PR #27151 完整报告

sgl-project/sglang

[diffusion] Skip unused WanVAE halo send copies

合并时间：2026-06-04 10:23

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/27151

执行摘要

- 一句话：跳过边界 rank 的 WanVAE halo 发送副本
- 推荐动作：值得精读的实现级优化，展示了如何通过内存格式感知来避免分布式推理中的显式 / 隐式数据副本。_halo_memory_format 的检测模式可推广到其他分布式卷积 / 注意力模块。

功能与动机

在 WanVAE 分布式解码中，halo 交换操作在边界 rank（rank=0 或 rank=world_size-1）上仍会为发送创建 contiguouis 副本，但这些副本实际上不会被使用（因为边界 rank 没有对应的发送目标）。同时，使用 torch.empty_like 分配接收缓冲区可能不匹配输入张量的 channels_last 或 channels_last_3d 格式，导致后续 concat 触发昂贵的布局转换。

实现拆解

1. 新增 _halo_memory_format 函数（wan_dist_utils.py:106-112）：根据参考张量的维度和步幅动态推断最佳内存格式（channels_last_3d / channels_last / contiguous_format），用于分配接收缓冲区和控制发送张量的布局。
2. 改造 _ensure_recv_buf（wan_dist_utils.py:124-141）：将 torch.empty_like(reference) 替换为 torch.empty(..., memory_format=memory_format)，并增加 is_contiguous(memory_format) 检查，若现有缓存不匹配则重新分配，避免后续 concat 的性能损失。
3. 优化 halo_exchange 函数（wan_dist_utils.py:144-197）：将 top_row/bottom_row = x[...].contiguous() 分解为 top_row_ref/bottom_row_ref = x[...]（延迟切片），仅在非边界 rank 上发送时才调用 .contiguous(memory_format=...) 创建副本，跳过边界 rank 的无用发送副本。

关键文件：

- python/sglang/multimodal_gen/runtime/models/vaes/parallel/wan_dist_utils.py（模块 分布式基础；类别 source；类型 data-contract；符号 _halo_memory_format, _ensure_recv_buf, halo_exchange）：唯一修改文件，核心变更：新增 _halo_memory_format、重构 _ensure_recv_buf 和 halo_exchange 以支持内存格式感知的缓冲区分配与延迟副本创建。

关键符号：_halo_memory_format, _ensure_recv_buf, halo_exchange

关键源码片段

python/sglang/multimodal_gen/runtime/models/vaes/parallel/wan_dist_utils.py

唯一修改文件，核心变更：新增 _halo_memory_format、重构 _ensure_recv_buf 和 halo_exchange 以支持内存格式感知的缓冲区分配与延迟副本创建。

```
# wan_dist_utils.py 关键变更：内存格式感知的 halo 交换

def _halo_memory_format(reference: torch.Tensor) -> torch.memory_format:
    # 根据参考张量的内存布局，推断接收 / 发送缓冲区应使用的格式
    if reference.dim() > 1 and reference.stride(1) == 1: # 末尾维度连续 => 可能是 channels_last
        if reference.dim() == 5 and hasattr(torch, "channels_last_3d"):
            return torch.channels_last_3d
        if reference.dim() == 4:
            return torch.channels_last
    return torch.contiguous_format # 默认 contiguous


def _ensure_recv_buf(
    recv_buf: torch.Tensor | None, reference: torch.Tensor
) -> torch.Tensor:
    # 动态检测参考张量的内存格式，用于分配接收缓冲区
    memory_format = _halo_memory_format(reference)
    if (
        recv_buf is None
        or recv_buf.shape != reference.shape
        or recv_buf.dtype != reference.dtype
        or recv_buf.device != reference.device
        or not recv_buf.is_contiguous(memory_format=memory_format)
    ):
        # 用推断的格式创建空张量，避免后续 concat 隐式转换
        return torch.empty(
            reference.shape,
            dtype=reference.dtype,
            device=reference.device,
            memory_format=memory_format,
        )
    return recv_buf


def halo_exchange(
    x: torch.Tensor,
    height_halo_size: int = 1,
    recv_top_buf: torch.Tensor | None = None,
    recv_bottom_buf: torch.Tensor | None = None,
) -> tuple[torch.Tensor, torch.Tensor, torch.Tensor]:
    if height_halo_size == 0:
```

```
    return x, recv_top_buf, recv_bottom_buf
# ... 省略 group/rank 初始化 ...
# 延迟切片，避免立即创建 contiguous 副本
top_row_ref = x[..., :height_halo_size, :]
bottom_row_ref = x[..., -height_halo_size:, :]

recv_top_buf = _ensure_recv_buf(recv_top_buf, top_row_ref)
recv_bottom_buf = _ensure_recv_buf(recv_bottom_buf, bottom_row_ref)

p2p_ops = []
if rank > 0:
    # 非首边界：发送时创建与接收缓冲区同格式的 contiguous 副本
    prev_rank = group_ranks[rank - 1]
    top_row = top_row_ref.contiguous(memory_format=_halo_memory_format(top_row_ref))
    p2p_ops.append(dist.P2POp(dist.irecv, recv_top_buf, prev_rank, group))
    p2p_ops.append(dist.P2POp(dist.isend, top_row, prev_rank, group))
if rank < world_size - 1:
    next_rank = group_ranks[rank + 1]
    bottom_row = bottom_row_ref.contiguous(
        memory_format=_halo_memory_format(bottom_row_ref)
    )
    p2p_ops.append(dist.P2POp(dist.isend, bottom_row, next_rank, group))
    p2p_ops.append(dist.P2POp(dist.irecv, recv_bottom_buf, next_rank, group))
# 边界 rank 的接收缓冲区直接置零（无发送操作）
if rank == 0:
    recv_top_buf.zero_()
if rank == world_size - 1:
    recv_bottom_buf.zero_()
# ... 执行 batch P2P 并 concat ...
```

评论区精华

AI 审核机器人指出两个关键问题：

- 使用 torch.empty 默认分配 contiguous 缓冲区，若输入张量是 channels_last_3d 格式，后续 concat 会触发昂贵的布局转换。
- 发送张量 .contiguous() 默认转为 contiguous 格式，而接收缓冲区可能是 channels_last_3d 格式，导致 P2P 通信时数据损坏。 PR 作者在后续提交中新增 _halo_memory_format 函数动态检测内存格式，并在 _ensure_recv_buf 和发送前 contiguous 调用中传入该格式，解决了上述问题。
- 内存格式不一致导致性能 / 正确性问题 (performance): PR 作者通过新增 _halo_memory_format 函数，在分配接收缓冲区和发送 contiguous 时都传入推断的格式，保证布局一致。

风险与影响

- 风险：核心风险集中在 _halo_memory_format 函数对内存格式的推断逻辑：对于高维张量（dim>5）或非标准步幅的输入，返回 contiguous_format 可能并非最优，但功能正确。

P2P 通信和 concat 路径的布局现已同步，回归风险低。未同步修改其他可能调用 _ensure_recv_buf 或 halo_exchange 的模块（目前仅 WanVAE 使用）。

- 影响：影响范围限于 WanVAE 分布式解码路径（wan_dist_utils.py）。性能影响：边界 rank 节省两次 contiguous 副本创建；所有 rank 的接收缓冲区布局与输入一致，避免 concat 隐式转换。功能上保持输出 bit-exact 一致（PR 验证了 MP4 sha 匹配）。对用户透明，无配置变更。
- 风险标记：核心路径变更，缺少测试覆盖

关联脉络

- 暂无明显关联 PR