLM Inference Speedup

- LLM end-to-end inference with MSCCL++ collectives
  - Geomean **1.2x** (up to 1.38x) speedup



Quick HW Adoption & Low Dev Efforts

- AMD MI300x support from NVIDIA-only: **7 weeks of 1 dev**
  - From learning about HIP to outperforming RCCL in 8-GPU AllReduce
  - 1307 LoC; compared to RCCL's 35480 LoC diffs before & after MI300x

- NVLS support (adding SwitchChannel): **8 weeks of 2 devs**
  - From learning about NVLS to outperforming NCCL in 8-GPU AllReduce

- Multi-Node NVLink (MNNVL, NVL72) support: **2 weeks of 1 dev**

MSCCL++ offers flexible HW abstractions & easy optimization

Summary

- Separation of concerns:
  - Primitive API: flexible, abstraction of the hardware
  - DSL and Collective API: optimized, potable implementations
- Improves performance over NCCL, RCCL, and MSCCL:
  - 1.7x (up to 5.4x) for collective communication
  - 1.2x (up to 1.4x) for AI inference workloads
- Adopted in AMD RCCL, SGLang, and Microsoft Azure AI services

microsoft/mscclpp