PR #23575 完整报告

sgl-project/sglang

[AMD] fused qk gemma norm kernels to reduce four kernels

合并时间：2026-04-25 15:30

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/23575

执行摘要

- 一句话：融合 QK Gemma RMSNorm 为单个 Triton 内核，减少 ROCm 内核启动开销
- 推荐动作：值得精读：展示了如何通过 Trition 内核融合减少 ROCm 平台内核启动开销，是 AMD 性能优化的典型实践。但数据类型硬编码和 reshape 拷贝争议应妥善解决；建议在同类 PR 中提前审查 dtype 与内存布局假设。

功能与动机

来自 PR 描述："From the profiling data, apply_qk_norm function will bring 4 kernels launch on ROCm platform compared two kernels overlapped on CUDA platform. In order to reduce the e2e time cost, fused 4 kernels into one triton kernel"，旨在消除 ROCm 上多余的内核启动开销。

实现拆解

步骤 1: 添加融合核函数 Triton 实现

在 python/sglang/srt/models/utils.py 中新增 _fused_qk_gemma_rmsnorm_kernel 和 fused_qk_gemma_rmsnorm 包装函数。核函数利用 tl.program_id(0) 索引全局行：每行计算 Q 的 GemmaRMSNorm，前 k_rows 行额外计算 K 的 Norm，从而单次启动完成原本需要四次 kernel launch 的操作。输入步长（stride）作为参数传入，以处理非连续切片的读取。

步骤 2: 在 Qwen3.5 模型中条件启用

在 python/sglang/srt/models/qwen3_5.py 的 _apply_qk_norm 方法中，当检测到 _is_hip 为真时，优先调用 fused_qk_gemma_rmsnorm，否则回退到原有逐层 Norm 或 alt-stream 重叠方案。同时移除无引用的 cached_get_processor 行以清理代码。

步骤 3: 性能验证

作者提供了多并发度下的吞吐对比，低并发改善显著，验证了融合的内核确实减少了调度开销。

关键文件：

- python/sglang/srt/models/utils.py（模块 模型工具；类别 source；类型 data-contract；符号 _fused_qk_gemma_rmsnorm_kernel, fused_qk_gemma_rmsnorm）：核心核函数实现及包装入口，新增 95 行 Triton 代码，是整个 PR 的算力来源

- python/sglang/srt/models/qwen3_5.py（模块 模型定义；类别 source；类型 data-contract；符号 _apply_qk_norm）：Qwen3.5 模型入口，新增 elif _is_hip 分支调用融合核函数，是实际产生性能收益的调用点

关键符号：_fused_qk_gemma_rmsnorm_kernel, fused_qk_gemma_rmsnorm, _apply_qk_norm

关键源码片段

python/sglang/srt/models/utils.py

核心核函数实现及包装入口，新增 95 行 Triton 代码，是整个 PR 的算力来源

```
@triton.jit
def _fused_qk_gemma_rmsnorm_kernel(
    Q_ptr, K_ptr, Q_out_ptr, K_out_ptr,
    QW_ptr, KW_ptr,
    q_stride, k_stride, k_rows,
    HEAD_DIM: tl.constexpr, BLOCK_HD: tl.constexpr,
    EPS: tl.constexpr, FP16: tl.constexpr,
):
    pid = tl.program_id(0)
    cols = tl.arange(0, BLOCK_HD)
    mask = cols < HEAD_DIM
    # 注意：输出类型硬编码为 FP16 或 BF16，若输入为 float32 会导致数据类型不匹配
    out_dtype = tl.float16 if FP16 else tl.bfloat16

    # Q norm —— 每个 block 处理一行 Q
    q_off = pid * q_stride + cols
    q = tl.load(Q_ptr + q_off, mask=mask, other=0.0).to(tl.float32)
    w_q = tl.load(QW_ptr + cols, mask=mask, other=0.0).to(tl.float32)
    q_var = tl.sum(q * q, axis=0) / HEAD_DIM
    q_normed = (q * tl.rsqrt(q_var + EPS) * (w_q + 1.0)).to(out_dtype)
    q_out_off = pid * HEAD_DIM + cols
    tl.store(Q_out_ptr + q_out_off, q_normed, mask=mask)

    # K norm —— 只有前 k_rows 个 block 执行
    if pid < k_rows:
        k_off = pid * k_stride + cols
        k = tl.load(K_ptr + k_off, mask=mask, other=0.0).to(tl.float32)
        w_k = tl.load(KW_ptr + cols, mask=mask, other=0.0).to(tl.float32)
        k_var = tl.sum(k * k, axis=0) / HEAD_DIM
        k_normed = (k * tl.rsqrt(k_var + EPS) * (w_k + 1.0)).to(out_dtype)
        k_out_off = pid * HEAD_DIM + cols
        tl.store(K_out_ptr + k_out_off, k_normed, mask=mask)
```

