PR #24271 完整报告

sgl-project/sglang

[KDA] Optimize prefill kernels with diagonal and recompute fuse

合并时间：2026-05-09 08:52

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/24271

执行摘要

- 一句话：融合对角线与重计算优化 KDA prefill kernel 性能
- 推荐动作：值得精读。该 PR 展示了 Triton kernel 优化的完整思考：autotune 的权衡、kernel 融合的粒度选择、网格启发式设计。审查评论中的讨论解决了关键的正确性和性能问题，尤其是 exp vs exp2 的澄清、chunk_indices 计数修正、以及单配置回退原因。适合 attention kernel 开发者和对 Triton 性能优化感兴趣的技术人员深入阅读。

功能与动机

Profiling 显示 KDA 占 prefill GPU 时间约 13%（4-GPU TP 配置），两个主导 kernel 的 autotune 配置受限，导致 SM 利用率不足。例如 chunk_gated_delta_rule_fwd_kernel_h_blockdim64 在 TP=8 H=4 时 grid 仅产生 16 个 CTA，远低于 Hopper 的 SM 数量，大部分 SM 空闲。

实现拆解

1. h-recurrence kernel autotune 扩展：在 chunk_delta_h.py 中，为 chunk_gated_delta_rule_fwd_kernel_h_blockdim64 添加 BV=16 和 num_warps=8 的尝试，并引入 NT_BUCKET（3 个桶：NT≤32、≤128、>128）作为 autotune key，使长 / 短 prefill 能选择各自的 optimal tile 大小。同时添加 restore_value 确保 autotune 不会因 in-place 写而污染缓存池。
2. chunk_intra kernel 融合：在 chunk_intra.py 的 chunk_kda_fwd_kernel_inter_solve_fused 中添加两个 tl.constexpr 模式：FUSE_DIAGONAL 将对角线 Akk 计算并入同一 kernel，消除 Akkd 的 HBM 往返；FUSE_RECOMPUTE 进一步将 w/u/kg 重计算并入，消除 Akk_inv 的 HBM 往返。同时对角块使用 bf16 输入进行 tl.dot，加速 tensor core 计算。
3. recompute_w_u kernel 配置调整：在 kda.py 中，recompute_w_u_fwd_kernel 的 autotune 配置从多组 (BK, BV) 缩减为仅 BK=64, BV=64，避免因 triton 3.6 兼容性问题导致非法内存访问或 autotune 选择次优配置；DOT_PRECISION 从 ieee 改为 tf32，利用 Hopper tensor core 加速。
4. 网格感知融合门控：在 kda.py 的 chunk_kda_fwd 函数中，计算 _small_grid = (B * NT_pr * H_per_rank) ≤ 256，当网格小时启用 FUSE_DIAGONAL 和 FUSE_RECOMPUTE（此时 launch 开销为主），大网格时回退到 unfused 路径（避免寄存器压力导致的 spilling）。NT_pr 对于 varlen 使用 chunk_indices.shape[0] 正确计数每

序列 chunk 数量。

5. cumsum 流水线加深：在 cumsum.py 的 chunk_local_cumsum_vector_kernel 的 autotune 配置中加入 num_stages，长 prefill 可选择更深流水线。

关键文件：

- python/sglang/srt/layers/attention/fla/chunk_intra.py（模块 KDA kernel；类别 source；类型 core-logic；符号 chunk_kda_fwd_kernel_inter_solve_fused, chunk_kda_fwd_intra）：核心变更文件：在 chunk_kda_fwd_kernel_inter_solve_fused 中实现了 FUSE_DIAGONAL 和 FUSE_RECOMPUTE 两个融合模式，消除了 Akkd 和 Akk_inv 的 HBM 往返，同时优化了 off-diagonal MMA 的 bf16 输入。文件改动量最大（+487/-96），是性能提升的主要来源。
- python/sglang/srt/layers/attention/fla/kda.py（模块 KDA 调度；类别 source；类型 core-logic；符号 chunk_kda_fwd, recompute_w_u_fwd_kernel, recompute_w_u_fwd, chunk_gla_fwd_kernel_o）：调度器和网格启发式所在：实现了 _small_grid 判断逻辑，根据总 CTA 数量动态选择融合程度；调整了 recompute_w_u_fwd_kernel autotune 配置以适配 triton 3.6；删除了未使用的 chunk_kda_fwd_h_output_fused_kernel；将 chunk_gla_fwd_kernel_o 的 allow_tf32=False 移除以利用 tf32 加速。
- python/sglang/srt/layers/attention/fla/chunk_delta_h.py（模块 KDA 递归；类别 source；类型 core-logic；符号 chunk_gated_delta_rule_fwd_kernel_h_blockdim64, chunk_gated_delta_rule_fwd_h）：h-recurrence kernel autotune 重构：原注释掉的 autotune 被激活但限制为单一配置（BV=32, num_warps=4, num_stages=2），并添加了 NT_BUCKET key 以支持未来分桶；通过 autotune_cache_kwargs 启用缓存；移除了 hardcoded 的 num_warps/num_stages 参数。
- python/sglang/srt/layers/attention/fla/cumsum.py（模块 Cumsum；类别 source；类型 core-logic；符号 chunk_local_cumsum_vector_kernel）：微小调整：在 chunk_local_cumsum_vector_kernel 的 autotune 配置中添加 num_stages 参数，允许长 prefill 选择更深流水线，提升吞吐。

