PR #33630 完整报告

sgl-project/sglang

Add kda replayssm tests

合并时间：2026-08-10 09:11

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/33630

执行摘要

- 一句话：KDA ReplaySSM 新增 5 组 parity 测试，锁定 gate 与 ring 正确性
- 推荐动作：值得精读。虽然文件全是测试，但它们展示了高质量内核 parity 测试的设计范式：1) 从原始输入同时驱动双侧，不让任何一侧预先享有 gate 公式；2) 用不同 shape 矩阵（GQA、非 pow2、单请求、padding）钉死索引偏移类 bug；3) 对性能敏感的批量折叠用位级相等而非数值容差；4) 在测试标题中明确记录“该测试在修复前代码上会红”的历史价值。建议后续维护 KDA/ReplaySSM 相关内核的开发者以此为模板补充测试，也建议阅读 test_kda_replayssm_fold.py 中 safe gate 与 softplus 的分支设计，这是捕获真实回归的关键。

功能与动机

PR body 说明这是 #32541 的 follow-up，并阐述了测试的必要性：

> The GDN side of ReplaySSM has parity tests upstream; the KDA side landed with none... The whole feature rests on one claim, that the state rebuilt by replaying raw inputs is bit-identical to what the recurrent baseline would have committed. That claim is exactly what silently breaks: an earlier version of this path recomputed the gate on the torch side with a subtly different formula, which left every output looking correct while the state drifted underneath. End-to-end accuracy does not catch that; a parity test does.

即此前 KDA 侧没有任何 parity 测试，而该功能正确性恰好依赖“重放原始输入重建的状态与基线逐位一致”，并且曾真实发生过 gate 公式（plain softplus 与 K3 safe gate）不一致导致状态悄然漂移的 bug，端到端精度无法发现，必须靠 parity 测试才能拦住。

实现拆解

实现分四步推进，全部落在 test/registered/kernels/ 目录下：

1. 确立 parity 测试方法：所有测试都采用“双臂对照”结构。baseline 臂用 fused_sigmoid_gating_delta_rule_update(is_kda=True, disable_state_update=True) 让内核内部生成 gate，并把每步状态快照写入 intermediate_states_buffer；ring 臂让同一内核以 cache_ring=True 模式把 rawv/rawk/gk/beta 写入每 slot 的环形缓冲，再用 commit_kda_replayssm_spec（或 commit_kda_replayssm_spec_all_layers）把环折叠回 checkpoint。两侧输入完全相同，最后比较折叠结果与基线最后一步快照的相对误差。关键

设计是“两侧都从原始 (a, b, A_log, dt_bias) 构造”，避免直接喂同一个 gk 导致 gate 公式差异被掩盖。

2. 分文件覆盖不同风险面：
   - test_kda_replayssm_fold.py：显式覆盖 K3 的 safe gate（lower_bound * sigmoid(exp(A_log) * x)，lower_bound = -5.0）与普通 softplus 两个分支，直接钉死 gate 公式；
   - test_kda_replayssm_ring_fused.py：验证融合进 verify 内核的 ring 写入（CACHE_RING=True）与内核自身上一步状态一致，覆盖 GQA、非 pow2 K/V、单请求、padding 槽位；
   - test_kda_replayssm_ring_ragged.py：覆盖变长（varlen）验证布局下的 ring 写入，含 full/partial commit 与 padding 槽位；
   - test_kda_replayssm_fold_batched.py：验证 layer-batched 折叠（一次 launch）与逐层循环调用结果 torch.equal 位级相同，防层偏移 /stride bug；
   - test_kda_mtp_cutedsl_replayssm_ring.py：SM100 专属，覆盖 CuTe MTP 验证内核的 CACHE_RING 模式，额外校验输出张量在两种模式下 bitwise 相等以及 CUDA graph padding 槽位不被写坏。
3. CI 注册：使用 register_cuda_ci(est_time=..., stage="base-b-kernel-unit", runner_config=...) 将测试注册进 kernel-unit stage，fold 系列与 fused/ragged 用 1-gpu-large，CuTe MTP 因 SM100 限制用 4-gpu-b200。
4. 演进修正：提交记录显示先添加测试、再注册 CI，随后发现 CuTe 测试在 1-gpu-h100 上失败（libNVVM 拒绝为 sm_90a 生成设备 IR），通过“keep the cutedsl ring test on b200”提交将之固定到 b200 runner，最终合并 main，全部 41 个用例在 8x B300 上通过。

