PR #35021 完整报告

sgl-project/sglang

[NPU] add causal conv1d for ascend kda backend

合并时间：2026-08-29 11:39

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/35021

执行摘要

- 一句话：NPU KDA 改用 causal_conv1d 原位回滚方案，推理提速约 16%
- 推荐动作：值得精读。该 PR 展示了硬件后端性能优化中一个典型设计权衡：用 " 原位写入 + 回滚 " 替代 " 中间快照 + 散射 "，并通过统一内存布局换取算子复用。重点关注 _get_conv_weights_t 的 dtype 缓存设计（首次 dtype 固定缓存的隐患）以及 conv_states_shape 转置与共享 KDA 后端的耦合点。建议后续补充 conv state 回滚正确性单测和不同 dtype 调用覆盖。

功能与动机

PR body 明确说明：现有的 causal_conv1d_linear_verify_npu + speculative_state_scatter_npu 机制在 kda spec branch 上性能差，因为它要求使用 intermediate_conv_window，引入更多读 / 写操作；作者提议用 causal_conv1d + conv_state_rollback 直接对 conv state 缓冲区做原位操作，同时用 CANN ascendc 的 causal_conv1d 替换 prefill/decode 分支现有的 torch native 或 triton 实现，并与 ascend gdn backend 的 conv stage 对齐。

实现拆解

实现分为 5 个步骤：

1. 统一 NPU conv state 内存布局（python/sglang/srt/hardware_backend/npu/memory_pool_npu.py 的 _init_npu_conv_state）：将 KDA 与 Mamba/GDN 统一为 [layers, pool, window, channels] 布局；对 is_kda=True 按 (window, channels) 轴序解析，并把窗口长度扩展 speculative_num_draft_tokens - 1，使 verify 阶段能把全部 draft token 的 conv state 顺序写入池中，为 conv_state_rollback 原位回滚做准备。原实现把 KDA 视为 [channels, window] 且 spec 窗口单独放在 intermediate cache。
2. 调整后端池形状暴露（python/sglang/srt/hardware_backend/npu/attention/ascend_kda_backend.py 的 __init__）：将 conv_states_shape 转置为 [layers, pool, window, channels]，同时把 supports_speculative_conv_state_snapshots 从 True 改为 False，使共享 KDA 后端的 _init_track_conv_indices 按 conv_states_shape[-1] 读取 conv 窗口长度，并关闭旧的快照散射机制。
3. 新增权重转置缓存 _get_conv_weights_t：CANN 的 causal_conv1d 要求权重为 [width, dim] 且 dtype 与输入一致；KDA 的 conv_weights 保持 FP32，而输入 /conv_states 是

BF16，因此首次调用时把 layer.conv_weights 转置并 cast 到调用者 dtype，缓存到 layer._conv_weights_t 上避免重复计算。

4. 替换内核调用路径：forward_decode 与 forward_extend 用 torch.ops.npu.causal_conv1d 替换 causal_conv1d_update_npu / causal_conv1d_fn_npu / causal_conv1d_linear_verify_npu；spec verify 后依赖 conv_state_rollback 原位回滚未接受的 token；同时修复 update_mamba_state_after_mtp_verify 中 host bound 造成的同步开销（对应 commit "fix host bound"）。
5. 验证与合入：未新增单测文件；PR body 提供 GPQA 准确率对比与 TTFT/TPOT benchmark 数据，CI 经多次 /rerun-failed-ci 后由 sglang-npu-bot 合入。

性能数据（8k in / 1k out / EP=64 / DSPARK=7）：

指标	Before	After	提升
TTFT (ms)	7364.2	6147.8	-16.5%
TPOT (ms)	15.05	12.68	-15.7%

关键文件：

- python/sglang/srt/hardware_backend/npu/attention/ascend_kda_backend.py（模块 注意力后端；类别 source；类型 core-logic；符号 _get_conv_weights_t, _causal_conv1d_extend, forward_decode, forward_extend）：核心后端文件，完成 KDA conv state 布局转置、spec 快照机制关闭、权重转置缓存与 decode/prefill 内核替换的全部核心逻辑。
- python/sglang/srt/hardware_backend/npu/memory_pool_npu.py（模块 内存池；类别 source；类型 core-logic；符号 _init_npu_conv_state）：NPU conv state 池布局统一的关键文件，_init_npu_conv_state 按 is_kda 解析轴序并扩展 spec 窗口，是 rollback 方案能够原位写入的前提。