关键符号：chunk_kda_fwd_kernel_inter_solve_fused, chunk_kda_fwd_intra, chunk_gated_delta_rule_fwd_kernel_h_blockdim64, chunk_kda_fwd, recompute_w_u_fwd_kernel, recompute_w_u_fwd, chunk_gla_fwd_kernel_o, chunk_local_cumsum_vector_kernel

关键源码片段

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

核心变更文件：在 chunk_kda_fwd_kernel_inter_solve_fused 中实现了 FUSE_DIAGONAL 和 FUSE_RECOMPUTE 两个融合模式，消除了 Akkd 和 Akk_inv 的 HBM 往返，同时优化了 off-diagonal MMA 的 bf16 输入。文件改动量最大（+487/-96），是性能提升的主要来源。

```
# python/sglang/srt/layers/attention/fla/chunk_intra.py
# 关键变更：融合对角线计算和重计算到 inter+solve kernel

@triton.autotune(
    configs=[
```

```
        triton.Config({"BK": BK, "BV": 64}, num_warps=num_warps)
        for BK in [32, 64]
        for num_warps in [1, 2, 4]
    ],
    # 新增 V、FUSE_RECOMPUTE、FUSE_DIAGONAL 作为 key，确保不同配置被独立缓存
    key=["H", "K", "BC", "V", "FUSE_RECOMPUTE", "FUSE_DIAGONAL"],
    **autotune_cache_kwargs,
)
@triton.jit(do_not_specialize=["T"])
def chunk_kda_fwd_kernel_inter_solve_fused(
    q, k, g, beta, Aqk, Akkd, Akk, scale,
    v_in, w_out, u_out, kg_out, # 新增输出指针，用于 FUSE_RECOMPUTE 写回
    cu_seqlens, chunk_indices, T,
    H: tl.constexpr, K: tl.constexpr, V: tl.constexpr,
    BT: tl.constexpr, BC: tl.constexpr, BK: tl.constexpr, BV: tl.constexpr,
    IS_VARLEN: tl.constexpr, USE_SAFE_GATE: tl.constexpr,
    FUSE_RECOMPUTE: tl.constexpr, FUSE_DIAGONAL: tl.constexpr, # 新增编译常量
):
    # ...（坐标计算略）...

    # 当 FUSE_DIAGONAL 时，额外分配对角线块累加器
    if FUSE_DIAGONAL:
        b_Aqk_d0 = tl.zeros([BC, BC], dtype=tl.float32)
        b_Akk_d0 = tl.zeros([BC, BC], dtype=tl.float32)
        # ... 类似 d1, d2, d3 ...
        m_tc0 = (i_tc0 + o_i) < T

    # 主循环：遍历 K 维度
    for i_k in range(tl.cdiv(K, BK)):
        # 加载 k0, g0（共享）
        p_k0 = tl.make_block_ptr(k, (T, K), (H*K, 1), (i_tc0, i_k*BK), (BC, BK), (1,0))
        p_g0 = tl.make_block_ptr(g, (T, K), (H*K, 1), (i_tc0, i_k*BK), (BC, BK), (1,0))
        b_k0 = tl.load(p_k0, boundary_check=(0,1)).to(tl.float32)
        b_g0 = tl.load(p_g0, boundary_check=(0,1)).to(tl.float32)

        # FUSE_DIAGONAL: 在线计算对角线块的 Aqk_d0 和 Akk_d0
        if FUSE_DIAGONAL:
            b_q0 = tl.load(p_q0, ...).to(tl.float32)
            b_gn0 = tl.load(g + i_tc0*H*K + o_k, ...)
            b_gm0 = tl.clamp(b_g0 - b_gn0[None,:], -126.0, 126.0)
            b_gq0 = tl.where(m_tc0[:,None], exp2(b_gm0), 0.0)
            b_gk0 = tl.where(m_tc0[:,None], exp2(-b_gm0), 0.0)
            b_kgt_d0 = tl.trans(b_k0 * b_gk0)
            b_Aqk_d0 += tl.dot(b_q0 * b_gq0, b_kgt_d0)
            b_Akk_d0 += tl.dot(b_k0 * b_gq0, b_kgt_d0)

        # off-diagonal 块的计算保持不变，但 tl.dot 输入使用 bf16（提高 tensor core 利用率）
        # ...
```

```
    # 融合重计算分支：在 forward substitution 后直接计算 w, u, kg
    if FUSE_RECOMPUTE:
        # 直接使用寄存器的 Akk_inv 块，无需写出到 HBM 再读回
        b_w0 = tl.dot(b_Akk_inv00, b_kb0.to(b_k0.dtype))
        b_u0 = tl.dot(b_Akk_inv00, b_v0)
        b_kg0 = b_k0 * exp(...) # 使用 exp（自然指数）计算绝对门控
        # ... 类似其他块 ...
        # 通过指针直接写回全局内存
        tl.store(p_w0, b_w0.to(p_w0.dtype.element_ty), boundary_check=(0,1))
        tl.store(p_u0, b_u0.to(p_u0.dtype.element_ty), boundary_check=(0,1))
        tl.store(p_kg0, b_kg0.to(p_kg0.dtype.element_ty), boundary_check=(0,1))
    else:
        # 原始路径：将 Akk_inv 写出到全局 Akk 供后续 kernel 使用
        tl.store(p_Akk, ...)
```

