PR #23925 完整报告

sgl-project/sglang

[NPU]use triton split_qkvgate_gemma_rmsnorm_rope for Qwen3.5 and Qwen3_next

合并时间：2026-05-20 20:22

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/23925

执行摘要

- 一句话：为 NPU 上 Qwen3.5/Next 引入融合 Triton 注意力预处理核
- 推荐动作：该 PR 值得精读，尤其是 mrope.py 中新 Triton kernel 的设计（如何通过条件掩码实现 interleaved 数据选择）以及 forward_prepare_npu 中融合 kernel 的调用模式。评论区的 stride 修正讨论也值得关注，能帮助理解 Triton 中多维度 stride 的正确用法。建议在后续类似 fusion 工作中参考此设计。

功能与动机

PR body 指出：'we implemented a kernel fusion for the attention layer's split_qkv, norm, and embedding_rope operations into a single Triton kernel to boost performance.' 通过 profile 结果可见，融合后第一个 attention 层相比之前有明显加速，后续层因复用 cos/sin 缓存获得更大收益。

实现拆解

1. 模型层重构（qwen3_5.py, qwen3_next.py）：将原来的 self_attention 方法拆分为 forward_prepare_native 和 forward_prepare_npu。forward_prepare_native 保持原有分步逻辑（先 QKV 投影，再 split，再 norm，再 RoPE）；forward_prepare_npu 调用来自 sgl_kernel_npu 的融合 kernel split_qkvgate_gemma_rmsnorm_rope，一步完成 split、norm、RoPE。在 self_attention 入口处根据硬件平台（_is_npu）、推理模式（是否为 extend/draft extend）以及是否带 output gate 动态选择路径，非 NPU 或不适合融合的场景回退至 native 路径。
2. 融合 Kernel 调用：forward_prepare_npu 中，仅在第一个完整 attention 层（self.attn.layer_id == self.config.full_attention_interval - 1）时调用 rotary_emb.get_cos_sin_with_position(positions) 预计算 cos/sin，后续层直接复用上一次缓存的 position_cos 和 position_sin，避免重复计算。融合 kernel 接受 QKV 输出、cos/sin、各项维度参数及 Q/K norm 的 weight/eps，一次性输出 Q、K、V 和 gate。
3. Triton 加速 interleaved RoPE（mrope.py）：新增 Triton JIT kernel apply_interleaved_rope_kernel 及其包装函数 apply_interleaved_rope_triton，用于多模态 RoPE 中按 interleaved 模式重新组织 cos/sin 向量。该 kernel 通过二维 grid 并行处理 sequence 和 dimension，对每个 dim 根据其索引所属 section 从对应模态的向量中加载数据。同时在 get_cos_sin_with_position 中根据 support_triton 条件动态选择 Triton 版本或原生 PyTorch 版本。

4. 配套调整：未增加新测试文件，但 PR body 附带了 accuracy 和 speed 基准测试结果。

关键文件：

- python/sglang/srt/models/qwen3_5.py（模块 模型实现；类别 source；类型 core-logic；符号 self_attention, forward_prepare_native, forward_prepare_npu）：核心模型文件，重构 self_attention，拆分 native 与 NPU 两条路径，NPU 路径调用融合 kernel 一步完成 split_qkv、rmsnorm、rope，是本次性能优化的主要载体。
- python/sglang/srt/models/qwen3_next.py（模块 模型实现；类别 source；类型 core-logic；符号 self_attention, forward_prepare_native, forward_prepare_npu）：与 qwen3_5.py 高度同步的模型文件，应用相同的 native/NPU 路径拆分和融合 kernel 调用。
- python/sglang/srt/layers/rotary_embedding/mrope.py（模块 旋转编码；类别 source；类型 core-logic；符号 apply_interleaved_rope_kernel, apply_interleaved_rope_triton）：新增 Triton JIT kernel 用于加速 apply_interleaved_rope，是多模态 RoPE 的关键算子，提升 interleaved 模式下 cos/sin 重组的性能。