关键符号：_get_conv_weights_t, _causal_conv1d_extend, forward_decode, forward_extend, _init_npu_conv_state

关键源码片段

python/sglang/srt/hardware_backend/npu/attention/ascend_kda_backend.py

核心后端文件，完成 KDA conv state 布局转置、spec 快照机制关闭、权重转置缓存与 decode/prefill 内核替换的全部核心逻辑。

```
# ascend_kda_backend.py —— Ascend KDA 后端关键片段（整理自 head 版本）
# 背景：NPU 的 causal_conv1d 走 CANN 算子，conv state 需要 GDN 风格的
# [layers, pool, window, channels] 布局（与共享 KDA 后端的 [channels, window]
# 互为转置）。这里转置池形状后暴露给 _init_track_conv_indices，让后者用
# conv_states_shape[-1] 读取 conv 窗口长度，同时关闭旧的快照散射机制。

def __init__(self, model_runner):
```

```
        super().__init__(model_runner)
        # NPU 池实际分配为 [layers, pool, window, channels]，此处转置 shape
        # 以便共享后端代码按 [layers, pool, channels, window] 的语义解析长度。
        conv_pool_shape = model_runner.req_to_token_pool.mamba_pool.mamba_cache.conv[0].
        shape
        self.conv_states_shape = torch.Size(
            (
                *conv_pool_shape[:-2],
                conv_pool_shape[-1],
                conv_pool_shape[-2],
            )
        )
        self.kernel_dispatcher.extend_kernel = _AscendKDAExtendKernel()

    # 权重转置缓存：CANN causal_conv1d 要求权重为 [width, dim] 且与输入同 dtype。
    # KDA 的 conv_weights 保持 FP32，而输入 /conv_states 是 BF16，所以按首次
    # 调用时的 dtype 缓存转置结果。注意：若后续不同分支以不同 dtype 调用，
    # 缓存可能不匹配，建议补充 dtype 断言。

    def _get_conv_weights_t(self, layer: RadixLinearAttention, dtype: torch.dtype) -> torch.Tensor:
        w = getattr(layer, "_conv_weights_t", None)
        if w is None:
            w = layer.conv_weights.transpose(0, 1).contiguous().to(dtype)
            layer._conv_weights_t = w
        return w

    # decode 分支调用示意：用 CANN 算子替代 causal_conv1d_update_npu，
    # conv_states 原位读写，省去 intermediate_conv_window 快照 + scatter。
    qkv = torch.ops.npu.causal_conv1d(
        mixed_qkv.contiguous(),
        self._get_conv_weights_t(layer, mixed_qkv.dtype),
        # 其余参数按 CANN 语义传入 conv_states、bias、conv_state_indices 等
        ...
    )
```

python/sglang/srt/hardware_backend/npu/memory_pool_npu.py

NPU conv state 池布局统一的关键文件，_init_npu_conv_state 按 is_kda 解析轴序并扩展 spec 窗口，是 rollback 方案能够原位写入的前提。

```
    # memory_pool_npu.py —— NPU conv state 池布局统一（整理自 head 版本）
    # 目标：KDA 与 Mamba/GDN 统一为 [layers, pool, window, channels] 布局。
    # - KDA 传入的 conv_state_shape 为 (window, channels)
    # - Mamba/GDN 传入的为 (channels, window)
    # 窗口额外扩展 speculative_num_draft_tokens - 1，使 verify 能直接把所有
    # draft token 的 conv state 顺序写入池中，再配合 conv_state_rollback 原地
    # 回滚未接受 token，替代旧的 intermediate_conv_window 快照 + 散射方案。

    def _init_npu_conv_state(conv_state_in, conv_state_shape, speculative_num_draft_tokens=None,
    is_kda=False):
```

```
    extra_conv_len = 0
    if speculative_num_draft_tokens is not None:
        extra_conv_len = speculative_num_draft_tokens - 1

    conv_state = [
        torch.zeros(
            size=(
                conv_state_in.shape[0],
                conv_state_in.shape[1],
                (conv_shape[0] if is_kda else conv_shape[1]) + extra_conv_len,
                conv_shape[1] if is_kda else conv_shape[0],
            ),
            dtype=conv_state_in.dtype,
            device=conv_state_in.device,
        )
        for conv_shape in conv_state_shape
    ]
    return conv_state
```

