CUDA MODE IRL

eric zhang, sage moore, clive chan

~~CUDA~~ NCCL MODE IRL

Idea

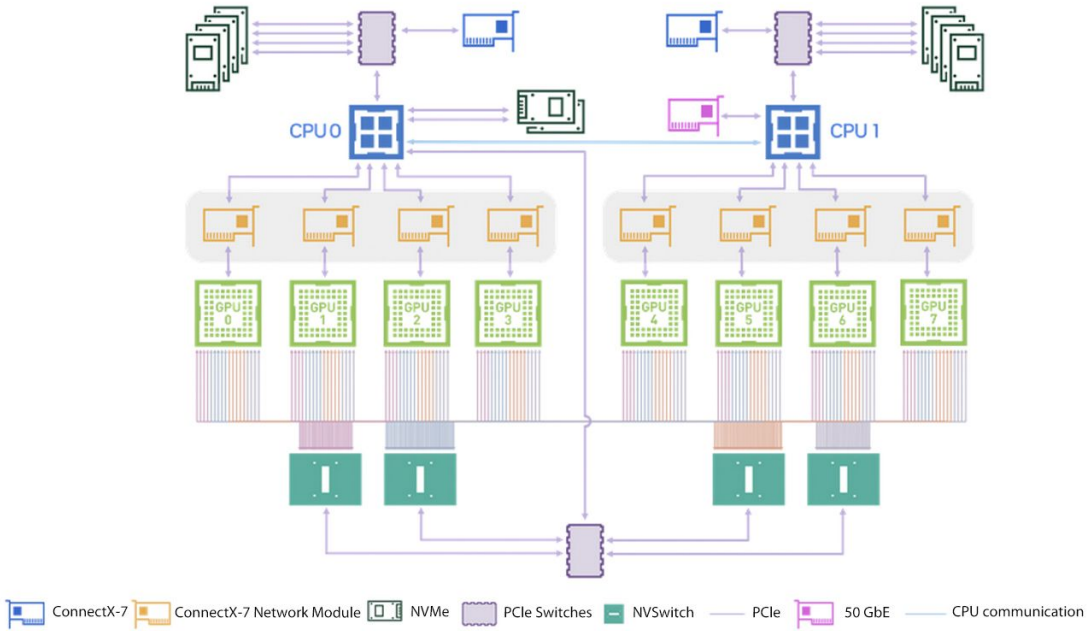
- There’s lots of compute kernels in Triton, C / C++, etc
  - Liger kernels, llm.c, …
- But what about collective communication kernels?

Why custom collectives?

- Allreduce is just another kernel!
- NCCL is not easily extensible
  - Fusing with other ops, like matmul-allreduce, to improve overlapping
  - Quantized communication for reduced comm time
- Triton kernels are easy to write, can we make that work?

HW setup

- 8xH100 intra-node only
- Thanks to Oracle!



NVLink

- DMA-based communication between GPUs
- Load store semantics
- Lots of fun and exciting new ways to screw up your memory ordering

8.7. Morally strong operations

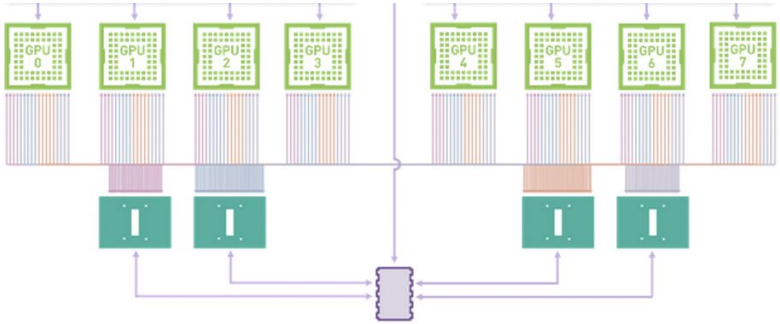
Two operations are said to be *morally strong* relative to each other if they satisfy all of the following conditions:

1. The operations are related in *program order* (i.e, they are both executed by the same thread), or each operation is *strong* and specifies a *scope* that includes the thread executing the other operation.
2. Both operations are performed via the same *proxy*.
3. If both are memory operations, then they overlap completely.

Most (but not all) of the axioms in the memory consistency model depend on relations between *morally strong* operations.

