PR #28320 完整报告

sgl-project/sglang

Fused QK GemmaRMSNorm + RoPE + gate kernel for Qwen3.5

合并时间：2026-06-25 15:58

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/28320

执行摘要

- 一句话：融合 QK 归一化、RoPE 与门控的 Triton 内核，提升 Qwen3.5 吞吐量
- 推荐动作：值得关注该 PR 中 Triton 内核融合的设计模式，特别是复用 RoPE cos/sin cache 的方式以及 PDL 的使用技巧。性能优化思路可推广至其他模型。

功能与动机

Qwen3.5 的注意力前向过程中，需要先对 Q 和 K 进行去交织（gate deinterleave），然后分别应用 GemmaRMSNorm，再应用 NeoX RoPE。原本这些操作由多个独立的 Triton 或 PyTorch 操作完成，导致多次 kernel launch 和中间显存读写，成为性能瓶颈。PR 汇总的基准测试数据证实，融合后吞吐量有显著提升，且无准确率损失。

实现拆解

1. 新增融合内核文件：创建 python/sglang/srt/layers/fused_qk_rmsnorm_rope_gate.py，包含一个 Triton JIT 内核 _fused_qk_rmsnorm_rope_gate_kernel。内核以 (token, head) 二维网格启动，区分 Q head 和 K head，对 Q head 同时处理 gate 的拷贝。支持 programmatic dependent launch（PDL），允许下游核提早启动。
2. 内核逻辑：先加载整个 head 维度计算 RMS 方差并应用权重，然后对旋转部分重新加载并应用 RoPE，尾部 pass-through。所有操作在单个 Triton 程序中完成。
3. 模型集成：在 qwen3_5.py 中新增 forward_prepare_cuda_fused 方法，调用 fused_qk_gemma_rmsnorm_rope_gate。在 self_attention 中，当 _is_cuda 且存在输出 gate 时，优先调度该融合路径，否则回退为原生路径。
4. 条件导入与守卫：仅在 CUDA 后端导入该内核，通过 _pdl_supported() 检查设备能力，确保非 Hopper 架构不使用 PDL。
5. 测试与兼容性：两个测试文件仅有格式修正（行尾、括号风格），不影响逻辑。

关键文件：

- python/sglang/srt/layers/fused_qk_rmsnorm_rope_gate.py（模块 融合内核；类别 source；类型 core-logic；符号 _pdl_supported, _fused_qk_rmsnorm_rope_gate_kernel, fused_qk_gemma_rmsnorm_rope_gate）：新增融合内核文件，包含所有核心逻辑，是 PR 的核心贡献。
- python/sglang/srt/models/qwen3_5.py（模块 模型实现；类别 source；类型 control-flow；符号 forward_prepare_cuda_fused, self_attention）：模型文件中新增融合路径的调度

逻辑，是集成融合内核的关键入口。

- test/registered/unit/hardware_backend/mlx/test_attention_patching.py（模块 MLX 测试；类别 test；类型 test-coverage）：测试文件，仅含格式修正（减少括号折行），与功能无直接关联。
- test/registered/unit/hardware_backend/mlx/test_mlx_runner_pool_contract.py（模块 MLX 测试；类别 test；类型 test-coverage）：测试文件，仅修复文件末尾缺失换行，与功能无直接关联。

关键符号：fused_qk_gemma_rmsnorm_rope_gate, _fused_qk_rmsnorm_rope_gate_kernel, forward_prepare_cuda_fused, _pdl_supported

