# Triton-distributed: Programming Overlapping Kernels on Distributed AI Systems with the Triton Compiler

Size Zheng[1,2,†] , Wenlei Bao[1,†] , Qi Hou[1] , Xuegui Zheng[1] , Jin Fang[1] , Chenhui Huang[1] , Tianqi Li[1,3] , Haojie Duanmu[1,4] , Renze Chen[1,3] , Ruifan Xu[1,3] , Yifan Guo[1,5] , Ningxin Zheng[1] , Ziheng Jiang[1] , Xinyi Di[1] , Dongyang Wang[1] , Jianxi Ye[1] , Haibin Lin[1] , Li-Wen Chang[1] , Liqiang Lu[5] , Yun Liang[3] , Jidong Zhai[2] , Xin Liu[1,†]

[1]ByteDance Seed, [2]Tsinghua University, [3]Peking University,
[4]Shanghai Jiao Tong University, [5]Zhejiang University
[†]Corresponding authors

**Shared by Zhou Ouxiang**

**2025.6.17**

# TL;DR（Too Long; Didn't Read）

- The authors developed **Triton-distributed** to achieve performance competitive with low-level CUDA/C++ at a **fraction of the development cost**.
- The approach requires **minimal changes** to existing Triton compute kernels.
- It enables **rapid hardware support**, making it ideal for adapting AI workloads across a diverse ecosystem of chips.

# Outline

- Background
- The Triton-distributed Architecture & Programming Model
- Overlapping Optimizations in Triton-distributed
- Experiments & Evaluations
- Conclusion

# Outline

- Background
- The Triton-distributed Architecture & Programming Model
- Overlapping Optimizations in Triton-distributed
- Experiments & Evaluations
- Conclusion

# Background: Beyond a Single Chip

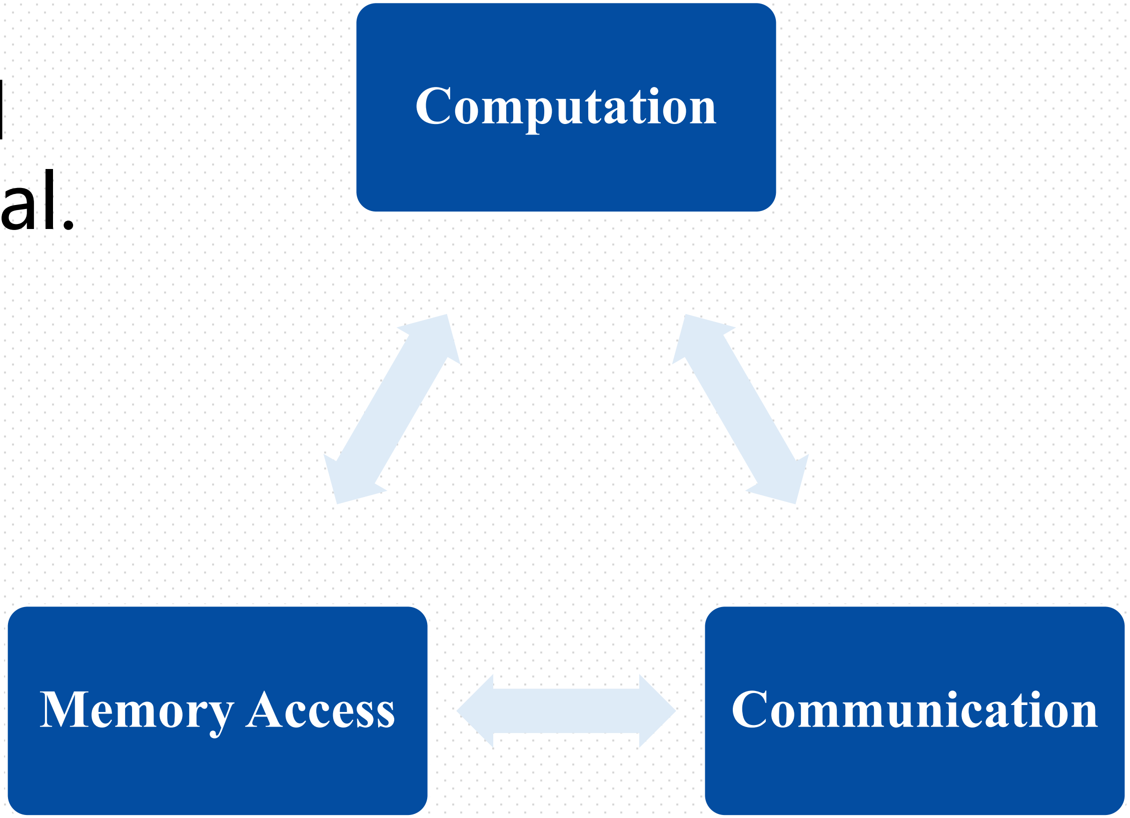
- Large Language Models (LLMs) have outgrown the memory and compute capacity of single accelerators.

- Distributed systems, composed of multiple accelerators, are now essential for both training and inference.

- This shift introduces significant new complexities.

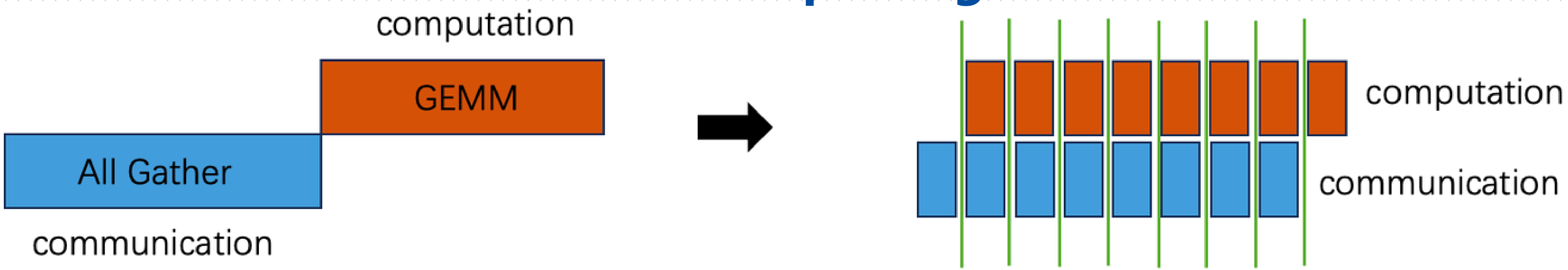
# Background: Computation-Communication Overlap

- As cluster scale grows, hiding communication latency behind computation time becomes vital.
- Effective overlap can save millions of GPU hours and significant operational costs (e.g., ByteDance's COMET project).

