

PR #36821 完整报告

sgl-project/sglang

[KDA] Support ReplaySSM ring-write in the fused chain-verify kernel

合并时间: 2026-09-15 10:22

原文链接: <http://prhub.com.cn/sgl-project/sglang/pull/36821>

执行摘要

- 一句话: fused KDA verify 内核支持 ReplaySSM ring 写入, 解锁组合优化
- 推荐动作: 值得精读。核心看三点: 一是位级一致性的论证方式——为什么必须存 pre-mutation 的操作数、为什么 ring 值 (elementwise, 位于所有 tl.sum 上游) 能保证 bit-exact 而输出不能; 二是测试设计——共享垃圾模板做全张量比较以捕捉越界写, 以及独立 snapshot oracle 防止两个 ring 分支共享同一错误; 三是 dispatch 门控的数据驱动决策——用 sweep 实测数据决定启用范围, 而不是凭直觉放开。对于后续要做 batch-aware dispatch 或重调 KDA_VERIFY_BLOCK_V 的开发者, 这份 PR 提供了完整工具链。

功能与动机

PR body 明确指出此前两个机制互斥的现状: "The fused KDA chain-verify kernel (fused_kda_conv_gating_verify ...) and ReplaySSM spec-verify (--enable-linear-replayssm + spec ring) were mutually exclusive: _can_run_fused_chain_verify returned False whenever replayssm_rawv was set", 结果是 "We could have the ~1.3x fused verify kernel or the ReplaySSM memory win (no intermediate_ssm snapshot buffer, the dominant spec scratch), but not both". 此外 ring 模式还能移除 fused 内核的 dominant store: "the per-step [K, V] fp32 state snapshot (64 KB per head-step) is replaced by the ring records (raw v + raw k + gate + beta, ~1 KB per head-step), on a kernel that is latency-bound". Blackwell 上 CuTe DSL dspark MTP 内核已具备 ring-write + fused verify 组合, 但仅限 SM100 且形状契约固定为 Kimi-K3, 本 PR 把同一组合带给 Hopper 可用的 Triton fused chain-verify。

实现拆解

1. 内核层新增 `CACHE_RING` 写入路径 (`python/sglang/kernels/ops/attention/fla/fused_kda_conv_recurrent_verify.py`): `fused_kda_conv_gating_verify_kernel` 新增编译期常量 `CACHE_RING` 与 `replayssm_rawv/rawk/g/beta` 指针参数、`stride*_slot` 间距常量及 `MAX_CACHE_LEN`。写入点刻意落在 `b_beta` 形成之后、`l2norm` 修改 `b_k` 与 `delta-rule` 修改 `b_v` 之前, 与 unfused `fused_sigmoid_gating_recurrent.py` 的 `CACHE_RING` 块位置一致, 保证 commit 时 fold 重放能得到 bit-identical 的值; `rawk` 用 `is_qk_owner` 去重 (每 k-head 仅一个写入者), `g/beta` 在 `i_v == 0` 时每 v-head 写一次, `h0_idx` 复用为 ring slot, `t < MAX_CACHE_LEN` 防护 absorb-overflow 步骤越界。`CACHE_RING=False` 时这些代码是编译期死代码, 原路径不变。