关键源码片段

python/sglang/srt/models/qwen3_5.py

模型文件中新增融合路径的调度逻辑，是集成融合内核的关键入口。

```
# 在文件顶部条件导入（只在 CUDA 后端）
if _is_cuda:
    from sglang.srt.layers.fused_qk_rmsnorm_rope_gate import (
        fused_qk_gemma_rmsnorm_rope_gate,
    )

# 新增融合前向方法
class Qwen3_5GatedDeltaNet(nn.Module):
    def forward_prepare_cuda_fused(self, positions, hidden_states):
        """Fused QK GemmaRMSNorm + NeoX RoPE + gate deinterleave."""
        qkv, _ = self.qkv_proj(hidden_states)
        if self.attn_output_gate:
            q_gate, k, v = qkv.split(
                [self.q_size * 2, self.kv_size, self.kv_size], dim=-1
            )
        else:
            q_gate, k, v = qkv.split(
                [self.q_size, self.kv_size, self.kv_size], dim=-1
            )
        q_out, k_out, gate_out = fused_qk_gemma_rmsnorm_rope_gate(
            q_gate, k,
            self.q_norm.weight.data, self.k_norm.weight.data,
            self.rotary_emb.cos_sin_cache, positions,
            self.q_norm.variance_epsilon,
            self.num_heads, self.num_kv_heads,
            self.head_dim, self.rotary_emb.rotary_dim,
            has_gate=self.attn_output_gate,
        )
        seq_len = hidden_states.shape[0]
        q = q_out.view(seq_len, -1)
        k = k_out.view(seq_len, -1)
        gate = gate_out.view(seq_len, -1) if gate_out is not None else None
```

```
        return q, k, v, gate

    def self_attention(self, positions, hidden_states, forward_batch):
        # 在 self_attention 顶部新增一个分支：CUDA 且带 gate 时走融合路径
        if _is_cuda and self.attn_output_gate:
            q, k, v, gate = self.forward_prepare_cuda_fused(
                positions=positions, hidden_states=hidden_states,
            )
        elif (_is_hip or _is_xpu) and self.attn_output_gate:
            # 原有 HIP/XPU 融合路径
            q, k, v, gate = self.forward_prepare_fused_gate(
                positions=positions, hidden_states=hidden_states,
            )
        else:
            # 原生路径
            q, k, v, gate = self.forward_prepare_native(
                positions=positions, hidden_states=hidden_states,
            )
        # ... 后续 attention 计算
```

评论区精华

审查者 zcnrex 指出，PDL（Programmatic Dependent Launch）可能并非所有硬件（如 AMD）支持，建议直接使用 tl.extra.cuda.gdc_launch_dependents()。作者 yhyang201 回复“Fixed, thanks!”，最终内核加入了 ENABLE_PDL 条件守卫，仅在支持时调用，防止在其他架构上出错。

- PDL 守卫与硬件兼容性 (design): 作者添加了 ENABLE_PDL 条件守卫和 _pdl_supported() 检查，确保仅在支持时调用。

风险与影响

- 风险：
  - 兼容性风险：内核依赖 Triton 和 NVIDIA CUDA 能力 >=9，在 AMD 或 Intel GPU 上无法运行。PR 通过条件导入和类型检查 _is_cuda 限制，仅 CUDA 后端启用，风险可控。
  - 数值精度风险：融合可能改变中间计算精度，但 AIME26 验证无精度下降，且代码针对 FP16/BF16 做了显式类型转换。
  - 调度变更风险：self_attention 中新增一个分支，可能与其他后端（NPU、AMD）的调度冲突。但 _is_cuda 检查确保仅 CUDA 生效，且默认保留原生路径作为 fallback。
  - 维护风险：新增的 Triton 内核需随 Triton 版本迭代维护，但已独立成文件，影响范围小。
- 影响：
  - 用户影响：仅影响使用 Qwen3.5 模型且在 NVIDIA GPU（sm90+）上推理的用户，获得 1%~9% 的吞吐量提升，无功能变化。
  - 系统影响：无架构性改变，仅模型层优化。
  - 团队影响：为后续模型算子融合提供可复用模式（如 PDL 使用、融合归一化 +RoPE）。

- 风险标记：非 CUDA 硬件不兼容，依赖 Triton 版本稳定性，仅覆盖 Qwen3.5 模型

关联脉络

- 暂无明显关联 PR