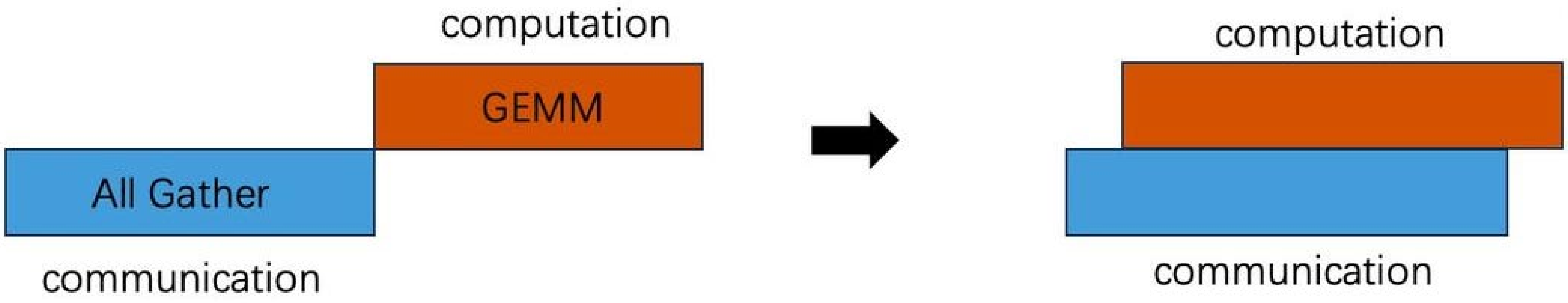


# Background: Computation-Communication Overlap

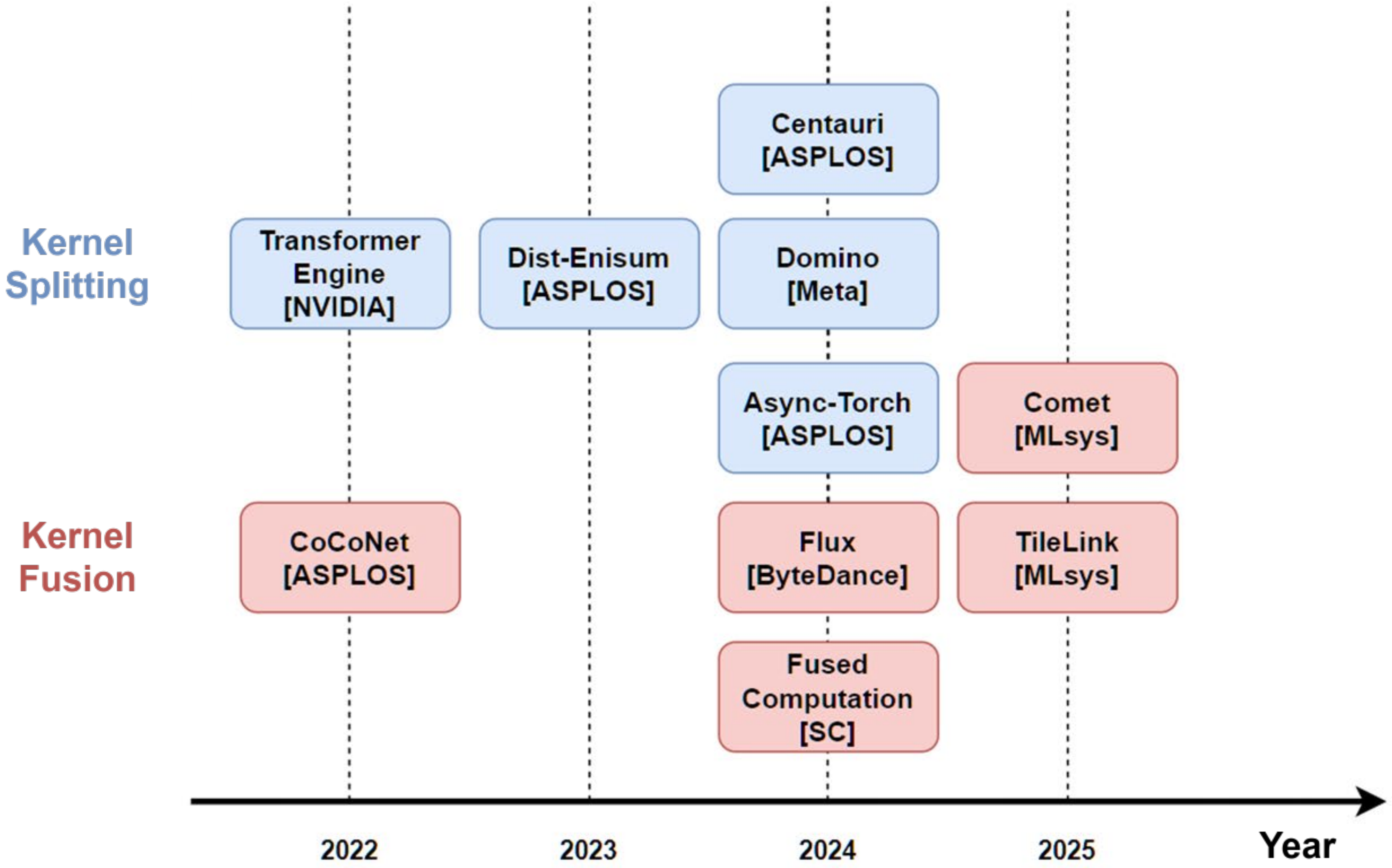
## Kernel Splitting




## Kernel Fusion

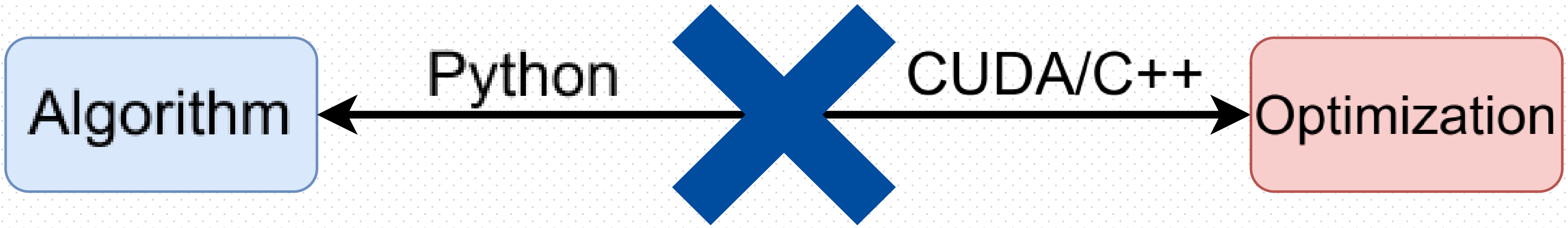


# Background: Computation-Communication Overlap



# The Gap Between Programming

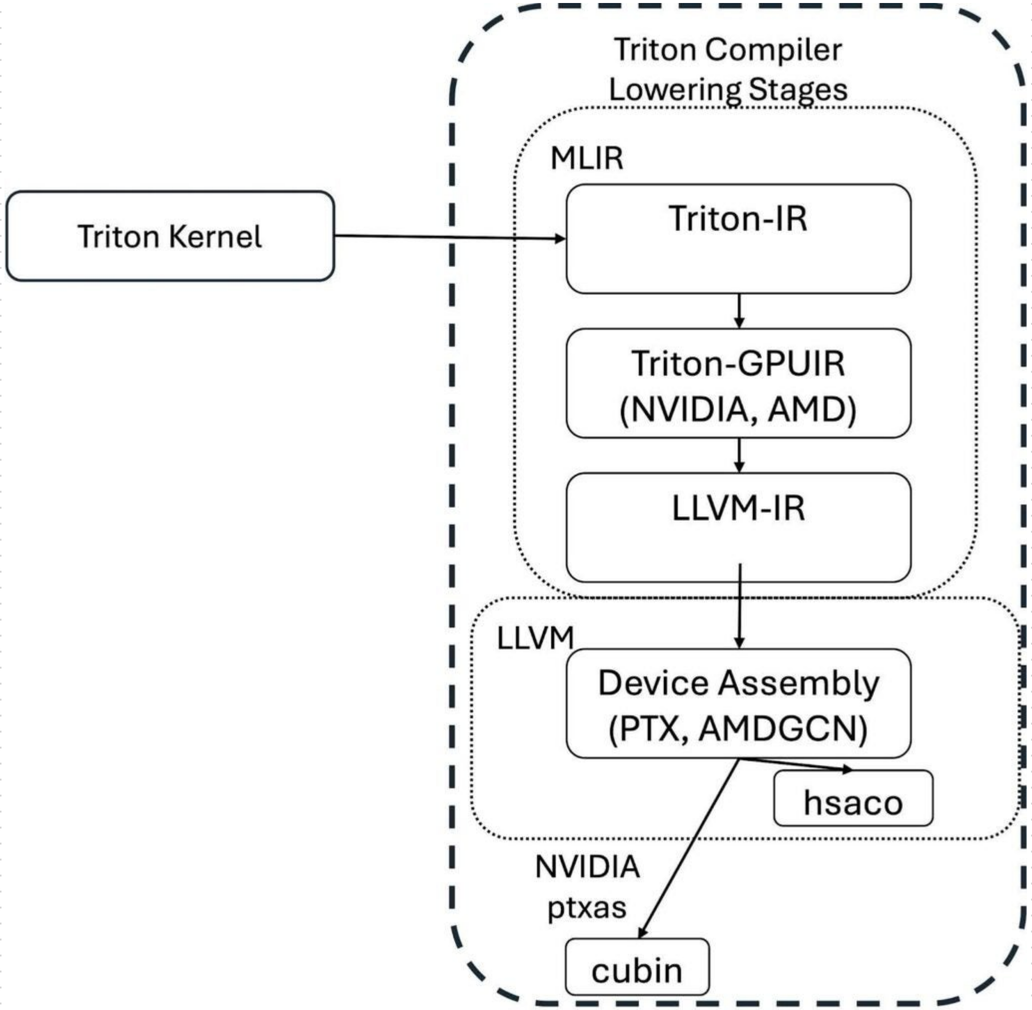
- AI Algorithms are developed in high-level **Python**.
- Performance-critical optimizations require low-level **CUDA/C++**.



# Background: What is Triton?

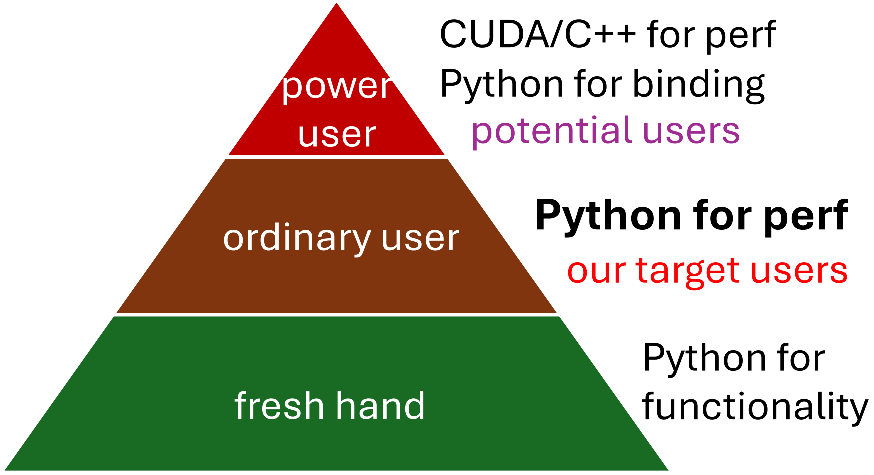
- ◆A Python-based language and compiler for writing high-performance GPU kernels.

- ◆It solves the problem for a **single GPU**; Triton-distributed extends this to **distributed systems**.