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

调度器和网格启发式所在：实现了 _small_grid 判断逻辑，根据总 CTA 数量动态选择融合程度；调整了 recompute_w_u_fwd_kernel autotune 配置以适配 triton 3.6；删除了未使用的 chunk_kda_fwd_h_output_fused_kernel；将 chunk_gla_fwd_kernel_o 的 allow_tf32=False 移除以利用 tf32 加速。

```
# python/sglang/srt/layers/attention/fla/kda.py
# 关键变更：网格感知融合门控 + recompute_w_u 配置简化

# 在 chunk_kda_fwd_intra 调用前计算网格大小，决定是否启用融合
# Total CTAs in inter_solve_fused = NT * B * H_per_rank
_NT_pr = (
    triton.cdiv(q.shape[1], chunk_size)
    if cu_seqlens is None
    else chunk_indices.shape[0] # varlen: chunk_indices 已枚举所有 (seq, chunk) 对
)
_H_pr = q.shape[2] # H_per_rank
_small_grid = q.shape[0] * _NT_pr * _H_pr <= 256

# 将网格启发式传递到 chunk_kda_fwd_intra
chunk_kda_fwd_intra(
    q, k, g, beta, v, scale,
    A=A, Aqk=Aqk, gk=gk, cu_seqlens=cu_seqlens,
    chunk_indices=chunk_indices,
    fuse_diagonal=_small_grid,
    fuse_recompute=_small_grid,
    ...
)

# recompute_w_u_fwd_kernel autotune 简化：移除 BK=32, BV=128 等不稳定配置
@triton.autotune(
    configs=[
        triton.Config({"BK": BK, "BV": BV}, num_warps=num_warps, num_stages=num_stages)
        for BK in [64] # 只有 BK=64 在 triton 3.6 上稳定且最优
```

```
        for BV in [64] # BV=64 经测试精度通过
        for num_warps in [2, 4, 8]
        for num_stages in [2, 3, 4]
    ],
    key=["H", "K", "V", "BT", "IS_VARLEN"],
)
@triton.jit(do_not_specialize=["T"])
def recompute_w_u_fwd_kernel(
    q, k, qg, kg, v, beta, w, u, A, gk,
    cu_seqlens, chunk_indices, T,
    H, K, V, BT, BK, BV, ...
):
    # ... kernel body ( 除配置外无大变化 )
    DOT_PRECISION="tf32" # 从 ieee 改为 tf32，利用 Hopper tensor core

# chunk_gla_fwd_kernel_o 中移除 allow_tf32=False 以启用 tf32
b_o += tl.dot(b_A, b_v) # 原来为 tl.dot(b_A, b_v, allow_tf32=False)
```

