PR #26586 完整报告

sgl-project/sglang

[KDA] Support KDA packed decode

合并时间：2026-06-01 16:52

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/26586

执行摘要

- 一句话：KDA 解码融合内核避免 Python 调度开销
- 推荐动作：值得精读。设计模式（针对特定形状定制 fusion kernel 消除调度开销）可推广至其他线性注意力变体。与 GDN packed decode 的差异点（per-K 门控 vs per-head 标量）展示了如何基于已有优化框架适配不同计算模式。

功能与动机

KDA decode 在小 batch 时 overhead-bound。现有流程中 causal_conv1d_update(mixed_qkv) 后需依次执行 split、unflatten、unsqueeze 等视图操作，以及一次额外核启动，导致每层 CPU 开销约 15–20 µs。同时通用 fused_sigmoid_gating_delta_rule_update_kernel 携带五个编译期标志位（for T 循环、IS_VARLEN、EAGLE 掩码等），在 T=1 退化分支中仍浪费寄存器。参考 GDN 在 #20627 中的 packed decode 优化（取得约 2.59× 加速），本 PR 将此方案移植至 KDA，并适配 KDA 的 per-K 门控。

实现拆解

1. 新增 Triton 融合核(fused_recurrent_kda_packed_decode_kernel)：位于 fused_recurrent.py，对 [B, qkv_dim] 的 packed conv1d 输出直接通过指针偏移提取 Q/K/V，并在寄存器中计算 gate（-exp(A_log) * softplus(a + dt_bias)）和 beta（sigmoid(b)），执行 per-K 向量衰减的 delta 规则状态更新，写回输出和更新后状态。核专门面向 T=1（无 query_start_loc）。
2. Triton kernel 封装(TritonKDAKernel.packed_decode)：在 kda_triton.py 中设置 supports_packed_decode = True，并在 packed_decode 方法中处理生产环境可能传入的多维形状（a 的 [B, HV, K] 等），零开销折叠为 2D 后调用融合核。非 CUDA 平台（CPU/NPU）该标志保持 False，自动回退原路径。
3. 后端调度层集成(KDAKernelDispatcher 和 KDAAttnBackend)：在 kda_backend.py 中，KDAKernelDispatcher 添加 supports_packed_decode 属性和 packed_decode 方法；KDAAttnBackend.forward_decode 在支持 packed decode 时短路调用，跳过 split/reshape。加入 assert qkv.shape[0] == cache_indices.shape[0] 确保 T=1 不变量。
4. 单元测试(test_kda_kernels.py)：新增 TestKDAPackedDecode，覆盖 B ∈ {1,4,32,128}、不对称头数、PAD_SLOT_ID=-1 场景，与基线输出 / 状态对比（atol ≤ 3e-5，bf16）。

5. 基准测试 + 正确性检查脚本(bench_kda_decode.py)：独立脚本支持 --mode bench 和 --mode correctness，采用 wall-clock 计时以体现 CPU 调度收益。

关键文件：

- benchmark/bench_linear_attention/bench_kda_decode.py（模块 KDA 基准；类别 source；类型 dependency-wiring；符号 make_inputs, run_baseline, run_packed, check_correctness）：新增基准测试和正确性校验脚本，用于度量 packed decode 的延迟收益并验证数值精度。
- python/sglang/srt/layers/attention/fla/fused_recurrent.py（模块 KDA 内核；类别 source；类型 core-logic；符号 fused_recurrent_kda_packed_decode_kernel, fused_recurrent_kda_packed_decode）：核心实现文件：新增 fused_recurrent_kda_packed_decode_kernel Triton 融合核及其 Python wrapper，实现 packed decode 的全部计算逻辑。
- test/registered/attention/test_kda_kernels.py（模块 KDA 测试；类别 test；类型 test-coverage；符号 TestKDAPackedDecode, _make_inputs, _run_baseline, _run_packed）：新增 TestKDAPackedDecode 测试类，验证 packed decode 与 baseline 在多种 batch 和形状下的数值一致性。
- python/sglang/srt/layers/attention/linear/kda_backend.py（模块 KDA 后端；类别 source；类型 core-logic；符号 packed_decode）：后端调度层集成，负责短路原 decode 路径为 packed 路径，并管理 supports_packed_decode 标志。
- python/sglang/srt/layers/attention/linear/kernels/kda_triton.py（模块 KDA Triton；类别 source；类型 core-logic；符号 packed_decode）：TritonKDAKernel 类增加 supports_packed_decode 标志和 packed_decode 方法，对输入形状进行归一化并调用融合核。

