PR #27383 完整报告

sgl-project/sglang

[diffusion] Optimize LingBot realtime SP cache path

合并时间：2026-06-06 09:37

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/27383

执行摘要

- 一句话：优化 LingBot 实时 SP 缓存路径与 USP 通信
- 推荐动作：值得精读。该 PR 展示了如何通过细粒度的缓存复用和通信路径选择来优化实时推理管线的关键瓶颈，设计决策清晰，性能收益量化充分。建议关注 GPT reviewer 提出的进一步优化建议。

功能与动机

LingBot 实时推理在上下文缓存更新阶段，相机条件器的 scale/shift 张量已经在之前的去噪阶段计算完毕，但原有代码在上下文更新时未复用该缓存，导致重复计算。此外，在常见的 SP4 配置下序列令牌被均匀切分，使用 varlen USP 通信原语会引入不必要的 split-size 计算和 chunk 打包开销。

实现拆解

1. 复用相机条件器缓存：在 lingbot_world.py 的 _cam_conditioner_scale_shift 方法中，去除了 if forward_context.current_timestep < 0: return self.cam_conditioner.compute_scale_shift(...) 的早期返回分支，使上下文更新阶段也能进入缓存逻辑（通过 forward_batch.extra 中的 lingbot_cam_conditioner 键）。
2. 新增均匀分片检测函数：新增 _sequence_splits_are_uniform 工具函数，用于判断序列分片列表是否所有 rank 的长度一致（即均匀切分）。
3. 优化 USP 通信路径：在 _sequence_all_gather_varlen 中增加均匀分片的 fast path，直接调用 sequence_model_parallel_all_gather；在 CausalLingBotWorldTransformerBlock.forward 的 QKV 输入 all-to-all 和输出 all-to-all 处，根据 uniform_seq_splits 标志选择固定大小的 _usp_input_all_to_all/_usp_output_all_to_all 而非 varlen 版本。
4. 更新测试：将原有测试函数 test_lingbot_cam_conditioner_cache_skips_context_update 重命名为 test_lingbot_cam_conditioner_cache_reuses_context_update，并调整断言以验证缓存复用行为（第一次与第二次返回同一对象，conditioner 仅被调用一次，且 forward_batch.extra 中包含缓存键）。

关键文件：

- python/sglang/multimodal_gen/runtime/models/dits/lingbot_world.py（模块 扩散模型；类别 source；类型 core-logic；符号 _sequence_splits_are_uniform）：核心实现文件，

包含相机条件器缓存复用逻辑、均匀分片检测函数以及 USP 通信路径选择的关键改动。

- python/sglang/multimodal_gen/test/unit/realtime/test_lingbot_causal_denoising.py（模块 扩散模型；类别 test；类型 test-coverage；符号 test_lingbot_cam_conditioner_cache_skips_context_update, test_lingbot_cam_conditioner_cache_reuses_context_update）：测试文件，验证相机条件器缓存复用行为，确保代码正确性。

关键符号：_sequence_splits_are_uniform, _sequence_all_gather_varlen, _cam_conditioner_scale_shift, forward

关键源码片段

python/sglang/multimodal_gen/runtime/models/dits/lingbot_world.py

核心实现文件，包含相机条件器缓存复用逻辑、均匀分片检测函数以及 USP 通信路径选择的关键改动。

```
# python/sglang/multimodal_gen/runtime/models/dits/lingbot_world.py

def _sequence_splits_are_uniform(seq_splits: list[int]) -> bool:
    """检查序列分片是否在所有 rank 上长度一致（均匀切分）。
    如果均匀，可以直接用更高效的固定大小 USP 通信，避免 varlen 的额外开销。
    """
    return len(seq_splits) <= 1 or all(
        seq_len == seq_splits[0] for seq_len in seq_splits
    )


def _sequence_all_gather_varlen(
    x: torch.Tensor,
    seq_splits: list[int],
    group: dist.ProcessGroup,
) -> torch.Tensor:
    rank = get_sp_parallel_rank()
    # 如果分片均匀，直接使用固定大小的 all_gather，无需 padding 和分片拼接
    if _sequence_splits_are_uniform(seq_splits):
        return sequence_model_parallel_all_gather(x.contiguous(), dim=1)

    # 原 varlen 路径：需要 padding 到最大长度，再 all_gather 后截断
    max_seq = max(seq_splits)
    local_seq = seq_splits[rank]
    if local_seq < max_seq:
        pad_shape = list(x.shape)
        pad_shape[1] = max_seq - local_seq
        pad = torch.zeros(pad_shape, dtype=x.dtype, device=x.device)
        x = torch.cat([x, pad], dim=1)
    gathered = [torch.empty_like(x) for _ in seq_splits]
    dist.all_gather(gathered, x.contiguous(), group=group)
    return torch.cat(
        [chunk[:, :seq_len, ...] for chunk, seq_len in zip(gathered, seq_splits)], dim=1
```

```
    )

# 在 CausalLingBotWorldTransformerBlock.forward 中，根据均匀标志选择 USP 通信路径
seq_splits = None
uniform_seq_splits = False # 新增标志
sequence_shard_enabled = (...)
if sequence_shard_enabled:
    seq_splits = list(seq_splits)
    uniform_seq_splits = _sequence_splits_are_uniform(seq_splits) # 检测均匀性

# 对于 QKV 输入 all-to-all，均匀时使用固定大小路径
qkv = torch.cat([roped_query, roped_key, v], dim=-1)
qkv = (
    _usp_input_all_to_all(qkv, head_dim=2) # 更高效的固定大小版本
    if uniform_seq_splits
    else _usp_input_all_to_all_varlen(qkv, seq_splits, head_dim=2) # 原 varlen 版本
)

# 对于输出 all-to-all，同样根据均匀性选择
x = (
    _usp_output_all_to_all(x, head_dim=2)
    if uniform_seq_splits
    else _usp_output_all_to_all_varlen(x, seq_splits, head_dim=2)
)

# 相机条件器 scale/shift 缓存复用：移除早期返回，让上下文更新也能缓存
def _cam_conditioner_scale_shift(
    self,
    c2ws_plucker_emb: torch.Tensor,
) -> tuple[torch.Tensor, torch.Tensor] | None:
    # ... 省略前面的 None 检查
    forward_context = get_forward_context()
    # 删除以下早期返回：
    # if forward_context.current_timestep < 0:
    # return self.cam_conditioner.compute_scale_shift(c2ws_plucker_emb)

    forward_batch = forward_context.forward_batch
    # 后续逻辑正常走缓存路径（通过 should_cache_cam_conditioner 判断是否缓存）
    if not CausalLingBotWorldTransformer3DModel._should_cache_cam_conditioner(
        forward_batch
    ):
        return self.cam_conditioner.compute_scale_shift(c2ws_plucker_emb)
    # ... 缓存读写逻辑
```