2. wrapper 参数透传与校验: `fused_kda_conv_gating_verify` 采用与 unfused 内核完全相同的 `cache_ring/replayssm_*` 参数名, 使 backend 能用一个 `ring_kwargs` dict 同时透传给两条路径; wrapper 内对四类 ring 的 shape、dtype、packed inner stride 做断言, 并把 `stride(0)` 作为 slot 间距传入内核。
3. backend dispatch 改造 (`python/sglang/srt/layers/attention/linear/kda_backend.py`) : `ring_kwargs` 构造从 unfused 分支提前到 `_forward_target_verify` 开头, fused 与 unfused 两条路径共享, 避免参数不一致。 `_can_run_fused_chain_verify` 移除 `replayssm_rawv` 拦截; ReplaySSM 模式下跳过 `intermediate_state_cache` 检查 (该缓冲在这种模式下本为 `None`) , 改用新增的 `_replayssm_ring_ok` 静态方法校验 `ndim/shape/dtype/packed stride/L >= draft_token_num/device`, 不匹配则回落 unfused 路径。同时按实测性能把新路径门控在 `batch_size == 1` 且 `is_sm90 or is_sm100`。
4. 测试配套: `test/registered/kernels/test_fused_kda_conv_recurrent_verify.py` 新增 `test_replayssm_ring_matches_unfused`, 参数化为原 8 个用例加 3 个 GQA (HV != H) 用例; 两侧 ring 从同一份垃圾模板克隆, 输出、conv state、conv window 与四个 ring 全张量逐位比较, 垃圾初值能同时捕捉越界写与负 slot 写。新增 `test/registered/kernel/attention/test_kda_fused_verify_backend.py`, 走生产 backend 与 GPU kernel 做 verify -> commit -> verify 全流程: 部分 / 全部接受、多层、GQA、独立 snapshot oracle (避免两个 ring 分支共享同错而互检通过) , 并断言 conv rollback 与下一轮输出。
5. 性能基准工具 (`benchmark/kernels/bench_kda_verify_sweep.py` 新增) : 对 {unfused, fused} x {snapshots, ring} 四种可 dispatch 变体做 CUDA-graph 计时 (约 10 us 的内核, 裸 launch 只会测到 launch 开销) ; 内置 fused-vs-unfused ring 位级守卫 (`check_ring_bitwise`)、`--sweep-b`重调 `KDA_VERIFY_BLOCK_V`、`--hv-head`覆盖 GQA、`--lower-bound` 覆盖 safe/std gate, 且只 import `sglang.kernels.*`, 可在 srt 导入链不可用的机器上运行。

关键文件:

- `python/sglang/kernels/ops/attention/fla/fused_kda_conv_recurrent_verify.py` (模块 verify 内核; 类别 source; 类型 core-logic; 符号 `fused_kda_conv_gating_verify_kernel`, `fused_kda_conv_gating_verify`) : PR 的技术核心: Triton 内核新增 `CACHE_RING` 编译期分支与 `replayssm_*` 参数, 把 pre-mutation 操作数写入 per-slot ring 并保证与 unfused 内核位级一致; `CACHE_RING=False` 时行为完全不变。
- `python/sglang/srt/layers/attention/linear/kda_backend.py` (模块 KDA 后端; 类别 source; 类型 core-logic; 符号 `_forward_target_verify`, `_can_run_fused_chain_verify`, `_replayssm_ring_ok`) : backend dispatch 改造: `ring_kwargs` 提升为两条路径共享; `_can_run_fused_chain_verify` 移除对 `replayssm_rawv` 的拦截并按实测数据门控 `B=1` 且 `SM90/SM100`; 新增 `_replayssm_ring_ok` 做 ring 视图校验。
- `test/registered/kernels/test_fused_kda_conv_recurrent_verify.py` (模块 内核测试; 类别 test; 类型 test-coverage; 符号 `_make_ring_buffers`, `_run_reference`, `_run_fused`, `_compare_case`) : 核心正确性测试: `_RING_CASES` 在原有 8 用例上叠加 3 个 GQA 形状, 用共享垃圾模板做全张量位级比较, 可同时捕捉越界写、负 slot 写与 GQA head 索引错误。

- test/registered/kernel/attention/test_kda_fused_verify_backend.py (模块 后端测试; 类别 test; 类型 test-coverage; 符号 TestKDAFusedVerifyBackend, _make_case, _make_backend, _verify) : backend 级 verify -> commit -> verify 全流程测试: 部分 / 全部接受、多层、GQA、平台 override, 独立 snapshot oracle 防止两个 ring 分支共享同一错误而未察觉。
- benchmark/kernels/bench_kda_verify_sweep.py (模块 性能基准; 类别 source; 类型 dependency-wiring; 符号 make_inputs, _ring_kwargs, make_runner, bench_graph) : 四模式 sweep 基准: {unfused, fused} x {snapshots, ring} 的 CUDA-graph 计时、ring 位级守卫、BV 重扫与 GQA 覆盖, 是 PR 结论与后续调优的数据来源。