8.7.1. Conflict and Data-races

Two *overlapping* memory operations are said to *conflict* when at least one of them is a *write*



Symmetric Memory

- New feature in PyTorch 2.5 RC
- Sets up the memory maps with other GPUs in the same node
- So now we need to write the Triton kernels that perform the memory reads and writes

pytorch / pytorch

Type / to search

Issues 5k+ | Pull requests 1k | Actions | Projects 12 | Wiki | Security 1 | Insights

Introduce a prototype for SymmetricMemory #128582

Closed yifuwang wants to merge 11 commits into gh/yifuwang/92/base from gh/yifuwang/92/head

Conversation 23 | Commits 11 | Checks 175 | Files changed 16

yifuwang commented on Jun 12 • edited — Contributor

Stack from ghstack (oldest at bottom):

This PR introduces a prototype for `SymmetricMemory` (including a CUDA implementation) - a remote-memory access-based communication primitive. It allows for user-defined communication patterns/kernels and is designed to be torch.compile-friendly. It addresses the major limitations of `IntraNodeComm` and `ProcessGroupCudaP2p` and serves as a replacement for them.

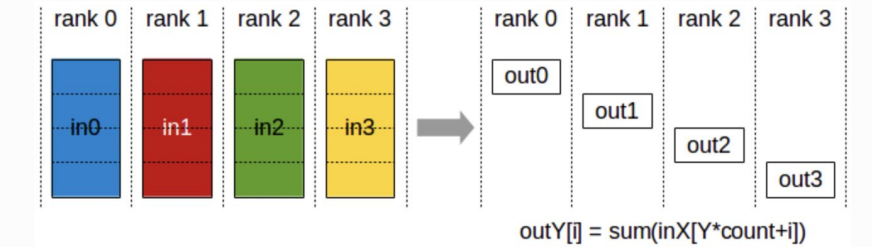
SymmetricMemory

`SymmetricMemory` represents symmetric allocations across a group of devices. The allocations represented by a `SymmetricMemory` object are accessible by all devices in the group. The class can be used for **op-level custom communication patterns** (via the get_buffer APIs and the synchronization primitives), as well as **custom communication kernels** (via the buffer and signal_pad device pointers).

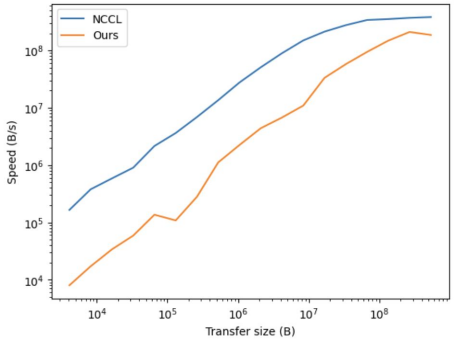
Custom CUDA Comm Kernels

Given a tensor, users can access the associated `SymmetricMemory` which provides pointer to remote buffers/signal_pads needed for custom communication kernels.

Triton reduce-scatter

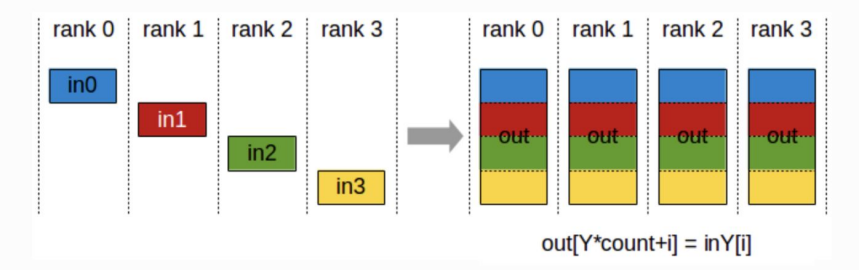


- Built on top of https://github.com/yifuwang/symm-mem-recipes
- Each GPU loads its input into symmetric memory
- Each GPU reads the relevant chunk (not everything) from every other GPU and reduces it and writes it out
- Accomplished using normal loads / stores!
- Verified correctness!!
- Slower than NCCL :(