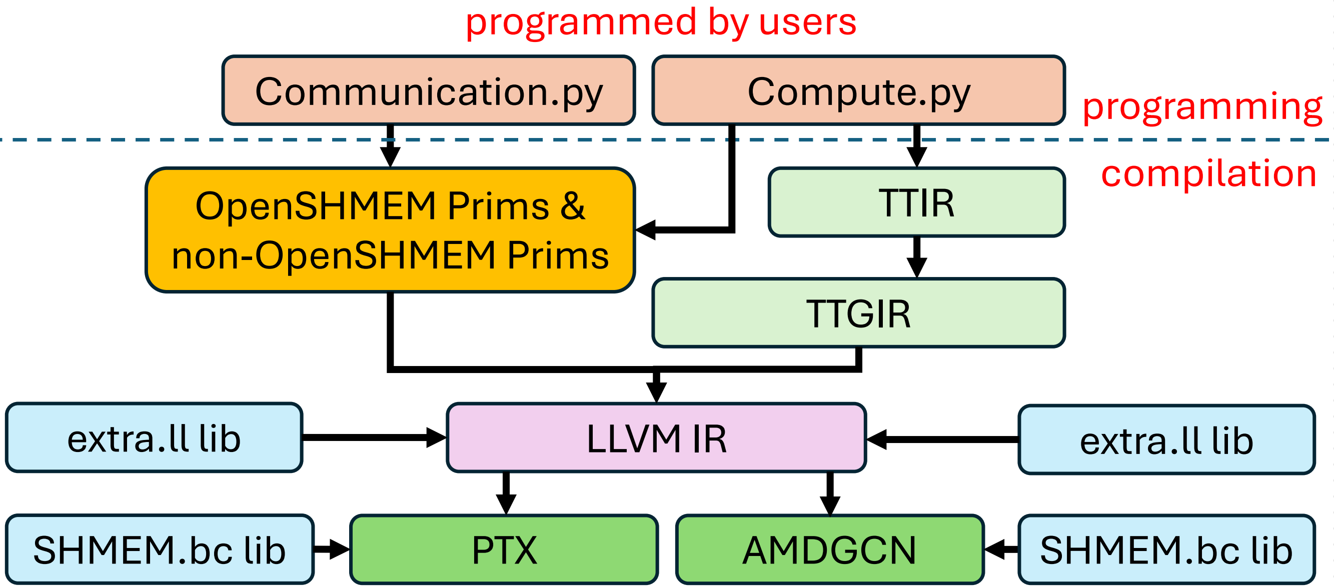


# Outline

# Triton-distributed Architecture



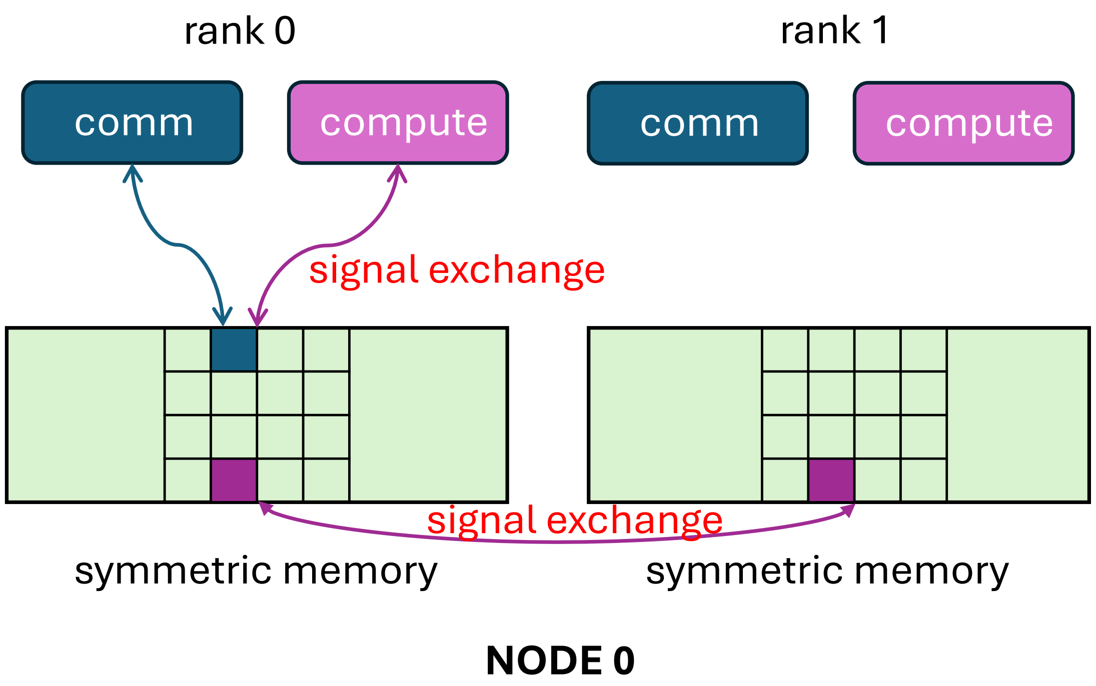

**User classification and our target**

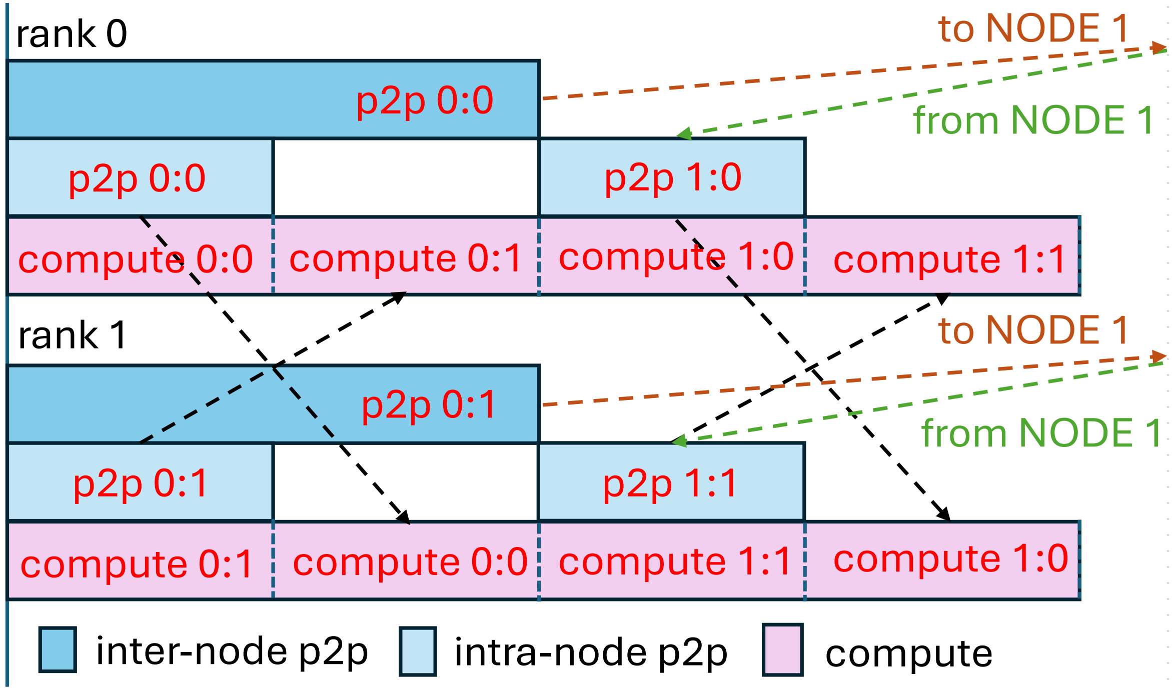



**Our compilation workflow**

# Triton-distributed Programming Model

- Symmetric Memory
- Signal Exchange
- Async-Task



# Communication Primitives of Triton-distributed

| **Primitive** | **Explanation** |
|---|---|
| | **OpenSHMEM Primitives** |
| *my_pe* | Get the current device id |
| *n_pes* | The number of devices in the world |
| *int_p* | Put an integer to remote device |
| *remote_ptr* | Convert local shared memory pointer to remote pointer |
| *barrier_all* | Barrier all the devices |
| *sync_all* | Synchronize all the devices |
| *quiet* | Ensure completion of shared memory operation of calling device |
| *fence* | Ensure order of shared memory operation of calling device |
| *getmem* | Blocking get data from remote device |
| *getmem_nbi* | Non-blocking get data from remote device |
| *putmem* | Blocking put data to remote device |
| *putmem_nbi* | Non-blocking put data to remote device |
| *putmem_signal* | Blocking put data and write signal to remote device |
| *putmem_signal_nbi* | Non-blocking put data and write signal to remote device |
| *signal_op* | Perform signal set/add operation to remote |
| *signal_wait_until* | Wait local signal until condition is meet |
| *broadcast* | Broadcast data into all the other ranks |

# Communication Primitives of Triton-distributed

| Primitive | Explanation |
|---|---|
| | **non-OpenSHMEM Primitives** |
| *wait* | Locally wait a signal until it equals to some given value |
| *consume_token* | used with *wait* primitive to create data dependency |
| *notify* | Notify a remote signal, similar to *signal_op* primitive |
| *atomic_cas* | Atomic compare and swap |
| *atomic_add* | Atomic add value |
| *ld_acquire* | Load with acquire semantic |
| *red_release* | Reduction add with release semantic |
| *multimem_ld_reduce* | Multimem load data and perform reduction |
| *multimem_st* | Multimem broadcast data |

# Example: Inter-node Overlapping AllGather GEMM

```
@triton.jit
def producer_allgather(
    A, signal_num_elem_per_rank,
    rank, local_world_size, world_size):
    pid = tl.program_id(0)
    node = rank // local_world_size
    local_rank = rank % local_world_size
    n_nodes = world_size // local_world_size

    if pid < local_world_size - 1:
        peer = (local_rank + pid + 1) % local_world_size \
                + node * local_world_size
        for i in range(n_nodes):
            seg = (local_rank +
               ((node + i) % n_nodes) * local_world_size)
            if tid(0) == 0:
                signal_wait_until(signal + seg, EQ, 1)
            __syncthreads()
            putmem_signal(
                A + seg * num_elem_per_rank,
                A + seg * num_elem_per_rank,
                signal + seg,
                1, SET, peer)

    else:
        pid = pid - local_world_size + 1
        if tid(0) == 0:
            signal_wait_until(signal + rank, EQ, 1)
        __syncthreads()
        peer = (local_rank + (node + pid + 1) % n_nodes \
            * local_world_size)
        putmem_signal(
            A + rank * num_elem_per_rank,
            A + rank * num_elem_per_rank,
            signal + rank,
            1, SET, peer)
```