关键符号: fused_kda_conv_gating_verify_kernel, fused_kda_conv_gating_verify, _forward_target_verify, _can_run_fused_chain_verify, _replayssm_ring_ok, test_replayssm_ring_matches_unfused, test_verify_commit_verify, test_ring_dispatch_falls_back, _make_ring_buffers, bench_graph, check_ring_bitwise

关键源码片段

python/sglang/srt/layers/attention/linear/kda_backend.py

backend dispatch 改造: ring_kwargs 提升为两条路径共享; _can_run_fused_chain_verify 移除对 replayssm_rawv 的拦截并按实测数据门控 B=1 且 SM90/SM100; 新增 _replayssm_ring_ok 做 ring 视图校验。

```
# ReplaySSM 模式下 fused+ring 只放开到 B=1 且 SM90/SM100。
# 背景: 运行时仍用 BV=4 (按 snapshot 写流量调优的常数), 实测
# fused+ring 在 B=1 胜出 (H20-3e 1.27x、B300 1.49x), 但从 B=2 (T=8)
# 或 B=4 (T=4) 开始落后于 unfused+ring; sweep 显示最优 BV 会随 ring
# 模式变大 (16-32), 该重调尚未接入运行时。因此保守门控, snapshot
# 路径与 CuTe 路径不受影响。
if replayssm_on and (
    batch_size != 1 or not (get_platform().is_sm90 or get_platform().is_sm100)
):
    return False

# ... 常规形状 /dtype 检查通过后, ReplaySSM 分支改用 ring 视图校验。
if replayssm_on:
    if not self._replayssm_ring_ok(
        layer=layer,
        draft_token_num=draft_token_num,
        mixed_qkv=mixed_qkv,
        replayssm_rawv=replayssm_rawv,
        replayssm_rawk=replayssm_rawk,
        replayssm_g=replayssm_g,
        replayssm_beta=replayssm_beta,
    ):
        return False
elif intermediate_state_cache is None or intermediate_state_cache.dtype != torch.float32:
    return False
```

```

@staticmethod
def _replayssm_ring_ok(*, layer, draft_token_num, mixed_qkv,
                        replayssm_rawv, replayssm_rawk,
                        replayssm_g, replayssm_beta) -> bool:
    """校验 per-layer ReplaySSM ring 是否满足 fused ring-write 的布局要求。

    memory_pool.py 的 KDA spec ring 布局: rawv [slots, HV, L, V] 与
    rawk [slots, H, L, K] 使用 activation dtype, g [slots, HV, L, K]
    fp32 (per-K KDA gate), beta [slots, HV, L] fp32。内核以 stride(0)
    作为 slot 间距并假定 inner dim 连续; 不满足时回落 unfused 路径。
    """
    if replayssm_rawk is None or replayssm_g is None or replayssm_beta is None:
        return False
    if (replayssm_rawv.ndim != 4 or replayssm_rawk.ndim != 4
        or replayssm_g.ndim != 4 or replayssm_beta.ndim != 3):
        return False
    H, HV = layer.num_q_heads, layer.num_v_heads
    K, V = layer.head_k_dim, layer.head_v_dim
    ring_len = replayssm_rawv.shape[-2]
    if ring_len < draft_token_num:
        return False
    # 各 ring 的 head/dim 维度必须与 layer 配置一致
    if (tuple(replayssm_rawv.shape[1:]) != (HV, ring_len, V)
        or tuple(replayssm_rawk.shape[1:]) != (H, ring_len, K)
        or tuple(replayssm_g.shape[1:]) != (HV, ring_len, K)
        or tuple(replayssm_beta.shape[1:]) != (HV, ring_len)):
        return False
    # dtype 必须与内核写入类型一致: raw 走 activation dtype, gate/beta 走 fp32
    if (replayssm_rawv.dtype != mixed_qkv.dtype
        or replayssm_rawk.dtype != mixed_qkv.dtype
        or replayssm_g.dtype != torch.float32
        or replayssm_beta.dtype != torch.float32):
        return False
    # inner dims 必须连续打包, 内核按连续地址直接偏移
    if (replayssm_rawv.stride()[1:] != (ring_len * V, V, 1)
        or replayssm_rawk.stride()[1:] != (ring_len * K, K, 1)
        or replayssm_g.stride()[1:] != (ring_len * K, K, 1)
        or replayssm_beta.stride()[1:] != (ring_len, 1)):
        return False
    return True