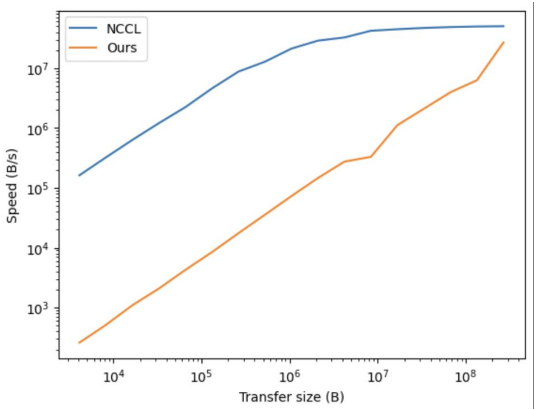



```python
blockwise_barrier(signal_pad_ptrs, None, rank, world_size)
pid = tl.program_id(axis=0)

per_rank_numel = numel // world_size

buffer_ptrs = buffer_ptrs.to(tl.pointer_type(tl.uint64))
output_ptr = output_ptr.to(tl.pointer_type(tl.uint64))
block_start = pid * BLOCK_SIZE

while block_start < (per_rank_numel // NUMEL_PER_THREAD):
    offsets = (block_start + tl.arange(0, BLOCK_SIZE)) * 2
    mask = block_start + tl.arange(0, BLOCK_SIZE) < per_rank_numel // NUMEL_PER_THREAD

    acc_hi = tl.zeros((BLOCK_SIZE,), tl.uint64)
    acc_lo = tl.zeros((BLOCK_SIZE,), tl.uint64)
    for i in range(world_size):
        buffer_ptr = tl.load(buffer_ptrs + i).to(tl.pointer_type(tl.uint64)) + rank * per_rank_numel // NUMEL_PER_THREAD * 2
        (hi, lo) = load_128(buffer_ptr + offsets, mask=mask)
        (acc_hi, acc_lo) = add_v8_bf16(acc_hi, acc_lo, hi, lo)

    tl.store(output_ptr + offsets + 0, acc_hi, mask=mask)
    tl.store(output_ptr + offsets + 1, acc_lo, mask=mask)
    block_start += tl.num_programs(axis=0) * BLOCK_SIZE
```

Triton all-gather



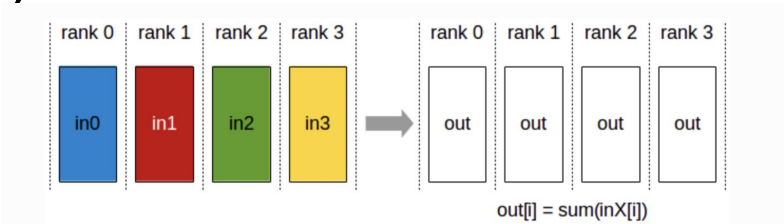
- Built on top of https://github.com/yifuwang/symm-mem-recipes
- Each GPU loads its input into symmetric memory
- Each GPU reads out every single input block
- Accomplished using normal loads / stores!
- Verified correctness, can be combined with reduce-scatter for all-reduce
- Much slower than NCCL... probably a bug (or Triton overhead…?)
-




```python
blockwise_barrier(signal_pad_ptrs, None, rank, world_size)
pid = tl.program_id(axis=0)

buffer_ptrs = buffer_ptrs.to(tl.pointer_type(tl.uint64))
output_ptr = output_ptr.to(tl.pointer_type(tl.uint64))
block_start = pid * BLOCK_SIZE

while block_start < (numel // NUMEL_PER_THREAD):
    offsets = (block_start + tl.arange(0, BLOCK_SIZE)) * 2
    mask = block_start + tl.arange(0, BLOCK_SIZE) < numel // NUMEL_PER_THREAD

    for i in range(world_size):
        buffer_ptr = tl.load(buffer_ptrs + i).to(tl.pointer_type(tl.uint64))
        (hi, lo) = load_128(buffer_ptr + offsets, mask=mask)

        scale_factor_for_uint64_ptr = tl.uint64.primitive_bitwidth // tl.bfloat16.primitive_bitwidth

        tl.store(output_ptr + i * numel // scale_factor_for_uint64_ptr + offsets + 0, hi, mask=mask)
        tl.store(output_ptr + i * numel // scale_factor_for_uint64_ptr + offsets + 1, lo, mask=mask)
    block_start += tl.num_programs(axis=0) * BLOCK_SIZE
```

Triton all-reduce (double binary tree)

- Standard NCCL tree algorithm
  - Reduce up the tree
  - Broadcast down the tree
- Double binary tree construction works
  - Each GPU is a leaf in one tree and a non-leaf in the other
- In principle, this allows much lower bandwidth allreduce - only two sends and two receives per GPU
- Triton kernel segfaults :(
  - some llvm / triton bug...?




```
E.g. if N is 8, the trees will be constructed like:
            0
          /
        4
    /     \\
    2      6
   / \\   / \\
  1   3  5   7
     AND
            7
          /
        3
    /     \\
    1      5
   / \\   / \\
  0   2  4   6
```

Takeaways

Comms are just kernels!

Comms ==> memory models and synchronization ==> is hard

We didn’t beat NCCL but we did get correct results with triton!

Thank you!

Questions?