Intra-node

Inter-node

```
@triton.jit
def consumer_gemm(A, B, C, signal):
    pid = tl.program_id(0)
    pid_m, pid_n = ...
    offs_A, offs_B, offs_C = ...
    acc = tl.zeros([TILE_M, TILE_N])
    for k in range(K // TILE_K):
        token = wait(
            signal + pid_m, 1, "gpu", "acquire", waitValue=1)
        a_ptrs = consume_token(A + offs_A, token)
        a_data = tl.load(a_ptrs)
        b_data = tl.load(b_ptrs)
        tl.dot(a_data, b_data, acc)
        offs_A, offs_B = ...
    tl.store(C + offs_C, acc)
```

```
def ag_gemm(A, B, C, signal):
    with comm_stream():
        grid = (local_world_size + n_nodes - 2, 1, 1)
        producer_allgather[grid](
            A, signal, num_elem_per_rank,
            rank, local_world_size, world_size)
    with compute_stream():
        grid = ((M//TILE_M) * (N//TILE_N), 1, 1)
        consumer_gemm[grid](A, B, C, signal)

A = create_tensor([global_M, K])
signal = create_tensor([world_size])
```

# Outline

- Background
- The Triton-distributed Architecture & Programming Model
- **Overlapping Optimizations in Triton-distributed**
- Experiments & Evaluations
- Conclusion

# Optimization Approaches and Comparison with Other Frameworks

| Name | NCCL | PyTorch | TE | Pallas | CoCoNet | FLUX | DeepEP | Ours |
|---|---|---|---|---|---|---|---|---|
| Intra-Node Swizzle | ■ | ✓ | ✓ | ■ | ✓ | ✓ | ■ | ✓ |
| Inter-Node Swizzle | ■ | ✗ | ✗ | ■ | ✓ | ✓ | ■ | ✓ |
| Inter-NUMA Swizzle | ■ | ✗ | ✗ | ✗ | ✗ | ✗ | ■ | ✓ |
| Copy Engine | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ | ✓ |
| High-BW Link | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Network Comm. | ✓ | ✓ | ✗ | ✓ | ✓ | ✓ | ✓ | ✓ |
| PCIe Comm. | ✓ | ✓ | ✗ | ✗ | ✗ | ✓ | ✗ | ✓ |
| OpenSHMEM Support | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ | ✓ |
| Low-latency Protocol | ✓ | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ | ✓ |
| Multimem Feature | ■ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ |
| Fusion | ✗ | ✗ | ✗ | ■ | ✓ | ✓ | ✓ | ✓ |
| Code Generation | ✗ | ✗ | ✗ | ✓ | ✓ | ✗ | ✗ | ✓ |
| Nvidia/AMD | ✓/✗ | ✓/✓ | ✓/✗ | ■/✗ | ✓/✗ | ✓/✗ | ✓/✗ | ✓/✓ |

# Communication Kernels (1/3): Intra-Node AllGather

- Primarily utilizes the dedicated Copy Engine to offload data transfer from compute cores.

- Offers two implementation modes:

  - **Push Mode (Algo 1):** Sender-initiated. Lower sync overhead, but uncontrolled arrival order.
  - **Pull Mode (Algo 2):** Receiver-initiated. Controlled order, but requires an extra barrier synchronization.

**Algorithm 2** One-sided Pull-mode Intra-node AllGather

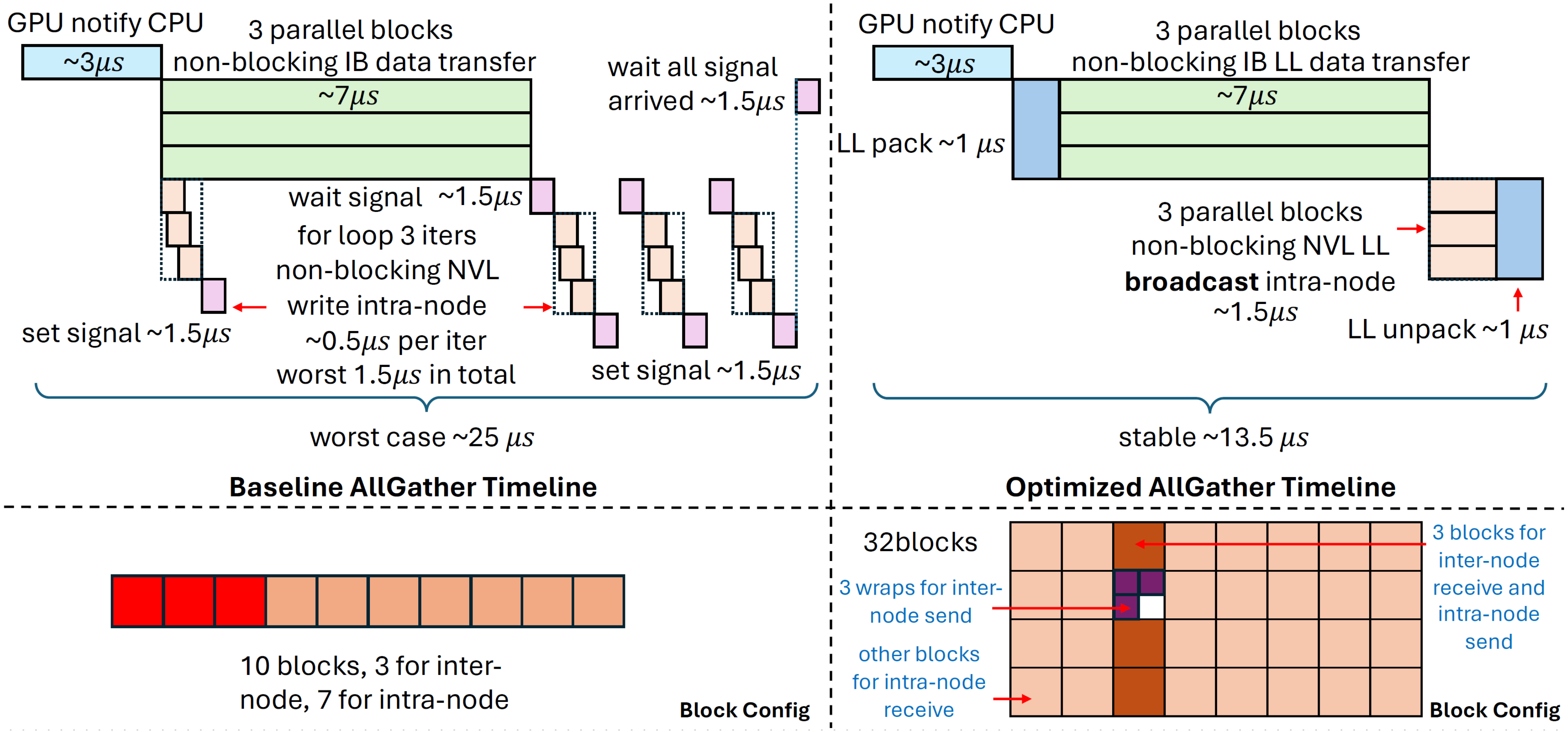
```
Input: Symmetric Buffer T, Signal S, Local Buffer L
local_t_buf = make_buffer(T + RANK ×L.size())
local_t_buf.copy_(L, L.size())
set_signal(S + RANK)
barrier_all()   // Make the local copy visible to all the other ranks
for r in range(WORLD_SIZE) do
    if r is not RANK then
        remote_buf = make_buffer(remote_ptr(T, r) + r × L.size()))
        local_t_buf = make_buffer(T + r × L.size())
        local_t_buf.copy_(remote_buf, L.size())
        set_signal(S + r)
    end if
end for
```

**Algorithm 1** One-sided Push-mode Intra-node AllGather

```
Input: Symmetric Buffer T, Signal S, Local Buffer L
for r in range(WORLD_SIZE) do
    remote_buf = make_buffer(remote_ptr(T, r) + RANK ×L.size())
    remote_buf.copy_(L, L.size()) // Memory Copy
    remote_sig = remote_ptr(S, r) + RANK
    set_signal(remote_sig)   // Notify the consumer
end for
```

# Communication Kernels (2/3): Low-Latency Inter-Node AllGather

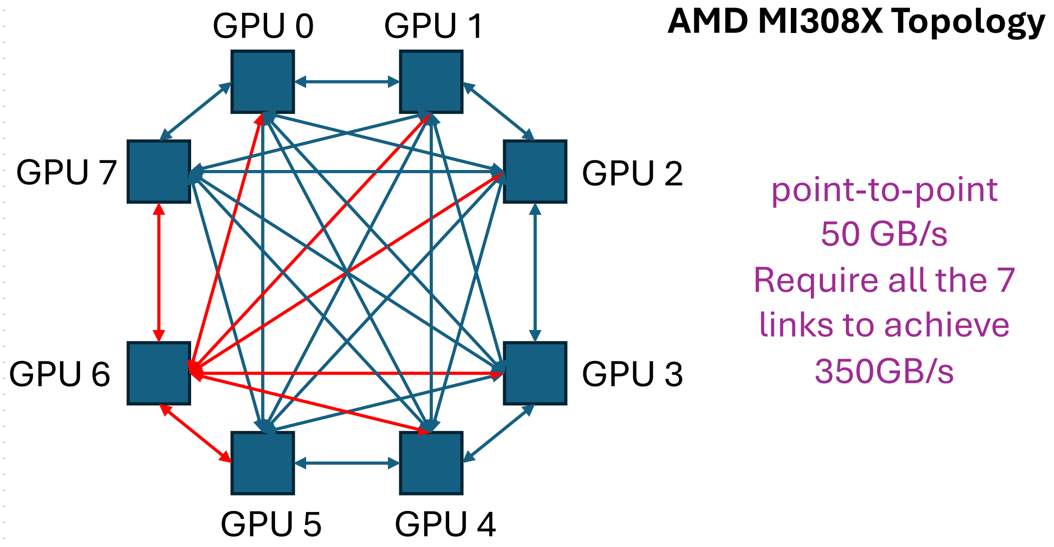
◆**Problem:** Baseline implementations can suffer from "skew," turning parallel sends into sequential ones and increasing latency.