python/sglang/srt/models/qwen3_5.py

Qwen3.5 模型入口，新增 elif _is_hip 分支调用融合核函数，是实际产生性能收益的调用点

```
def _apply_qk_norm(self, q, k):
```

```
    if self.alt_stream is not None and get_is_capture_mode():
        # CUDA graph 捕获场景的 alt_stream 重叠
        current_stream = torch.cuda.current_stream()
        self.alt_stream.wait_stream(current_stream)
        q_by_head = q.reshape(-1, self.head_dim)
        q_by_head = self.q_norm(q_by_head)
        with torch.cuda.stream(self.alt_stream):
            k_by_head = k.reshape(-1, self.head_dim)
            k_by_head = self.k_norm(k_by_head)
        current_stream.wait_stream(self.alt_stream)
    elif _is_hip:
        # ROCm 专用融合核函数路径
        q_by_head, k_by_head = fused_qk_gemma_rmsnorm(
            q, k,
            self.q_norm.weight.data,
            self.k_norm.weight.data,
            self.q_norm.variance_epsilon,
            self.head_dim,
        )
    else:
        # 常规逐层计算（CUDA 默认路径）
        q_by_head = q.reshape(-1, self.head_dim)
        q_by_head = self.q_norm(q_by_head)
        k_by_head = k.reshape(-1, self.head_dim)
        k_by_head = self.k_norm(k_by_head)
    q = q_by_head.view(q.shape)
    k = k_by_head.view(k.shape)
    return q, k
```

评论区精华

数据类型硬编码风险

gemini-code-assist[bot] 指出核函数输出类型基于 FP16 布尔标志硬编码为 float16 或 bfloat16，若输入为 float32 会导致 tl.store 写入半精度长度，造成数据损坏。作者未公开回复，但已被 HaiShaw 通过 @kkHuang-amd 标记关注。

reshape 内存拷贝争议

gemini-code-assist[bot] 质疑文档中声称“通过传递步长避免 contiguetry 拷贝”，但 q.reshape(-1, head_dim) 仍可能触发拷贝（若 tensor 非连续），建议直接操作原始高维 tensor 或显式处理 contiguity。此项亦无公开答复。

审核状态

HaiShaw 最终给出 Approved，但 bot 的两条评论未得到 resolv e，属已合并但遗留的未解决疑虑。

- 内核输出类型硬编码导致 float32 数据损坏 (correctness): 未解决，作者未回应，PR 已合并

- reshape 隐含内存拷贝与文档声称不符 (performance): 未解决，PR 已合并

风险与影响

- 风险：
  1. 数据类型推断缺陷：若模型某个权重的 dtype 不是 float16/bfloat16 唯一二元选择（例如输入 float32），核函数将产生错误输出，且不易被常见测试捕获（精度下降可能不明显）。文件：utils.py 中 _fused_qk_gemma_rmsnorm_kernel 的 out_dtype 赋值。
  2. reshape 隐含拷贝：虽然核函数支持非连续读取，但 fused_qk_gemma_rmsnorm 内调用 q.reshape(-1, head_dim) 会触发内存重新排列，与文档声称的“避免拷贝”相悖，可能在高并发时引入意外内存开销。文件：utils.py 的 fused_qk_gemma_rmsnorm 函数。
  3. 仅覆盖 Qwen3.5 模型：_apply_qk_norm 耦合在 Qwen3.5 子类中，若其他模型也使用 GemmaRMSNorm，无法复用此融合内核。 - 影响：用户影响：AMD ROCm 平台使用 Qwen3.5 模型的用户可获得 1.9%-2.6% 低并发吞吐提升，高并发无明显改善。无 API 变化。

系统影响：无配置变更，仅修改模型前向路径。ROCm 路径新增一个 triton 核函数编译，可能增加首次启动耗时，但不影响 CUDA 侧。

团队影响：核函数未附带单元测试，仅依赖端到端精度验证；数据类型问题需后续跟进修复。

- 风险标记：数据类型硬编码风险，reshape 隐含拷贝，缺少单元测试，仅覆盖 Qwen3.5 模型

关联脉络

- PR #23620 [AMD] Optimize MiniMax-M2.5 - enable fused Triton kernel for FP8 KV cache write in aiter decode path: 同为 AMD 平台 Triton 内核融合优化，共享性能优化模式
- PR #23611 [AMD] Optimize MiniMax-M2.5 - use aiter biased_grouped_topk for sigmoid scoring in MoE routing: 同为 AMD 性能优化系列 PR，体现团队在 ROCm 上持续投入 fusion 优化