```

test/registered/kernels/test_fused_kda_conv_recurrent_verify.py

核心正确性测试: `_RING_CASES` 在原有 8 用例上叠加 3 个 GQA 形状, 用共享垃圾模板做全张量位级比较, 可同时捕捉越界写、负 slot 写与 GQA head 索引错误。

```

# 垃圾初值模板: 两侧 ring 从同一份 randn 克隆, 因此全张量 bitwise 比较
# 既能证明写入位置一致, 也能证明两侧都没有越界写或负 slot 写——任何
# 意外触碰都会让该位置的垃圾值产生差异。
def _make_ring_buffers(H, HV, K, V):

```

```

slots = 8
return {
    "rawv": torch.randn(slots, HV, _RING_LEN, V,
                        device=_DEVICE, dtype=torch.bfloat16),
    "rawk": torch.randn(slots, H, _RING_LEN, K,
                        device=_DEVICE, dtype=torch.bfloat16),
    "g": torch.randn(slots, HV, _RING_LEN, K,
                     device=_DEVICE, dtype=torch.float32),
    "beta": torch.randn(slots, HV, _RING_LEN,
                       device=_DEVICE, dtype=torch.float32),
}

```

```

def _compare_case(case, num_warps, use_ring=False):
    # ... 构造输入, ring 模式克隆两份垃圾模板 ...
    if use_ring:
        template = _make_ring_buffers(H, HV, K, V)
        rings_ref = {name: buf.clone() for name, buf in template.items()}
        rings_fus = {name: buf.clone() for name, buf in template.items()}
    else:
        rings_ref = rings_fus = None
    o_ref, conv_ref, win_ref, ic_ref = _run_reference(
        inp, B, T, H, HV, K, V, lower_bound, rings=rings_ref)
    o_fus, conv_fus, win_fus, ic_fus = _run_fused(
        inp, B, T, H, HV, K, V, lower_bound, num_warps, rings=rings_fus)

    if use_ring:
        # 全张量逐位比较: ring 值是 elementwise 计算 (conv FMA 链、gate、
        # sigmoid), 位于所有 tl.sum 上游, 因此任何 num_warps 下都应 bit-exact。
        # 共享垃圾初值保证越界 / 负 slot 写入也会导致 mismatch。
        for name in ("rawv", "rawk", "g", "beta"):
            assert torch.equal(rings_ref[name], rings_fus[name]), name
    else:
        # snapshot 模式: intermediate_ssm 是 fp32 缓存, 允许归约顺序
        # 差异累积的 ~1 ulp/step 误差。
        torch.testing.assert_close(
            ic_ref[valid_rows], ic_fus[valid_rows], atol=4e-3, rtol=0)

```

评论区精华

围绕 dispatch 门控的讨论最有价值: BBuf 指出 "This enables fused+ring for all eligible batches, but the runtime still uses BV=4. In the H20 results above, B=4/16/64 go from 10.18/14.99/51.46 us to 19.06/67.38/290.05 us. Can we keep the unfused path for the batch/architecture combinations that regress?". yuan-luo 先回应 "I've restricted the new fused+ring path to SM90 with B=1", 随后重新扫描 Hopper/Blackwell 数据后修订为 "The B=1 restriction stays; the SM90-only restriction had no measurement behind it", 最终门控改为 SM90/SM100 且 B=1。BBuf 还建议补充 backend 级测试: "Could we add a backend test that runs verify -> commit -> verify with both partial and full

acceptance? ... Please compare the committed SSM state, conv state, and next verify output against the unfused path", yuan-luo 落实为新增的

`test_kda_fused_verify_backend.py`。

- fused+ring 在大 batch 回归，dispatch 如何门控 (performance): yuan-luo 先限制为 SM90 B=1，随后重新扫描 Hopper 与 Blackwell 数据，确认两个架构都在 B=1 胜出、B>=2 (T=8) 或 B>=4 (T=4) 败退，最终门控改为 SM90/SM100 且 B=1，snapshot 与 CuTe 路径不变。
- backend 级 verify -> commit -> verify 覆盖建议 (testing): yuan-luo 新增 `test_kda_fused_verify_backend.py`，覆盖部分 / 全部接受、多层与 GQA，并用独立 snapshot oracle 断言 committed 状态，避免两个 ring 分支共享同一错误。
- fused 与 reference 输出差 1 bf16 ulp 导致的 CI 失败 (correctness): 输出比较从 `torch.equal` 改为 `torch.testing.assert_close`(`rtol=2**-7`, `atol=1e-7`); ring 值保持全等断言，因为它们位于所有 `tl.sum` 上游，是 elementwise 计算。