关键符号：fused_recurrent_kda_packed_decode_kernel, fused_recurrent_kda_packed_decode, TritonKDAKernel.packed_decode, KDAKernelDispatcher.packed_decode, KDAAttnBackend.forward_decode

关键源码片段

python/sglang/srt/layers/attention/fla/fused_recurrent.py

核心实现文件：新增 fused_recurrent_kda_packed_decode_kernel Triton 融合核及其 Python wrapper，实现 packed decode 的全部计算逻辑。

```
@triton.jit
def fused_recurrent_kda_packed_decode_kernel(
    mixed_qkv, a, b, A_log, dt_bias, o, h0, ht, ssm_state_indices, scale,
    stride_mixed_qkv_tok: tl.constexpr, stride_a_tok: tl.constexpr,
    stride_b_tok: tl.constexpr, stride_init_state_token: tl.constexpr,
    stride_final_state_token: tl.constexpr, stride_indices_seq: tl.constexpr,
    H: tl.constexpr, HV: tl.constexpr, K: tl.constexpr, V: tl.constexpr,
    BK: tl.constexpr, BV: tl.constexpr, SOFTPLUS_THRESHOLD: tl.constexpr,
    USE_QK_L2NORM_IN_KERNEL: tl.constexpr,
):
```

```
# 每个程序处理一个 V 维块和一个 (head, token) 对
i_v, i_nh = tl.program_id(0), tl.program_id(1)
i_n, i_hv = i_nh // HV, i_nh % HV
i_h = i_hv // (HV // H)

o_k = tl.arange(0, BK)
o_v = i_v * BV + tl.arange(0, BV)
mask_k = o_k < K
mask_v = o_v < V
mask_h = mask_v[:, None] & mask_k[None, :]

state_idx = tl.load(ssm_state_indices + i_n * stride_indices_seq).to(tl.int64)
p_o = o + (i_n * HV + i_hv) * V + o_v

if state_idx < 0:
    zero = tl.zeros([BV], dtype=tl.float32).to(p_o.dtype.element_ty)
    tl.store(p_o, zero, mask=mask_v)
    return

p_h0 = h0 + state_idx * stride_init_state_token
p_h0 = p_h0 + i_hv * V * K + o_v[:, None] * K + o_k[None, :]
b_h = tl.load(p_h0, mask=mask_h, other=0).to(tl.float32)

p_mixed = mixed_qkv + i_n * stride_mixed_qkv_tok
q_off = i_h * K + o_k
k_off = (H * K) + i_h * K + o_k
v_off = (2 * H * K) + i_hv * V + o_v
b_q = tl.load(p_mixed + q_off, mask=mask_k, other=0).to(tl.float32)
b_k = tl.load(p_mixed + k_off, mask=mask_k, other=0).to(tl.float32)
b_v = tl.load(p_mixed + v_off, mask=mask_v, other=0).to(tl.float32)

if USE_QK_L2NORM_IN_KERNEL:
    b_q = b_q / tl.sqrt(tl.sum(b_q * b_q) + 1e-6)
    b_k = b_k / tl.sqrt(tl.sum(b_k * b_k) + 1e-6)
b_q = b_q * scale

# KDA per-K 门控：a 是 [B, HV*K] 的向量，dt_bias 是 [HV*K] 的向量
p_a = a + i_n * stride_a_tok + i_hv * K + o_k
p_dt = dt_bias + i_hv * K + o_k
b_a = tl.load(p_a, mask=mask_k, other=0).to(tl.float32)
b_dt = tl.load(p_dt, mask=mask_k, other=0).to(tl.float32)
A_log_val = tl.load(A_log + i_hv).to(tl.float32)

x = b_a + b_dt
softplus_x = tl.where(x <= SOFTPLUS_THRESHOLD, tl.log(1.0 + tl.exp(x)), x)
b_g = -tl.exp(A_log_val) * softplus_x

b_val = tl.load(b + i_n * stride_b_tok + i_hv).to(tl.float32)
beta_val = tl.sigmoid(b_val).to(tl.float32)
```

```
    b_h *= tl.exp(b_g)[None, :]
    b_v -= tl.sum(b_h * b_k[None, :], axis=1)
    b_v *= beta_val
    b_h += b_v[:, None] * b_k[None, :]
    b_o = tl.sum(b_h * b_q[None, :], axis=1)

    tl.store(p_o, b_o.to(p_o.dtype.element_ty), mask=mask_v)
    p_ht = ht + state_idx * stride_final_state_token
    p_ht = p_ht + i_hv * V * K + o_v[:, None] * K + o_k[None, :]
    tl.store(p_ht, b_h.to(p_ht.dtype.element_ty), mask=mask_h)
```