评论区精华

Review 讨论以作者自查为主，没有外部 reviewer 交锋，核心澄清点是布局轴序与 API 注释：

- 在 forward_decode（line 190）与 forward_extend（line 294）处，作者先标记 "add note about activation."，随后分别回复 "done"，补齐了 causal_conv1d 中 silu 激活语义的代码注释（归属 documentation 类）。
- 在 memory_pool_npu.py 的 _init_npu_conv_state（line 46），作者先标记 "maybe need reshape here."，随后自我澄清 "both gdn and kda arrive as (window, channel) for causal conv1d"，确认 GDN 与 KDA 的 conv state 对 causal_conv1d 都是 (window, channel) 轴序，统一布局后无需额外 reshape（归属 question 类，是本次布局统一正确性的关键确认）。

所有线程均已 resolved，未发现遗留疑问；最终由 sglang-npu-bot 直接合入。

- decode/extend 分支补充 activation 说明注释 (documentation): 作者分别回复 "done"，已在代码中补齐激活相关注释。
- memory_pool 中 KDA 布局是否需要 reshape (question): 无需 reshape；KDA 与 GDN 统一为 [layers, pool, window, channels] 布局，是本次布局统一正确性的关键确认。

风险与影响

- 风险：风险点如下：

1. 布局轴序敏感（高）：ascend_kda_backend.py 的 conv_states_shape 改为交换最后两维，依赖共享 KDA 后端 _init_track_conv_indices 按 conv_states_shape[-1] 读取窗口长度；任何一端布局假设变化都会导致索引错位。同时 supports_speculative_conv_state_snapshots 翻转，旧快照 / 散射路径被关闭，若其他硬件 / 路径仍依赖该快照机制会受影响（当前该类仅 NPU 使用，风险有限）。

2. dtype 缓存易错（中）：_get_conv_weights_t 按首次调用 dtype 缓存转置权重到 layer._conv_weights_t，若不同分支以不同 dtype 调用（如 FP32 vs BF16），会静默返回错误 dtype 的权重；当前 KDA 路径输入为 BF16，风险中低，但建议加 dtype 断言。
3. 缺少测试覆盖（中高）：两个源码文件均无对应单测，spec 回滚正确性依赖线上 GPQA 准确率验证；若 rollback 窗口计算错误，可能产生静默精度劣化。
4. host bound 修复影响（中）："fix host bound" 改动涉及 update_mamba_state_after_mtp_verify 的同步语义，需关注 MTP verify 后 conv state 更新时序是否在其他配置下引入竞态。
5. 性能收益依赖配置（低）：benchmark 仅在 8k in / 1k out、EP=64、DSPARK=7 下测得，其他输入长度或 EP 组合的收益不一定同等显著。 - 影响：影响范围集中在 NPU 上的 KDA（Kimi Delta Attention）模型推理：

- 用户侧：spec decode 场景 TTFT 降低 16.5%、TPOT 降低 15.7%，decode/prefill 也有收益；GPQA 准确率 93.5 → 93.9，无精度回退。
- 系统侧：去掉 intermediate_conv_window 的中间读写，降低显存带宽压力；将 NPU conv state 布局统一为 GDN 风格，为 Mamba/GDN/KDA 后续共享 kernel 和算子复用铺路。
- 团队侧：删除一套专用 verify + scatter 内核路径，维护成本下降；但需要保持与共享 KDA 后端布局假设（[channels, window]）的同步，避免后续改动时回归。
- 风险标记：核心路径变更，缺少测试覆盖，布局轴序敏感，权重 dtype 缓存易错，无外部 review 确认

关联脉络

- PR #33576 [AMD] Add Work-Centric (Lean) Attention: a persistent-CTA decode kernel for long-context serving: 同为硬件后端专属 attention 性能优化，体现各后端（AMD/NPU）自研或选型专用 kernel 替代通用实现的演进主线，可作为对照参考。
- PR #35434 [CPU] Fix wrongly causal-masked bidirectional attention: 同为 attention/causal 掩码语义修复，提醒 causal conv1d 的窗口与掩码语义对模型正确性的影响，与本 PR 的 conv_state_rollback 正确性风险相关。
- PR #36934 [Fix] Drop the duplicated DSpark draft sample_block call: 同为 speculative decoding 路径的修复 / 优化，与 KDA spec branch 的 conv_state_rollback 同属 spec 解码优化线。