评论区精华

- exp vs exp2 一致性：gemini-code-assist 指出 FUSE_RECOMPUTE 块中使用 exp（自然指数）而与同一 kernel 内 exp2（基 2 指数）不一致，可能导致错误。作者 yuan-luo 解释这是有意为之：Akk/Aqk 使用 exp2(g_i - g_j) 用于相对门控差异，而 kb = k•β•exp(gk_cumulative) 使用 exp(gk) 表示绝对累积门控，该约定与原始 FLA 代码一致，经测试验证精度正确。
- chunk_indices 传递：gemini-code-assist 建议在调用 kda_recompute_w_u_fwd 时传递预计算的 chunk_indices 避免重复计算。作者接受并修复。
- 未使用的 fused output kernel：gemini-code-assist 指出 chunk_kda_fwd_h_output_fused_kernel 及其 wrapper 定义但未使用。作者回应是临时调试 kernel，已移除。
- allow_tf32 删除：BBuf 询问为何删除 chunk_gla_fwd_kernel_o 中的 allow_tf32=False。作者说明 tf32 在 Hopper 上更快，精度影响经测试验证可接受。
- recompute_w_u 配置简化：BBuf 询问为何移除 BK=32 和 BV=128 配置。作者报告 triton 3.6 上 BV=128 导致 CUDA 非法内存访问，BK=32 在 autotune 中被误选为最佳但实际慢 2-10 倍，因此仅保留安全且最优的 BK=64, BV=64。
- 网格启发式改进：kaixih 指出 _NT_pr 对于 packed 张量应使用 len(chunk_indices) 而非 cdiv(cu_seqlens[-1], chunk_size)，因为 chunk 不跨序列边界，后者会低估。作者同意并修复；同时 gemini-code-assist 建议在非 varlen 时也应包括 batch 维度，作者部分采纳（已包含 batch 维度）。
- chunk_delta_h autotune 单配置：kaixih 指出 PR 描述声称扩展 BV 到 [16,32,64]，但实际代码只有单一配置 {BV=32, num_warps=4, num_stages=2}。作者解释 triton 3.6 上任何偏离该配置都会导致精度回归（max_diff 达 4.49e-2），因此回退；实际性能增益来自融合优化，而非 h-recurrence autotune 扩展。
    - FUSE_RECOMPUTE 中 exp vs exp2 一致性 (correctness): 保留 exp，无需修改。

- chunk_indices 传递优化 (performance): 已修复，传递 chunk_indices 参数。
- 未使用的 fused output kernel (design): 已移除。
- allow_tf32 删除原因 (performance): 接受删除。
- recompute_w_u 配置简化 (correctness): 简化配置，保持稳定性和性能。
- 网格启发式改进 (correctness): 已修复，使用 chunk_indices.shape[0] 并包含 batch 维度。
- chunk_delta_h autotune 单配置与 PR 描述不一致 (documentation): 更新描述，保持单配置。

风险与影响

- 风险:
  - 精度风险：FUSE_RECOMPUTE 分支使用 exp 而不是 exp2，虽然作者声称是设计如此，但若未来有代码重构或社区其他分支可能误解导致精度错误。此外，off-diagonal MMA 使用 bf16 输入可能产生数值差异，但作者验证 ATOL=2e-2。
  - 环境兼容性风险：recompute_w_u kernel 的 autotune 配置针对 triton 3.6.0 / torch 2.11 / cu130 环境裁剪，旧版 triton 可能无法正确运行或选择次优配置。chunk_delta_h 的 autotune 单配置同样针对特定环境，其他平台可能性能更差或编译失败。
  - 网格启发式不准确：_small_grid 阈值 256 是基于实验选择的魔法数字，可能在不同模型或硬件上不是最优。若 batch 或 head 数变化大，可能导致融合策略错误，影响性能。
  - 测试覆盖不足：PR 没有新增测试用例（仅依赖现有 test_chunk_gated_delta_rule.py），未覆盖融合路径的边界条件（如极端短序列、大 batch、多节点）。
- 影响:
  - 用户影响：对使用 KDA（如 Kimi-Linear-48B）的用户，prefill 吞吐预计提升，延迟降低。对其他不涉及 KDA 的模型无影响。
  - 系统影响：仅影响 attention/fla 模块内的 kernel，无外部 API 变更。需要 triton 3.6+ 兼容性。
  - 团队影响：优化方案具有参考价值，尤其是关于 autotune 的陷阱（in-place 写、版本兼容性）和融合取舍的讨论。后续维护者需注意内核配置与 triton 版本的耦合。
  - 风险标记：核心 kernel 路径变更，autotune 配置与 triton 版本耦合，网格启发式魔法数字，测试覆盖不足

关联脉络

- PR #24627 logits: remove blocking H2D copy: 同样属于 prefill 性能优化系列，移除 logits 处理器的阻塞 H2D 拷贝，与本 PR 目标一致（提升 prefill 阶段 GPU 利用率）。
- PR #24724 [Spec] Disambiguate verified_id into bonus_token(s) / accept_tokens: 虽然主题不同，但该 PR 重构了 speculative decoding 的 token 跟踪，与 KDA 同属 attention 优化方向，且都涉及内部 kernel 的数据流调整。