PR #24696 完整报告

sgl-project/sglang

[Gemma4] Optimize Gemm4 with fused Q/K/V RMSNorm + per-expert FP8 ckpt loader

合并时间：2026-05-10 15:24

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/24696

执行摘要

- 一句话：融合 QKV RMSNorm 并修复 FP8 MoE 权重加载
- 推荐动作：此 PR 包含两个值得关注的设计：融合 Triton 内核使用 stride view 避免拷贝，以及保守的 fallback 策略；加载器中的正则映射模式可复用于其他支持 per-expert 格式的模型。测试用例的三阶段设计（健康、非垃圾、精度）提供良好的回归保护。

功能与动机

原始动机是加速 Gemma4 模型。两个独立问题：

1. Attention 的 Q/K/V RMSNorm 每次 launch 三个独立内核，小 token 数时 host 开销占主导；
2. 压缩张量 per-expert FP8 检查点（如 RedHatAI/gemma-4-26B-A4B-it-FP8-Dynamic）中专家权重以单独键存储，现有加载器静默跳过，导致生成全为 ，GSM8K 降为 0.0。

实现拆解

1. 在 python/sglang/srt/layers/gemma4_fused_ops.py 中新增 _gemma_qkv_rmsnorm_kernel Triton JIT 内核，通过 tl.static_range 遍历 Q/K/V 头数，单次 launch 完成所有归一化；同时增加 gemma_qkv_rmsnorm 入口函数，接受可选 K、V 支持 KV 共享层。
2. 在 python/sglang/srt/models/gemma4_causal.py 的 Gemma4Attention.forward 中，检查条件（CUDA 设备、标准 scale_shift、V 无权重）决定使用融合路径还是原有三次 norm 路径。
3. 在 python/sglang/srt/models/gemma4_mm.py 的 Gemma4ForConditionalGeneration.load_weights 中新增正则分支，匹配 ^.*\.moe\.experts\.<id>\.(gate|up|down)_proj\.(weight|weight_scale)$，映射到 FusedMoE 的 w13_weight/w2_weight 参数和分片。
4. 新增 test/registered/models/test_gemma4_fp8_per_expert_loading.py，启动 TP=4 服务器后依次检查健康状态、生成不含 、GSM8K-20 准确率≥0.80。

关键文件：

- python/sglang/srt/layers/gemma4_fused_ops.py（模块 融合算子；类别 source；类型 core-logic；符号 _gemma_qkv_rmsnorm_kernel, gemma_qkv_rmsnorm）：核心变更，新

增融合 QKV RMSNorm 的 Triton 内核和 Python 入口函数，是实现性能优化的关键。

- python/sglang/srt/models/gemma4_causal.py（模块 注意力层；类别 source；类型 core-logic；符号 Gemma4Attention.forward）：在注意力层 forward 中集成融合 RMSNorm 调用，并保留回退路径。
- python/sglang/srt/models/gemma4_mm.py（模块 权重加载；类别 source；类型 data-contract；符号 Gemma4ForConditionalGeneration.load_weights）：在 load_weights 中新增 per-expert FP8 检查点的正则加载分支，修复静默跳过 bug。
- test/registered/models/test_gemma4_fp8_per_expert_loading.py（模块 测试；类别 test；类型 test-coverage；符号 TestGemma4FP8PerExpertLoading, setUpClass, tearDownClass, test_health）：新增端到端测试，覆盖 per-expert FP8 加载和模型推理质量，三阶段断言检测修复是否生效。

关键符号：_gemma_qkv_rmsnorm_kernel, gemma_qkv_rmsnorm, Gemma4Attention.forward, Gemma4ForConditionalGeneration.load_weights

关键源码片段

python/sglang/srt/layers/gemma4_fused_ops.py

核心变更，新增融合 QKV RMSNorm 的 Triton 内核和 Python 入口函数，是实现性能优化的关键。

```
# 新增的融合 Q/K/V RMSNorm Triton 内核
# 一次 launch 完成 Q、K、V 三个头组的 RMSNorm，
# 避免三个独立 kernel 的 launch 开销
@triton.jit
def _gemma_qkv_rmsnorm_kernel(
    Q_ptr, K_ptr, V_ptr,
    Q_w_ptr, K_w_ptr,
    stride_q_m, stride_k_m, stride_v_m,
    NUM_Q_HEADS: tl.constexpr,
    NUM_KV_HEADS: tl.constexpr,
    HEAD_DIM: tl.constexpr,
    eps, HAS_KV: tl.constexpr, BLOCK: tl.constexpr,
):
    m = tl.program_id(0)
    cols = tl.arange(0, BLOCK)
    mask = cols < HEAD_DIM

    qw = tl.load(Q_w_ptr + cols, mask=mask, other=0.0).to(tl.float32)

    # Q 头组
    for h in tl.static_range(NUM_Q_HEADS):
        off = m * stride_q_m + h * HEAD_DIM + cols
        x = tl.load(Q_ptr + off, mask=mask, other=0.0).to(tl.float32)
        rrms = tl.rsqrt(tl.sum(x * x, axis=0) / HEAD_DIM + eps)
        # Q 乘以 q weight
        out = x * rrms * qw
```

```
        tl.store(Q_ptr + off, out.to(Q_ptr.dtype.element_ty), mask=mask)

    if HAS_KV:
        kw = tl.load(K_w_ptr + cols, mask=mask, other=0.0).to(tl.float32)
        # K 头组
        for h in tl.static_range(NUM_KV_HEADS):
            off = m * stride_k_m + h * HEAD_DIM + cols
            x = tl.load(K_ptr + off, mask=mask, other=0.0).to(tl.float32)
            rrms = tl.rsqrt(tl.sum(x * x, axis=0) / HEAD_DIM + eps)
            out = x * rrms * kw
            tl.store(K_ptr + off, out.to(K_ptr.dtype.element_ty), mask=mask)
        # V 头组（无 weight：Gemma4 的 V norm 使用 weight=ones）
        for h in tl.static_range(NUM_KV_HEADS):
            off = m * stride_v_m + h * HEAD_DIM + cols
            x = tl.load(V_ptr + off, mask=mask, other=0.0).to(tl.float32)
            rrms = tl.rsqrt(tl.sum(x * x, axis=0) / HEAD_DIM + eps)
            out = x * rrms
            tl.store(V_ptr + off, out.to(V_ptr.dtype.element_ty), mask=mask)


def gemma_qkv_rmsnorm(
    q: torch.Tensor,
    k: Optional[torch.Tensor],
    v: Optional[torch.Tensor],
    q_weight: torch.Tensor,
    k_weight: Optional[torch.Tensor],
    num_q_heads: int,
    num_kv_heads: int,
    head_dim: int,
    eps: float = 1e-6,
) -> None:
    # 融合入口函数，支持在 qkv strided view 上原位操作
    ...
```