关键文件：

- test/registered/kernels/test_kda_replayssm_fold.py（模块 折叠测试；类别 test；类型 test-coverage；符号 TestKDAReplaySSMFoldParity, _parity, test_safe_gate, test_softplus_gate）：最关键的 parity 测试：从原始输入同时驱动 verify 内核与 fold 路径，显式覆盖 K3 safe gate 与普通 softplus 两个分支，能捕获真实发生过的 gate 公式漂移 bug（body 明确提到修复前会使此测试变红）。
- test/registered/kernels/test_kda_mtp_cutedsl_replayssm_ring.py（模块 内核测试；类别 test；类型 test-coverage；符号 _run, test_cutedsl_ring_fold_parity, test_cutedsl_fused_output_norm, test_cutedsl_cuda_graph_padding_slot_is_safe）：覆盖 SM100 专属 CuTe MTP 验证内核的 CACHE_RING 模式：验证 ring 折叠与内核自身快照的 parity、输出张量 bitwise 相等、CUDA graph padding 槽位不被写坏，并因 libNVVM 的 sm_90a 限制必须跑在 4-gpu-b200。
- test/registered/kernels/test_kda_replayssm_ring_ragged.py（模块 环形测试；类别 test；类型 test-coverage；符号 _run_case, test_ragged_full_commit, test_ragged_partial_commit, test_ragged_pad_slot）：覆盖变长（varlen/ragged）验证布局下 CACHE_RING 环形写入的正确性，包括 full/partial commit 与 padding 槽位，形状覆盖 GQA 与 K3-like TP8。

- test/registered/kernels/test_kda_replayssm_fold_batched.py（模块 批量折叠；类别 test；类型 test-coverage；符号 _rand_rings, test_fold_batched_matches_per_layer）：验证 layer-batched 折叠（一次 launch 把层打包进 head grid 轴）与逐层循环调用在 bit 级完全一致（torch.equal），防止 grid 展开引入层偏移 /stride 错误，覆盖 GQA、非 pow2 K/V、track 开关、padding 槽位等。
- test/registered/kernels/test_kda_replayssm_ring_fused.py（模块 环形融合；类别 test；类型 test-coverage；符号 test_ring_fold_parity）：验证融合进 verify 内核的 ring 写入（CACHE_RING=True）与内核自身逐步状态一致，覆盖两种 gate 分支（safe/softplus）与 padding 槽位，是 KDA 侧核心 ring 写入路径的直接回归测试。

关键符号：test_cutedsl_ring_fold_parity, test_cutedsl_fused_output_norm, test_cutedsl_cuda_graph_padding_slot_is_safe, test_ragged_full_commit, test_ragged_partial_commit, test_ragged_pad_slot, TestKDAReplaySSMFoldParity._parity, test_safe_gate, test_softplus_gate, test_fold_batched_matches_per_layer, test_ring_fold_parity