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

后端调度层集成，负责短路原 decode 路径为 packed 路径，并管理 supports_packed_decode 标志。

```
class KDAAttnBackend(MambaAttnBackendBase):
    def forward_decode(self, layer, mixed_qkv, a, b, **kwargs):
        # ... conv1d ...
        qkv = causal_conv1d_update(
            mixed_qkv, conv_states.transpose(-1, -2),
            layer.conv_weights, layer.bias,
            activation="silu", conv_state_indices=cache_indices,
        )
        # 支持 packed 时短路
        if self.kernel_dispatcher.supports_packed_decode:
            assert qkv.shape[0] == cache_indices.shape[0], (
                "KDA packed decode requires T=1"
            )
            return self.kernel_dispatcher.packed_decode(
                mixed_qkv=qkv, a=a, b=b,
                A_log=layer.A_log, dt_bias=layer.dt_bias,
                scale=layer.head_k_dim ** -0.5,
                ssm_states=ssm_states,
                cache_indices=cache_indices,
                num_v_heads=layer.num_v_heads,
                head_v_dim=layer.head_v_dim,
            )
        # 原始路径
        q, k, v = qkv.split([layer.q_dim, layer.k_dim, layer.v_dim], dim=-1)
        q = q.unflatten(-1, (-1, layer.head_q_dim)).unsqueeze(0)
        k = k.unflatten(-1, (-1, layer.head_k_dim)).unsqueeze(0)
        v = v.unflatten(-1, (-1, layer.head_v_dim)).unsqueeze(0)
        return self.kernel_dispatcher.decode(
            q=q, k=k, v=v, a=a, b=b,
            A_log=layer.A_log, dt_bias=layer.dt_bias,
            ssm_states=ssm_states,
            cache_indices=cache_indices,
            query_start_loc=query_start_loc,
        )
```

评论区精华

Review 中主要围绕两点展开：

- 基准测试方法：DarkSharpness 建议复用 JIT kernel benchmark 工具（do_bench_cudagraph）以消除 L2 缓存效应。作者回应：wall-clock 计时更能反映实际解码延迟，因为本优化约 2/3 的收益来自消除 CPU 调度开销，若用 do_bench 会低估收益，已在脚本中添加注释说明。
- 形状归一化与 T=1 假设：BBuf 指出 TritonKDAKernel.packed_decode 中的 a/b/A_log/dt_bias 形状归一化缺少单元测试，且 packed 核硬假定 T=1。作者确认 speculative decode（EAGLE）走 forward_extend，不会误入 forward_decode，并在 forward_decode 中添加了 assert 以强制不变量。关于形状测试，作者在第二次提交中可能已补充（commit 消息为 "Address review comments"），但未在评论中明确说明。
  - 基准测试方法选择 (performance): 保留 wall-clock 计时，并在脚本中添加注释说明选择理由。
  - 形状归一化缺少测试和 T=1 假设确认 (correctness): 已添加 assert 强制 T=1 不变量；形状测试可能在后续提交中补充（commit 消息 'Address review comments'）。

风险与影响

- 风险：
  1. T=1 hard assumption: forward_decode 中的 assert 可防止未来路由变更误用，但若 assert 被禁用或移除，多 token 序列会导致状态错误。
  2. 形状归一化零拷贝视图安全：reshape(-1) 仅改变元数据，但假设输入张量连续性，如果生产代码某些分支传入非连续张量，可能引发错误。当前基线的 fused_sigmoid_gating_delta_rule_update 内部使用 input_guard 处理连续性，packed 路径未显式调用 contiguous()。
  3. CUDA 专用优化：CPU/NPU 平台不启用 packed decode，路径安全但无收益。
  4. benchmark脚本自实现计时：未使用do_bench可能导致L2污染，但作者已声明意图。
     - 影响：对用户：在 NVIDIA GPU（Hopper 架构）上运行 KDA 模型（如 Kimi-Linear-48B）的用户，解码阶段小批量（B≤64）延迟降低约 1.7 倍，大批量（B≥128）降低约 1.4 倍。端到端准确率经 GSM8K 验证不受影响（92.5% vs 88% 阈值）。对系统：新增一个 Triton 编译内核，增加首次启动编译时间约 1–2 秒。无其他运行时影响。对团队：维护两条解码路径（packed/unpacked），packed 路径仅对 CUDA 启用，未来若其他平台支持 Triton 可复用。
- 风险标记：核心路径变更，T=1 假设，缺少 NPU/CPU 测试

关联脉络

- PR #20627 [GDN] Support GDN packed decode: 该 PR 的优化方案直接借鉴了 GDN packed decode 的实现模式，包括融合 kernel 设计、dispatch 层短路逻辑等。