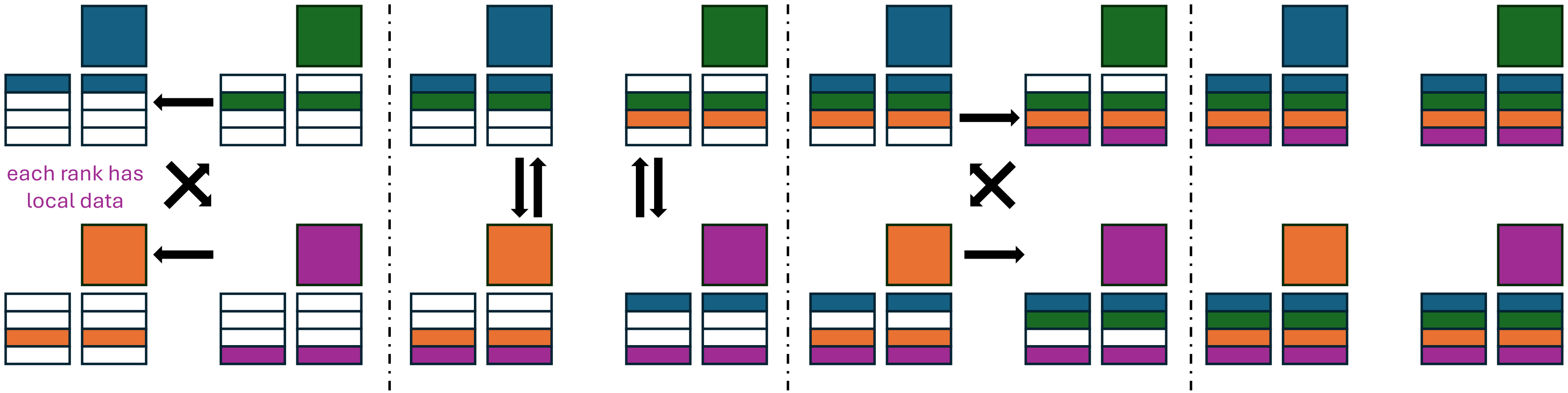
# Communication Kernels (3/3): Platform Adaptation (AMD)

- The framework adapts to different hardware topologies and behaviors.

- **On AMD MI308X:**
  - Requires launching transfers on multiple streams simultaneously to maximize bandwidth on its full-mesh topology.
  - Works around problematic driver APIs by fusing the scatter operation directly into the producer compute kernel, avoiding the API call.



# Overlapping Computation with Swizzling Optimization



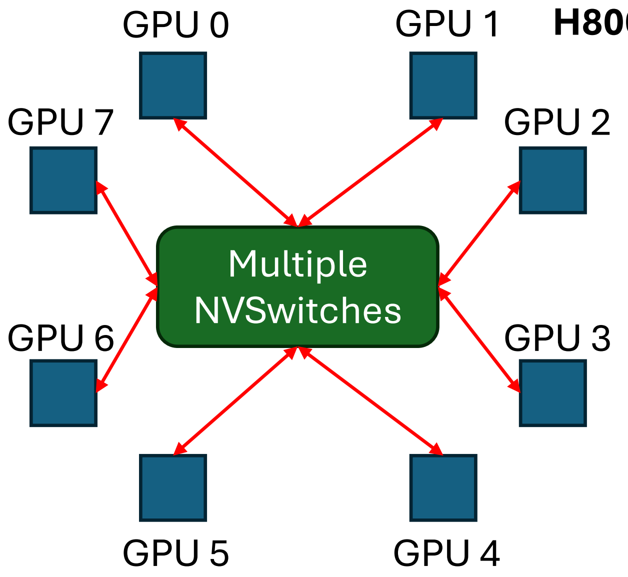

Step 1 compute with local data and gather data

Step 2 compute with Step 1 data and gather data

Step 3 compute with Step 2 data and gather data

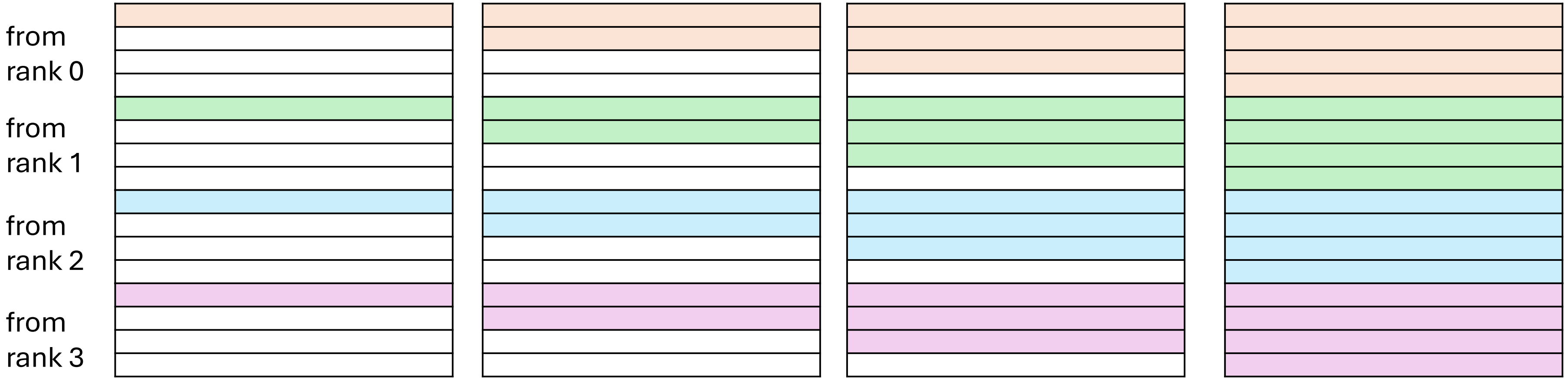
Step 4 compute with Step 3 data

**H800 NVLink Topology**




point-to-point
200 GB/s
(~170GB/s in
real system)

# Overlapping Computation with Swizzling Optimization



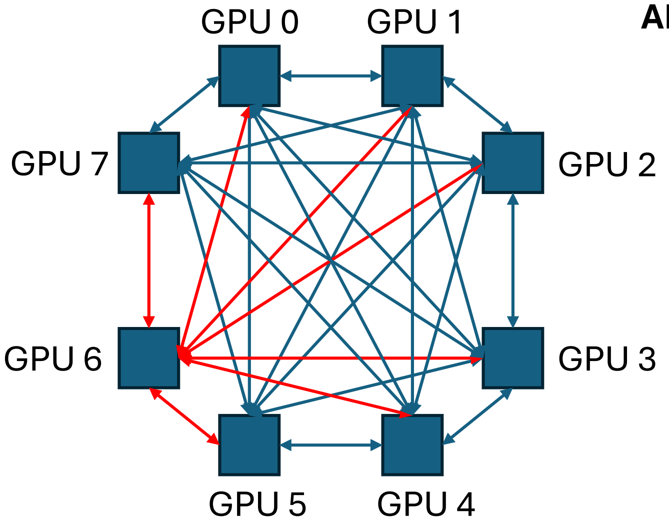

Step 1: AllGather 4 chunks and compute

Step 2: AllGather the next 4 chunks and compute

Step 3: AllGather the next 4 chunks and compute

Step 4: AllGather the final 4 chunks and compute

**AMD MI308X Topology**

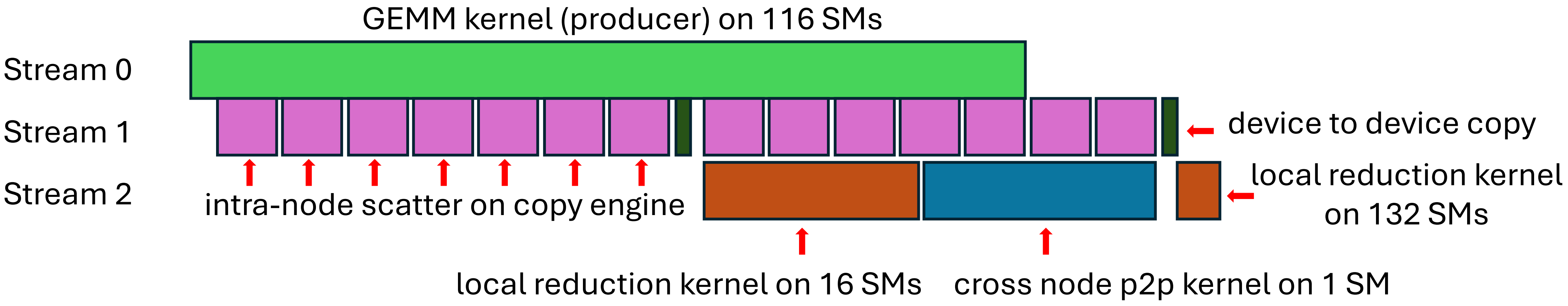



point-to-point
50 GB/s
Require all the 7
links to achieve
350GB/s

# Distributed Auto-Tuning and Resource Partitioning

◆**Distributed Auto-Tuning:** A novel auto-tuner designed specifically for distributed, overlapping kernels.