关键源码片段

test/registered/kernels/test_kda_replayssm_fold.py

最关键的 parity 测试：从原始输入同时驱动 verify 内核与 fold 路径，显式覆盖 K3 safe gate 与普通 softplus 两个分支，能捕获真实发生过的 gate 公式漂移 bug（body 明确提到修复前会使此测试变红）。

```
def _parity(self, lower_bound):
    # 双臂 parity：baseline 由 verify 内核内部生成 gate，fold 侧在 torch 里
    # 重算 gk/beta 写入 ring 再折叠。两侧输入完全一致，只有 gate 来源不同，
    # 因此 gate 公式不一致会直接表现为相对误差超限。
    dev = "cuda"
    B, T, HV, K, V, H, L = self.B, self.T, self.HV, self.K, self.V, self.H, self.L
    scale = K**-0.5
    torch.manual_seed(0)

    # 使用固定的随机种子构造 q/k/v/a/b/A_log/dt_bias/h0，保证可复现。
    q = torch.randn(B, T, H, K, device=dev, dtype=torch.float32)
    k = torch.randn(B, T, H, K, device=dev, dtype=torch.float32)
    v = torch.randn(B, T, HV, V, device=dev, dtype=torch.float32)
    a = torch.randn(B, T, HV, K, device=dev, dtype=torch.float32)
    b = torch.randn(B, T, HV, device=dev, dtype=torch.float32)
    A_log = torch.randn(HV, device=dev, dtype=torch.float32)
    dt_bias = torch.randn(HV, K, device=dev, dtype=torch.float32)
    h0 = torch.randn(B, HV, V, K, device=dev, dtype=torch.float32)

    # slot 从 1 开始（0 保留给无效槽位），与生产内存池约定一致。
    slots = torch.arange(1, B + 1, device=dev, dtype=torch.int32)
    num_slots = B + 1

    # baseline 臂：verify 内核内部按 KDA 规则形成 gate，
```

```
# 并把每步状态写入 intermediate_states_buffer（禁掉最终 state 更新）。
h0_src = torch.zeros(num_slots, HV, V, K, device=dev, dtype=torch.float32)
for j in range(B):
    h0_src[slots[j]] = h0[j]
inter = torch.zeros(num_slots, T, HV, V, K, device=dev, dtype=torch.float32)
fused_sigmoid_gating_delta_rule_update(
    A_log=A_log, a=a, dt_bias=dt_bias,
    softplus_beta=1.0, softplus_threshold=20.0,
    q=q, k=k, v=v, b=b,
    initial_state_source=h0_src, initial_state_indices=slots,
    scale=scale, use_qk_l2norm_in_kernel=True,
    is_kda=True, lower_bound=lower_bound,
    disable_state_update=True,
    intermediate_states_buffer=inter, intermediate_state_indices=slots,
    cache_steps=T,
)
accept = T # 提交完整窗口
base = torch.stack([inter[slots[j], accept - 1] for j in range(B)], 0)

# fold 臂：在 torch 侧按两个分支显式重算 gate，避免两端共用同一个 gk
# 而掩盖公式差异。这是本测试能抓到 softplus/safe-gate 漂移的关键。
x = a + dt_bias.view(1, 1, HV, K)
exp_a_log = torch.exp(A_log).view(1, 1, HV, 1)
if lower_bound is not None:
    gk = lower_bound * torch.sigmoid(exp_a_log * x)
else:
    gk = -exp_a_log * torch.nn.functional.softplus(x)
beta = torch.sigmoid(b)

# 把 v/k/gk/beta 转置后写入 slot 的 ring，再调用 commit 折叠回 checkpoint。
rawv = torch.zeros(num_slots, HV, L, V, device=dev, dtype=torch.float32)
rawk = torch.zeros(num_slots, H, L, K, device=dev, dtype=torch.float32)
gkr = torch.zeros(num_slots, HV, L, K, device=dev, dtype=torch.float32)
betar = torch.zeros(num_slots, HV, L, device=dev, dtype=torch.float32)
ckpt = torch.zeros(num_slots, HV, V, K, device=dev, dtype=torch.float32)
for j in range(B):
    s = slots[j].item()
    rawv[s, :, :T] = v[j].transpose(0, 1)
    rawk[s, :, :T] = k[j].transpose(0, 1)
    gkr[s, :, :T] = gk[j].transpose(0, 1)
    betar[s, :, :T] = beta[j].transpose(0, 1)
    ckpt[s] = h0[j]
acc = torch.full((B,), accept, device=dev, dtype=torch.int32)
commit_kda_replayssm_spec(
    ckpt, rawv, rawk, gkr, betar, slots, acc,
    max_cache_len=L, num_k_heads=H, use_qk_l2norm_in_kernel=True,
)
fold = torch.stack([ckpt[slots[j].item()] for j in range(B)], 0)
```

```
# 相对误差必须小于 1e-3，fold 与 verify 内核状态应高度一致。
rel = ((fold - base).abs().max() / base.abs().max().clamp_min(1e-6)).item()
self.assertLess(rel, 1e-3, f"fold vs verify parity failed: rel={rel:.3e}")
```

评论区精华

本 PR 没有收到任何 review 评论（review_comments_count = 0），评论区全部是 /rerun-test 命令与 CI 结果回执。从中能提炼出一个有技术含量的点：