python/sglang/multimodal_gen/test/unit/realtime/test_lingbot_causal_denoising.py

测试文件，验证相机条件器缓存复用行为，确保代码正确性。

```
# python/sglang/multimodal_gen/test/unit/realtime/test_lingbot_causal_denoising.py
```

```
def test_lingbot_cam_conditioner_cache_reuses_context_update(monkeypatch):
    class _CamConditioner:
        def __init__(self):
            self.calls = 0

        def compute_scale_shift(self, c2ws_plucker_emb):
            self.calls += 1
            return c2ws_plucker_emb + 1, c2ws_plucker_emb + 2

    block = CausalLingBotWorldTransformerBlock.__new__(
        CausalLingBotWorldTransformerBlock
    )
    block.cam_conditioner = _CamConditioner()
    forward_batch = SimpleNamespace(extra={}, enable_sequence_shard=True)
    # 设置 ulysses 并行度为 2，模拟均匀分片（序列长度整除）
    monkeypatch.setattr(
        lingbot_world_module, "get_ulysses_parallel_world_size", lambda: 2
    )
    monkeypatch.setattr(
        lingbot_world_module,
        "get_forward_context",
        lambda: SimpleNamespace(forward_batch=forward_batch, current_timestep=-1),
    )

    c2ws_plucker_emb = torch.ones(1, 2, 3)
    first = block._cam_conditioner_scale_shift(c2ws_plucker_emb)
    second = block._cam_conditioner_scale_shift(c2ws_plucker_emb)

    # 断言缓存被复用：两次返回同一对象
    assert first is second
    # conditioner 只被调用了一次（第一次计算）
    assert block.cam_conditioner.calls == 1
    # extra 中包含缓存键
    assert "lingbot_cam_conditioner" in forward_batch.extra
```

评论区精华

Gemini Code Assist 的 review 指出，在 `_cam_conditioner_scale_shift` 中移除 `current_timestep < 0` 检查后，`_prepare_cam_conditioner_scale_shifts` 方法中仍有类似的早期返回（`if forward_context.current_timestep < 0: return None`），建议一并移除，以便在一次传递中为所有块急切检索并复用缓存的 scale/shift。

- 移除 _prepare_cam_conditioner_scale_shifts 中的早期返回以完整实现缓存复用 (design): 未明确回应，但当前 PR 已合并。后续可考虑采纳建议进一步优化。

风险与影响

- 风险：风险较低。主要变更集中在特定模型（LingBot）的特定路径（实时推理、序列分片均匀），且通过基准测试验证了性能提升。需要注意：

- 当分片不均匀时，仍回退到 varlen 路径，行为不变。
- 测试覆盖了均匀分片下的缓存复用，但未包含不均匀分片场景的回归测试。
- 建议在 _prepare_cam_conditioner_scale_shifts 中也移除早期返回，以完整实现缓存复用。
- 影响：对用户：LingBot 实时推理的延迟显著降低（请求总耗时降低约 2.4%）。不涉及公共 API 或配置变更，无 breaking change。 对系统：改动仅影响 LingBot 模型和其单元测试，模块隔离良好。 对团队：为 Diffusion 模型路径的性能优化提供了模式参考（缓存复用、均匀分片 fast path）。

- 风险标记：缺少不均匀分片测试

关联脉络

- PR #27096 [diffusion] Cosmos3 fused qknorm rope: 同为 Diffusion 模型性能优化，涉及注意力层和 JIT 内核，体现了该系列的持续性能改进方向。