◆**Resource Partitioning:** A spatial optimization that maps tasks to different hardware units to balance load and prevent bottlenecks.



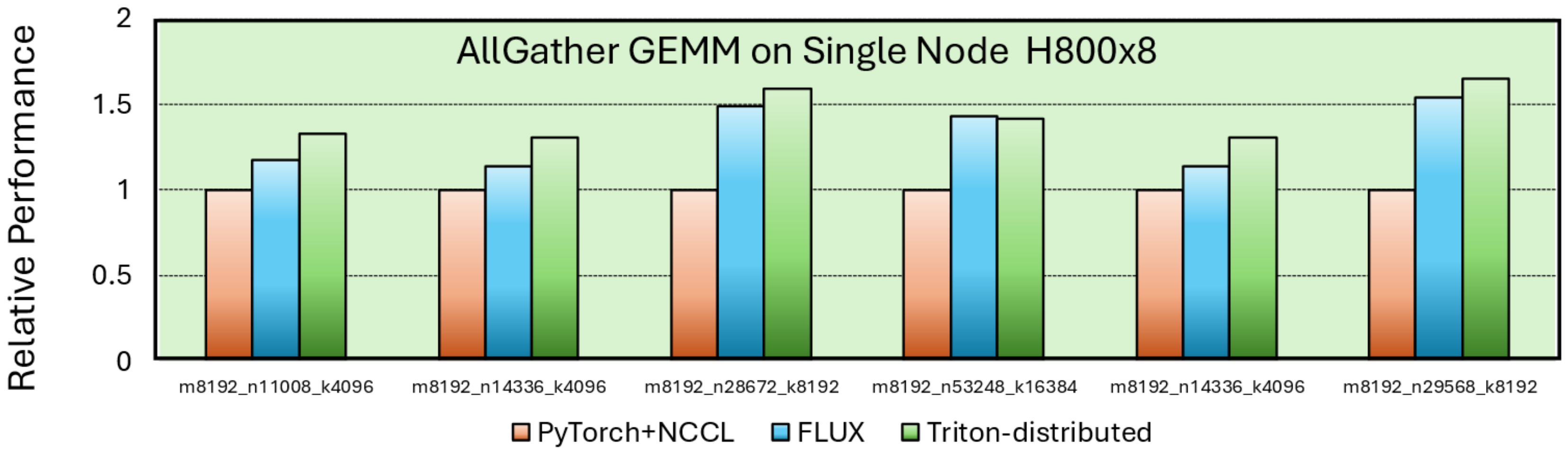
# Outline

- ◆ Background
- ◆ The Triton-distributed Architecture & Programming Model
- ◆ Overlapping Optimizations in Triton-distributed
- ◆ **Experiments & Evaluations**
- ◆ Conclusion

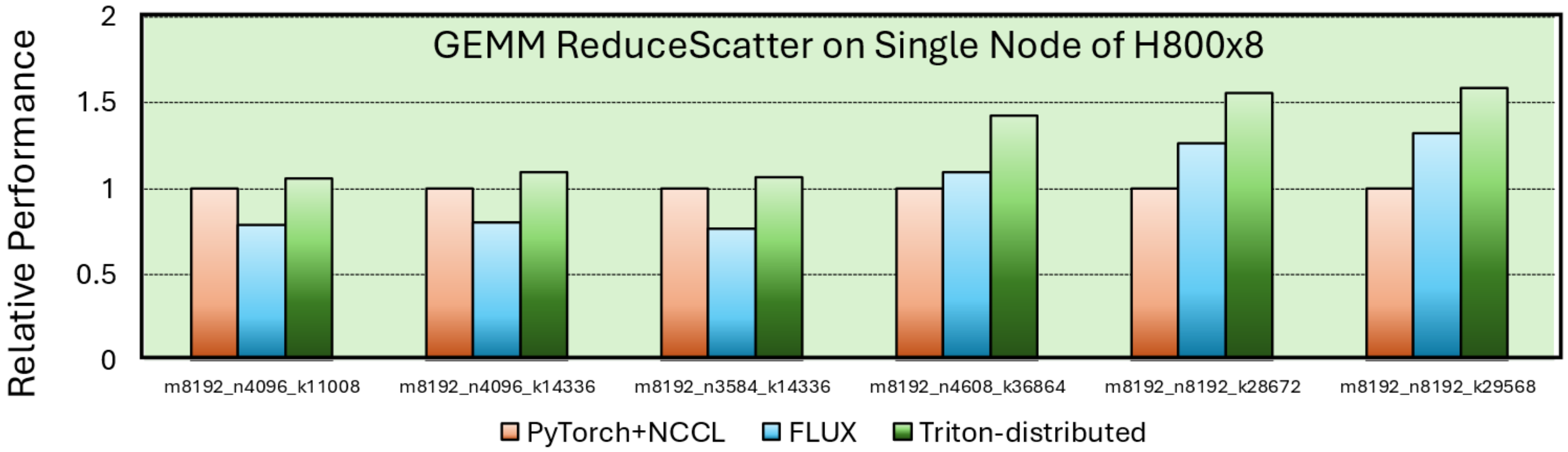
# List of Optimized Kernels

| Name | Explanation | Tested Hardware Cluster |
|---|---|---|
| AG+GEMM-intra | Intra-node AllGather GEMM Overlapping | 8 H800 and MI308X GPUs. |
| GEMM+RS-intra | Intra-node GEMM ReduceScatter Overlapping | 8 H800 and MI308X GPUs. |
| AG+MoE-intra | Intra-node AllGather MoE GroupGEMM Overlapping | 8 H800 GPUs |
| MoE+RS-intra | Intra-node MoE GroupGEMM ReduceScatter Overlapping | 8 H800 GPUs |
| FlashDecode+AG-intra | Intra-node Flash Decode AllGather and Combine | 8 H800 GPUs |
| AllToAll-intra | Intra-node Low-latency AllToAll | 8 H800 GPUs |
| AG+GEMM-inter | Inter-node AllGather GEMM Overlapping | 16 H800 GPUs |
| GEMM+RS-inter | Inter-node GEMM ReduceScatter Overlapping | 16 H800 GPUs |
| AG+MoE-inter | Inter-node AllGather MoE GroupGEMM Overlapping | 16 H800 GPUs |
| MoE+RS-inter | Inter-node MoE GroupGEMM ReduceScatter | 16 H800 GPUs |
| FlashDecode+AG-inter | Inter-node Flash Decode AllGather and Combine | 16 and 32 H800 GPUs |
| AllToAll-inter | Inter-node Low-latency AllToAll | 16, 32, and 64 H800 GPUs |

# Intra-node Kernel Performance on Nvidia GPUs

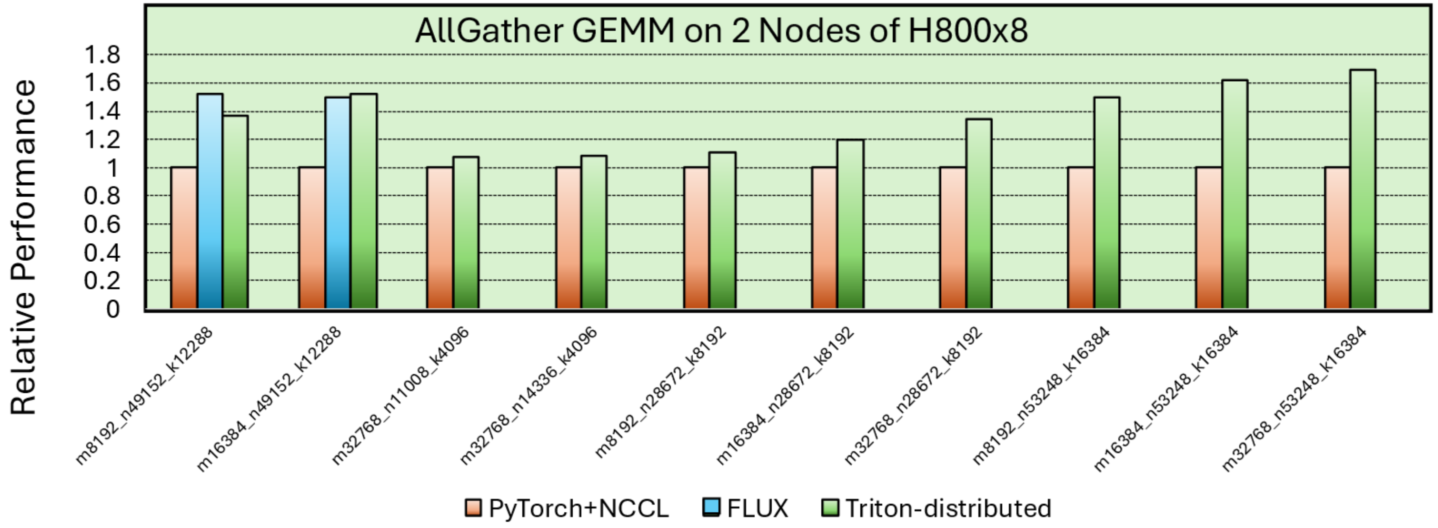



Performance of Intra-node AllGather GEMM on 8 H800 GPUs.