- 第一次把 5 个测试一起放到 1-gpu-h100 上跑时失败，原因是 test_kda_mtp_cutedsl_replayssm_ring.py 依赖 SM100 的 CuTe DSL 内核，h100 的 sm_90a 无法编译；
- 作者随后通过提交"keep the cutedsl ring test on b200"将 CuTe 测试单独固定到 4-gpu-b200，后续 rerun 全部通过，说明该测试的硬件约束被 CI 配置正确吸收。

这同时印证了测试文件中的注释：k3 等非 SM100 平台跑这个文件是编译失败而非数值失败，因此必须选择正确的 runner_config。

- SM100 专属 CuTe 测试的 runner 选择 (testing): CuTe DSL 内核仅支持 SM100（libNVVM 拒绝为 sm_90a 生成设备 IR），该测试必须运行在 b200，不能与其他测试混在 h100 上。

风险与影响

- 风险：具体风险如下：
  - 硬件依赖：test_kda_mtp_cutedsl_replayssm_ring.py 仅能在 SM100 上运行，并要求 runner 池中有 4-gpu-b200；若 CI 机器池变动，该测试可能无法调度或被迫跳过，降低 KDA CuTe 路径的覆盖。
  - 容差设置：CuTe 测试相对误差阈值给到 2e-2（bf16 存储所限），理论上可能放过小幅数值漂移；其余测试为 1e-3，更严格。若后续调整阈值应结合 bf16 精度理解。
  - 回归防护面：测试全部为 parity 性质，若 KDA ReplaySSM 未来的行为有意改变（如 gate 公式变更），测试会失败并需要显式更新，这是正面约束，但对不熟悉该路径的开发者可能产生困惑。
  - CI 时长：新增 5 个 GPU 内核测试（每个约 15-27 秒），虽然注册到 kernel-unit stage，但仍会占用 CI 资源；若并发 runner 不足可能拖慢合入节奏。
  - 无源码改动：本 PR 不改任何生产代码，不会直接引入回归，但它把当前 KDA ReplaySSM 行为固化为契约，任何相关源码改动都会在此显性暴露。
- 影响：影响范围集中在测试与 CI 侧：
  - 对用户：无运行时影响，但 KDA ReplaySSM 路径获得可执行的正确性护栏，防止"输出正常但状态漂移"的静默回归再次出现。
  - 对系统：CI 新增 5 个 GPU 内核测试，其中 1 个要求 SM100（b200 runner），其余在 1-gpu-large 上运行，增加约 100 秒的 kernel-unit 总时长。
  - 对团队：明确了该功能的正确性契约——gate 公式必须与 K3 一致（safe gate）、ring 偏移 / 层折叠必须位级一致、padding 槽位必须零写入，后续开发与 code review 有了

具体依据。

- 与 GDN 侧已有 parity 测试形成对照，补齐了 KDA 侧缺失的另一半覆盖，使 ReplaySSM 两条实现路径都具备回归保障。
- 风险标记：纯测试 PR，无源码主路径改动，SM100-only 测试依赖 b200 runner，gate 公式回归是核心风险，新增 GPU CI 时长

关联脉络

- PR #32541 KDA ReplaySSM spec-verify path: PR body 明确声明本 PR 是其 follow-up，为其补齐 KDA 侧 parity 测试；该特性引入了 kda_replayssm_spec_decode.py、CACHE_RING 写入与 is_kda 环形形状。
- PR #34189 [DSV4] Fix silent KV corruption when speculative draft tokens > 4: 同为投机解码下环形缓冲 / 状态写入正确性修复（压缩环静默写坏 KV），与 KDA ReplaySSM 的 ring 写入属于同一风险面，回归测试可互相印证。
- PR #34184 Fix stale track rows corrupting conv checkpoints under the prefill graph: 同为 checkpoint 重建正确性问题（残留 track 行导致 conv checkpoint 错乱），KDA ReplaySSM fold 也依赖从原始输入重建 checkpoint，目标一致。
- PR #34043 [srt] Fix sconv state memory corruption on specdec: 同属投机解码状态一致性与内存破坏修复，KDA ReplaySSM 的中间状态快照替代是其演进方向之一。