关键符号：apply_interleaved_rope_kernel, apply_interleaved_rope_triton, forward_prepare_native, forward_prepare_npu, self_attention, get_cos_sin_with_position

关键源码片段

python/sglang/srt/models/qwen3_5.py

核心模型文件，重构 self_attention，拆分 native 与 NPU 两条路径，NPU 路径调用融合 kernel 一步完成 split_qkv、rmsnorm、rope，是本次性能优化的主要载体。

```
def self_attention(
    self,
    positions: torch.Tensor,
    hidden_states: torch.Tensor,
    forward_batch: ForwardBatch,
) -> torch.Tensor:
    """Full attention forward pass with dynamic path selection."""
    # 根据平台和模式选择原生或 NPU 融合路径
    if (
        not _is_npu
        or forward_batch.forward_mode.is_extend_or_draft_extend_or_mixed()
        or not self.attn_output_gate
    ):
        q, k, v, gate = self.forward_prepare_native(
            positions=positions,
            hidden_states=hidden_states,
        )
    else:
        q, k, v, gate = self.forward_prepare_npu(
            positions=positions,
            hidden_states=hidden_states,
            forward_batch=forward_batch,
        )
```

```
    attn_output = self.attn(q, k, v, forward_batch)

    if self.attn_output_gate:
        gate = torch.sigmoid(gate)
        attn_output = attn_output * gate

    output, _ = self.o_proj(attn_output)
    return output


def forward_prepare_npu(self, positions, hidden_states, forward_batch):
    """NPU-optimized forward preparation: fused split_qkv + rmsnorm + rope."""
    qkv, _ = self.qkv_proj(hidden_states)
    # 仅第一个完整 attention 层计算 cos/sin，后续层复用缓存
    if self.attn.layer_id == (self.config.full_attention_interval - 1):
        self.rotary_emb.get_cos_sin_with_position(positions)

    q, k, v, gate = split_qkvgate_gemma_rmsnorm_rope(
        qkv,
        self.rotary_emb.position_sin,
        self.rotary_emb.position_cos,
        self.q_size,
        self.kv_size,
        self.head_dim,
        int(self.head_dim * self.partial_rotary_factor),
        eps=self.q_norm.variance_epsilon,
        q_weight=self.q_norm.weight,
        k_weight=self.k_norm.weight,
    )
    return q, k, v, gate
```

python/sglang/srt/layers/rotary_embedding/mrope.py

新增 Triton JIT kernel 用于加速 apply_interleaved_rope，是多模态 RoPE 的关键算子，提升 interleaved 模式下 cos/sin 重组的性能。

```
@triton.jit
def apply_interleaved_rope_kernel(
    x_ptr, out_ptr,
    S: tl.constexpr, D: tl.constexpr,
    stride_x_m, stride_x_s, stride_out_s,
    section_1_end, section_2_end,
    BLOCK_S: tl.constexpr, BLOCK_SIZE: tl.constexpr,
):
    # 按序列和维度两个维度进行并行
    start_s = tl.program_id(0) * BLOCK_S
    s_offsets = start_s + tl.arange(0, BLOCK_S)
    dim_offset = tl.program_id(1) * BLOCK_SIZE
    dim_indices = dim_offset + tl.arange(0, BLOCK_SIZE)
    mask = (s_offsets[:, None] < S) & (dim_indices[None, :] < D)
```

```
    # 加载第 0 个模态的值（默认）
    val = tl.load(x_ptr + 0 * stride_x_m + s_offsets[:, None] * stride_x_s + dim_indices[None, :],
                  mask=mask, other=0.0)

    # 对于第 1 模态（索引 %3==1 且在 section_1 范围内）覆盖为 val_a
    cond_a = (dim_indices[None, :] % 3 == 1) & (dim_indices[None, :] < section_1_end * 3)
    val_a = tl.load(x_ptr + 1 * stride_x_m + s_offsets[:, None] * stride_x_s + dim_indices[None, :],
                    mask=mask & cond_a, other=0.0)
    val = tl.where(cond_a, val_a, val)

    # 对于第 2 模态（索引 %3==2 且在 section_2 范围内）覆盖为 val_b
    cond_b = (dim_indices[None, :] % 3 == 2) & (dim_indices[None, :] < section_2_end * 3)
    val_b = tl.load(x_ptr + 2 * stride_x_m + s_offsets[:, None] * stride_x_s + dim_indices[None, :],
                    mask=mask & cond_b, other=0.0)
    val = tl.where(cond_b, val_b, val)

    # 写入输出
    tl.store(out_ptr + s_offsets[:, None] * stride_out_s + dim_indices[None, :], val, mask=mask)
```