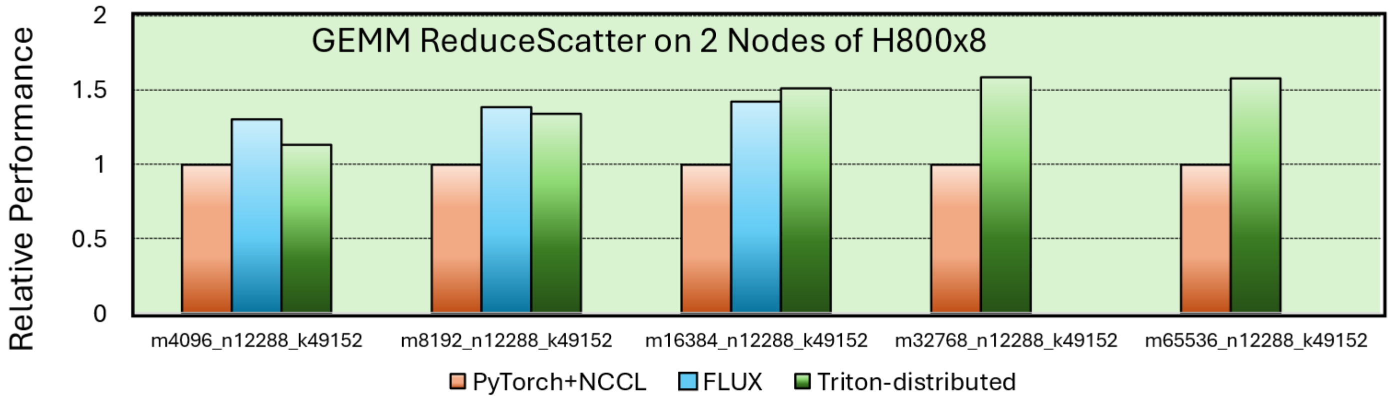



Performance of Intra-node GEMM ReduceScatter on 8 H800 GPUs.

# Inter-node Kernel Performance on Nvidia GPUs




Performance of Inter-node AllGather GEMM on 16 H800 GPUs.




Performance of Inter-node GEMM ReduceScatter on 16 H800 GPUs.

# MoE Performance on Nvidia GPUs

| Name | tokens/rank | in hidden | out hidden | experts | topk | Ours | | PyTorch | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | Intra | Inter | Intra | Inter |
| AG+MoE-1 | 256 | 2048 | 1408 | 60 | 4 | 0.33 | 0.45 | 23.95 | 28.84 |
| AG+MoE-2 | 512 | 2048 | 1408 | 60 | 4 | 0.40 | 1.37 | 26.25 | 29.77 |
| AG+MoE-3 | 1024 | 2048 | 1408 | 60 | 4 | 0.58 | 1.80 | 30.42 | 43.31 |
| AG+MoE-4 | 2048 | 2048 | 1408 | 60 | 4 | 0.97 | 3.07 | 55.63 | 63.73 |
| AG+MoE-5 | 256 | 14336 | 4096 | 8 | 2 | 0.54 | 1.01 | 7.05 | 19.92 |
| AG+MoE-6 | 512 | 14336 | 4096 | 8 | 2 | 0.72 | 1.89 | 26.34 | 36.07 |
| AG+MoE-7 | 1024 | 14336 | 4096 | 8 | 2 | 1.19 | 3.41 | 52.99 | 67.61 |
| AG+MoE-8 | 2048 | 14336 | 4096 | 8 | 2 | 2.10 | 6.51 | 107.32 | 129.30 |
| AG+MoE-9 | 256 | 16384 | 6144 | 8 | 2 | 0.81 | 1.39 | 11.02 | 27.29 |
| AG+MoE-10 | 512 | 16384 | 6144 | 8 | 2 | 1.06 | 2.21 | 39.65 | 52.32 |
| AG+MoE-11 | 1024 | 16384 | 6144 | 8 | 2 | 1.66 | 4.32 | 80.46 | 101.61 |
| AG+MoE-12 | 2048 | 16384 | 6144 | 8 | 2 | 2.92 | 8.28 | 159.69 | 192.67 |
| AG+MoE-13 | 512 | 1408 | 2048 | 64 | 6 | 0.45 | 0.84 | 29.25 | 38.17 |
| AG+MoE-14 | 1024 | 1408 | 2048 | 64 | 6 | 0.67 | 1.26 | 48.86 | 56.77 |
| AG+MoE-15 | 2048 | 1408 | 2048 | 64 | 6 | 1.18 | 2.18 | 74.26 | 90.44 |

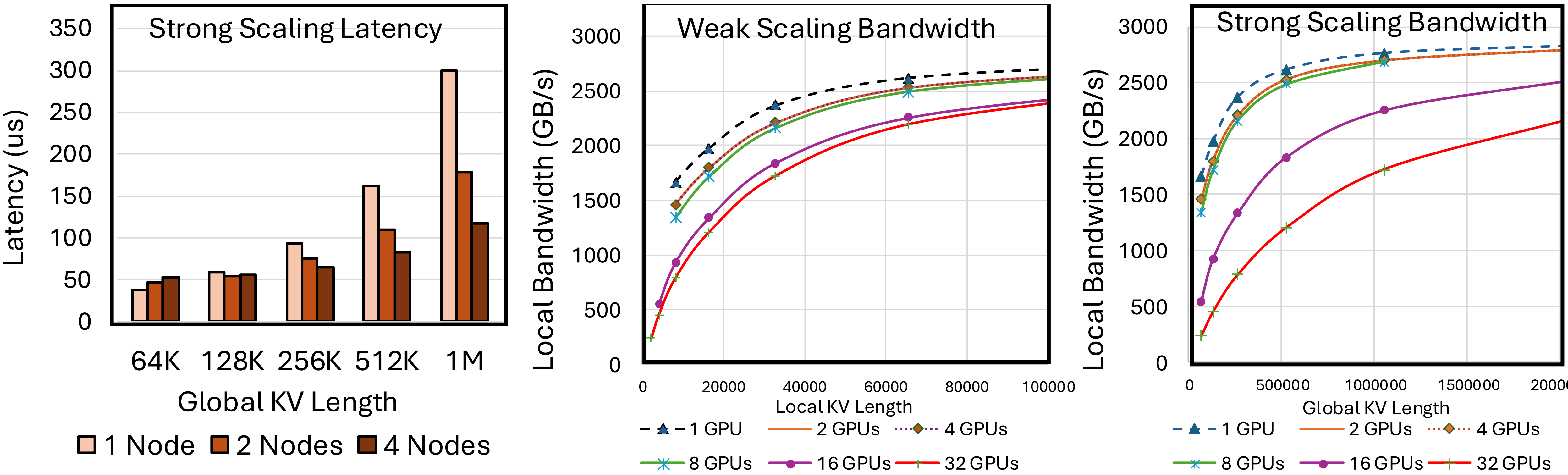
Test Shapes for AllGather MoE and Performance (ms).

# MoE Performance on Nvidia GPUs

| Name | tokens/rank | in hidden | out hidden | experts | topk | Ours | | PyTorch | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | Intra | Inter | Intra | Inter |
| MoE-RS-1 | 1024 | 1536 | 2048 | 8 | 2 | 0.51 | 3.62 | 4.35 | 12.41 |
| MoE-RS-2 | 1024 | 1536 | 2048 | 32 | 2 | 0.55 | 3.90 | 13.89 | 33.05 |
| MoE-RS-3 | 1024 | 1536 | 2048 | 64 | 2 | 0.67 | 4.82 | 27.91 | 61.70 |
| MoE-RS-4 | 1024 | 1536 | 2048 | 32 | 5 | 0.92 | 7.78 | 14.48 | 35.35 |
| MoE-RS-5 | 1024 | 1536 | 2048 | 64 | 5 | 0.93 | 8.25 | 29.96 | 64.88 |
| MoE-RS-6 | 1024 | 2048 | 4096 | 8 | 2 | 0.98 | 7.00 | 5.02 | 17.93 |
| MoE-RS-7 | 1024 | 2048 | 4096 | 32 | 2 | 1.08 | 7.86 | 14.12 | 38.24 |
| MoE-RS-8 | 1024 | 2048 | 4096 | 64 | 2 | 1.34 | 9.87 | 28.61 | 66.48 |
| MoE-RS-9 | 1024 | 2048 | 4096 | 32 | 5 | 1.84 | 15.51 | 16.70 | 44.37 |
| MoE-RS-10 | 1024 | 2048 | 4096 | 64 | 5 | 1.86 | 16.60 | 27.71 | 71.82 |

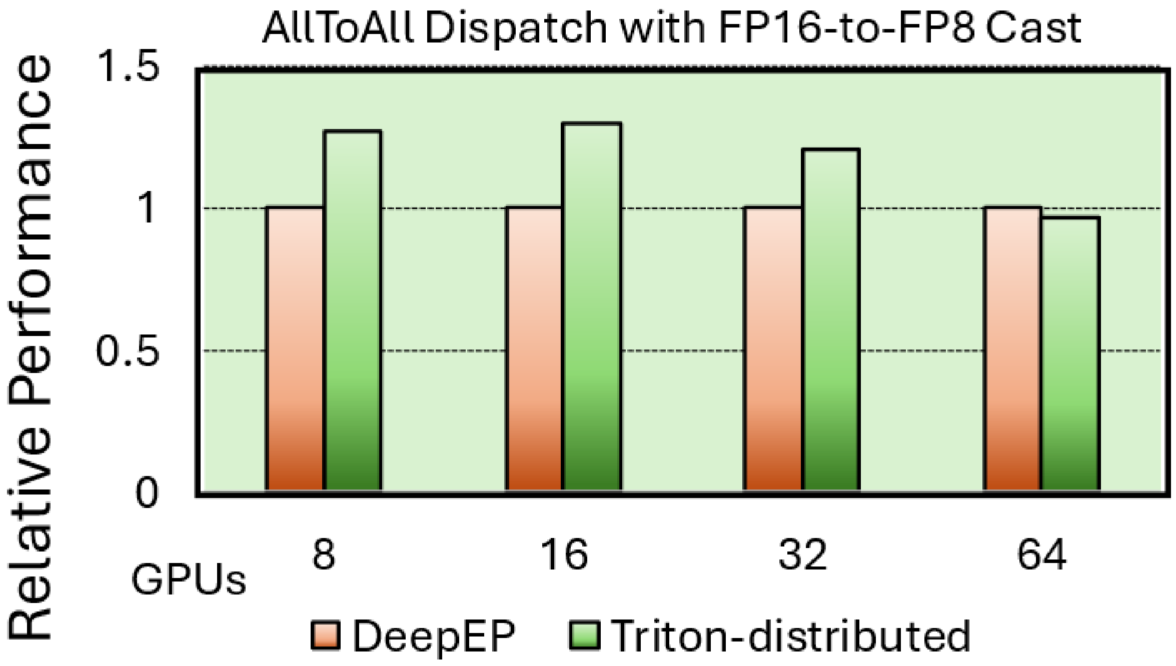
Test Shapes for MoE ReduceScatter and Performance (ms).