风险与影响

- 风险：
 1. 位级一致性契约脆弱：ring 值的正确性依赖 fold 内核 (`kda_replayssm_spec_decode.py`) 与 unfused recurrence 的 bit-clone 关系，本 PR 只在写入侧保证字节一致，若 fold 或 recurrence 后续改变运算顺序会静默产生错误 committed state。backend 测试中的独立 snapshot oracle 是间接缓解，但未直接对 fold 输出做位级断言。
 2. 经验性 dispatch 门控：B=1 + SM90/SM100 的限制来自 H20-3e 与 B300 两卡实测，B200 (生产目标) 数字待测；PBV=4 仍是按 snapshot 写流量调优的常数，ring 模式下最优 BV 已观测到 16-32，未重调前部分 batch 场景收益打折。
 3. 浮点容差回归面：CI 在 H100 上曾出现 `test_replayssm_ring_matches_unfused`[case 10] 一个元素差 1 bf16 ulp 的失败，原因是 fused BV=4 tile 与 reference BV=32 tile 归约顺序不同；输出比较已改为 `rtol=2**-7` 容差，但 conv state 与 ring 仍用 `torch.equal` 全等断言 (这些值是 elementwise，理论上安全)。
 4. GQA 写入者分裂隐患：rawk 按 k-head 写、g/beta 按 v-head 写，若 head 索引错误会静默污染其他 head 的 ring，这是测试专门覆盖的场景。
 5. 默认路径安全性：CACHE_RING=False 为编译期死代码，snapshot 与 CuTe 路径完全不变，回归面被限制在显式启用 ReplaySSM 的场景。
 - 影响：对用户：启用 `--enable-linear-replayssm` 的 KDA 投机验证用户可同时获得 fused verify 加速与 ring 内存收益，B=1 场景延迟提升 1.27x-1.49x，且 interaction 快照写流量下降约 64 倍。
 - 对系统：ring 缓冲区沿用 `memory_pool.py` 既有不变量 (长度为 2 的幂且不小于 2 倍 draft token 数)，无新增池配置或布局变更；ReplaySSM 下 `intermediate_ssm` 本就被为 None，本 PR 只是让 fused 路径不再绕过该设计。
 - 对团队：确立了 KDA verify 的实证调优流程 (sweep 基准 -> 数据驱动门控 -> 位级一致性测试范式)，为后续 batch-aware dispatch 与 BV 重调提供了工具和测试基座。
 - 风险标记：核心 verify 路径变更，B=1 限定启用 (经验门控)，位级一致性依赖 fold 内核，输出容差 1 bf16 ulp，BV 未重调 (B200 待测)

关联脉络

- PR #39405 [misc] Revert #38346, #33426, #39061 and #39219: 该批量回退涉及 `test_fused_kda_conv_recurrent_verify.py` 及注意力相关内核，说明本 PR 所在的 fused verify 内核区域近期经历过回归波折；本 PR 的 ring 位级一致性测试可视为对该区域的加固。
- PR #39446 Reland fix(qsa): clamp the compress gather to the rows (#38346): 同为 spec-verify 内核路径的 '回退后重新落地' 模式，与 `qsa_indexer.py` 相邻的 KDA verify 代码共享相同的正确性风险面。
- PR #38409 [Fix] Wait for PDL before reading DeepSeek V4 K cache locations: 本 PR 内核同样依赖 PDL (prologue 与 producer qkv-projection GEMM 尾部重叠)，两者同属 verify 热路径的时序正确性范畴。
- PR #38554 [Spec] Allow speculative workers to stage prefill shared reads: 同属 speculative-decode verify 路径的性能演进线，与本 PR 一起构成 KDA/spec verify 的持续优化趋势。