评论区精华

1. Triton kernel stride 计算错误（Critical）：review 发现 apply_interleaved_rope_kernel 最初使用 stride_x_s（sequence 维度 stride）来访问不同模态分量，这会导致访问到错误的 sequence 元素。最终代码修正为使用 stride_x_m（第一维 stride），确保正确从第 1、2 个模态分量读取数据。
2. 跳过位置编码更新的硬编码问题（High）：原先的 forward_prepare_npu 中直接硬编码了 layer ID 0 和 3 来决定是否调用 get_cos_sin_with_position，这在多 layer 且 rotary_emb 非共享时会导致未初始化崩溃。review 后改为使用 self.config.full_attention_interval 动态计算完整 attention 层的序号（self.attn.layer_id == self.config.full_attention_interval - 1），同时确保只有第一个 attention 层才计算 cos/sin，后续层复用。
3. iforgetmyname 建议使用配置：作者接受并实现了 use self.config.full_attention_interval directly 的建议。

- Triton kernel stride 计算错误 (correctness): 最终代码在 kernel 参数中增加 stride_x_m，并在访问分量时使用该 stride，确认修复。
- NPU 路径中跳过 get_cos_sin_with_position 的逻辑问题 (correctness): 最终代码改用 self.config.full_attention_interval 动态识别第一个 full attention 层（self.attn.layer_id == self.config.full_attention_interval - 1），确保只在该层计算 cos/sin，后续层复用。

风险与影响

- 风险：
  1. Triton kernel 正确性：apply_interleaved_rope_kernel 对 stride 计算敏感，虽然已修正，但若未来输入张量 layout 变化（如非 contiguous）仍可能出错。代码中已调用 .contiguous() 前置转换，降低此风险。

2. NPU 平台依赖：融合 kernel split_qkvgate_gemma_rmsnorm_rope 来自 sgl_kernel_npu，该包仅在 NPU 可用。若 NPU 环境缺少该包，导入时即错误。在非 NPU 环境中通过 _is_npu 条件守卫，不会触发导入。
3. cos/sin 缓存共享：forward_prepare_npu 假设 rotary_emb 是每个 attention 层的独立实例且 position_cos/position_sin 被本层复用。若未来实现跨层共享或重置，需重新审视缓存逻辑。
4. 缺少测试覆盖：本 PR 未附带单元测试，仅依赖 CI 和手动 benchmark 验证。 - 影响：对用户：NPU 上运行 Qwen3.5/Qwen3_next 模型时，推理速度得到提升，尤其对于长序列和 decode 阶段收益明显。对其他硬件（CUDA、CPU）无影响。对系统：引入新的 sgl_kernel_npu 依赖（条件导入），不影响现有 CUDA 路径。对团队：展示了 kernel fusion 在 NPU 上的可行方法，后续可推广到其他模型。影响程度中等。 - 风险标记：Triton kernel 偏移计算敏感，NPU 平台特有路径，依赖 sgl_kernel_npu，缺少测试覆盖

关联脉络

- 暂无明显关联 PR