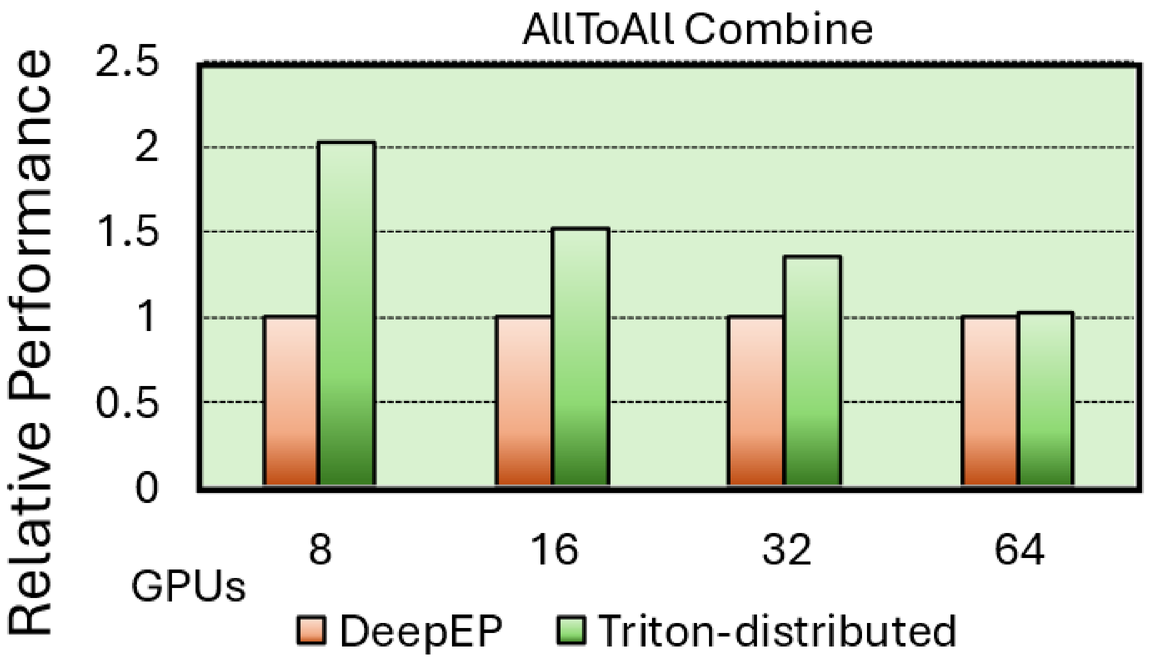
# Distributed Flash Decoding Performance




Performance of Distributed Flash Decoding.

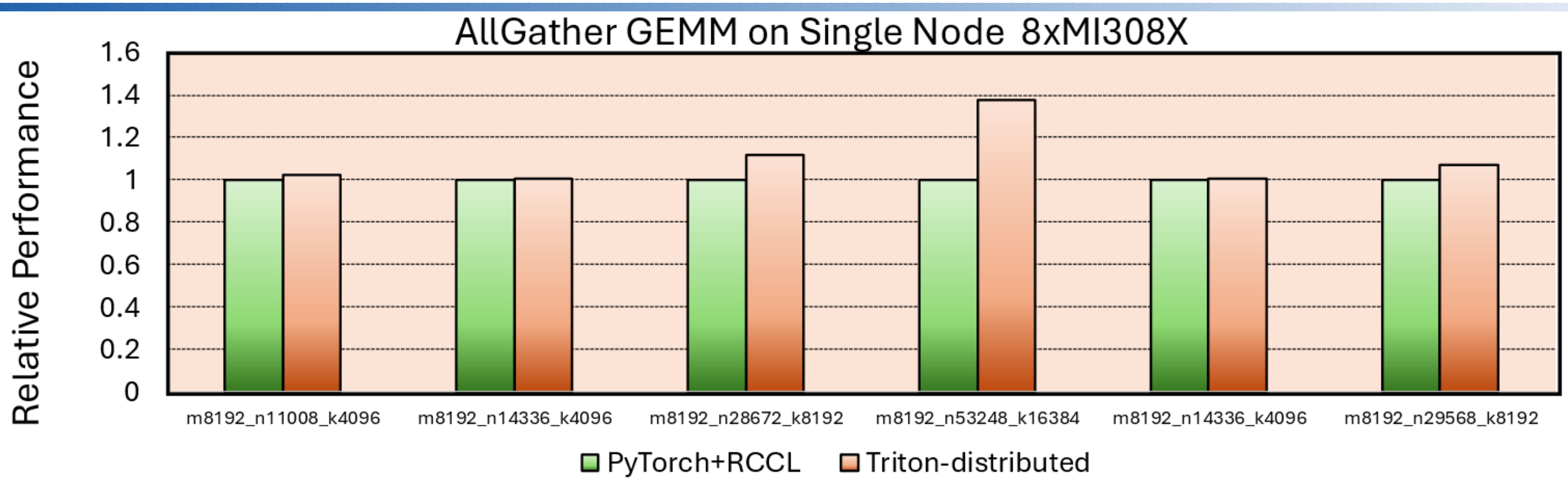
# Low Latency AllToAll Performance



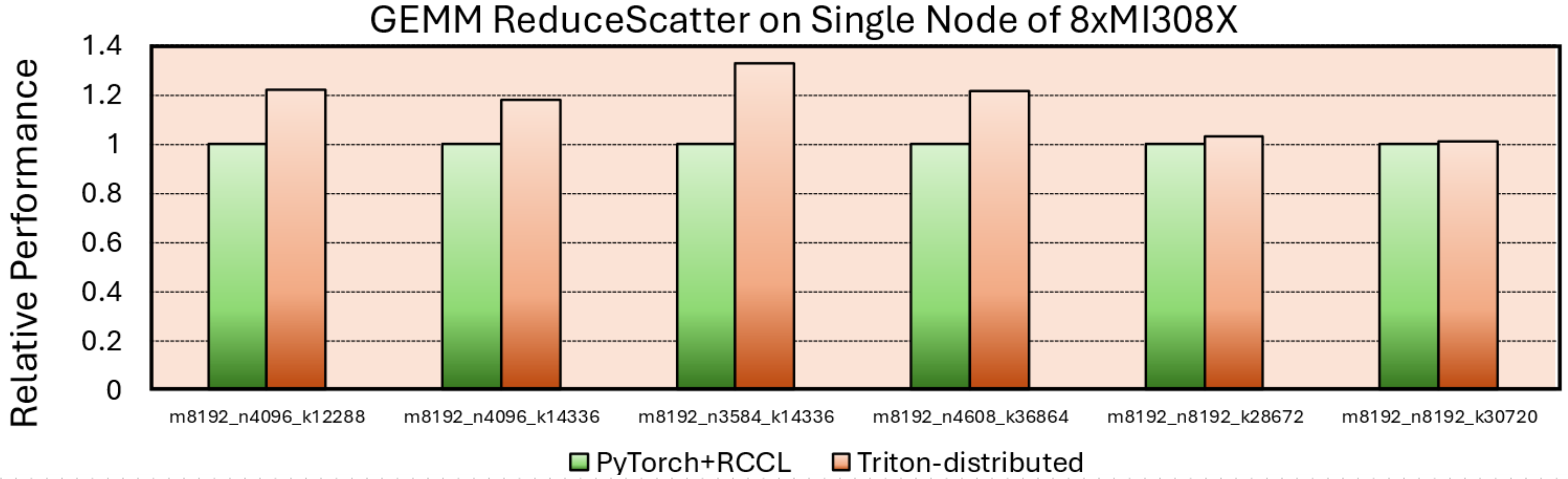




Performance of AllToAll

# Intra-node Kernel Performance on AMD GPUs




Performance of Intra-node AllGather GEMM on AMD GPUs.




Performance of Intra-node GEMM ReduceScatter on AMD GPUs.

# Outline

- ◆ Background
- ◆ The Triton-distributed Architecture & Programming Model
- ◆ Overlapping Optimizations in Triton-distributed
- ◆ Experiments & Evaluations
- ◆ **Conclusion**

# Conclusion

- Triton-distributed successfully **unifies distributed programming into Python,** drastically lowering the development barrier.

- The generated code achieves performance that is **competitive with, or superior to, hand-optimized low-level code**.

- The methodology is **portable across different hardware platforms**, demonstrating its general applicability.

# Thanks