python/sglang/srt/models/gemma4_causal.py

在注意力层 forward 中集成融合 RMSNorm 调用，并保留回退路径。

```
# Gemma4Attention.forward 中的关键变更
qkv, _ = self.qkv_proj(hidden_states)
q, k, v = qkv.split([self.q_size, self.kv_size, self.kv_size], dim=-1)

# 检查是否可以走融合路径
is_kv_shared = (
    self.is_kv_shared_layer and self.kv_shared_layer_index is not None
)
can_fuse_qkv_norm = (
    q.is_cuda
    and self.q_norm.scale_shift == 0.0
    and self.k_norm.scale_shift == 0.0
```

```
        and not self.v_norm.with_scale
    )
    if can_fuse_qkv_norm:
        if is_kv_shared:
            # KV 共享层只处理 Q，K/V 参数传递 None
            gemma_qkv_rmsnorm(
                q, None, None,
                self.q_norm.weight.data, None,
                num_q_heads=self.num_heads, num_kv_heads=self.num_kv_heads,
                head_dim=self.head_dim, eps=self.q_norm.eps,
            )
            k = None
            v = None
        else:
            # 完整融合 Q、K、V
            gemma_qkv_rmsnorm(
                q, k, v,
                self.q_norm.weight.data, self.k_norm.weight.data,
                num_q_heads=self.num_heads, num_kv_heads=self.num_kv_heads,
                head_dim=self.head_dim, eps=self.q_norm.eps,
            )
            # 调整维度以便后续 flatten
            k = k.reshape(-1, self.num_kv_heads, self.head_dim)
            v = v.reshape(-1, self.num_kv_heads, self.head_dim)
    else:
        # 非标准配置回退到原有三次 norm 路径
        q = q.unflatten(-1, (self.num_heads, self.head_dim))
        q = self.q_norm(q)
        q = q.flatten(-2, -1)
        if is_kv_shared:
            k = v = None
        else:
            k = k.unflatten(-1, (self.num_kv_heads, self.head_dim))
            k = self.k_norm(k)
            v = v.unflatten(-1, (self.num_kv_heads, self.head_dim))
            v = self.v_norm(v)
    # 后续 RoPE 和 attention 处理不变
```

python/sglang/srt/models/gemma4_mm.py

在 load_weights 中新增 per-expert FP8 检查点的正则加载分支，修复静默跳过 bug。

```
    # load_weights 中的新增正则分支
    # 匹配格式：experts.<id>.gate_proj.weight, experts.<id>.gate_proj.weight_scale 等
    per_expert_match = re.match(
        r"^(.*?\.moe\.experts\.)(\d+)\.(gate_proj|up_proj|down_proj)"
        r"\.(weight|weight_scale)$",
        name,
    )
    if per_expert_match:
```

```
    prefix = per_expert_match.group(1) # 路径前缀
    expert_id = int(per_expert_match.group(2))
    proj = per_expert_match.group(3) # 投影名称
    suffix = per_expert_match.group(4) # weight 或 weight_scale

    # 映射到 FusedMoE 的参数名和分片 ID
    if proj == "gate_proj":
        base, sid = "w13_weight", "w1"
    elif proj == "up_proj":
        base, sid = "w13_weight", "w3"
    else: # down_proj
        base, sid = "w2_weight", "w2"
    if suffix == "weight_scale":
        base += "_scale"

    fused_name = prefix + base
    if fused_name in params_dict:
        param = params_dict[fused_name]
        weight_loader = param.weight_loader
        # 调用 FusedMoE 的 weight_loader，传递分片和 expert id
        weight_loader(param, loaded_weight, fused_name, sid, expert_id)
        loaded_params.add(fused_name)
    continue # 跳过后续映射
```

评论区精华

pyc96 指出 gemma4_mm.py 中内联 logger.warning 可以移除，让缺失参数通过中央 unloaded_params 检查统一记录，yuan-luo 采纳并修改。pyc96 还要求验证质量分数（MMLU），作者回复更新了 MMLU 结果（总体 0.885，STEM 0.952）。

- 加载路径中的日志处理 (design): yuan-luo 采纳并移除了内联 logger.warning，让参数缺失通过通用机制处理。
- 质量分数验证 (testing): yuan-luo 回复更新了 MMLU 分数，显示总体 0.885，STEM 0.952 等，确保了模型质量。

风险与影响

- 风险：融合 kernel 仅在满足严格条件时启用（scale_shift==0 且 V 无权重），非标准配置自动回退原路径，无正确性风险。加载器正则表达式假设检查点格式与当前一致，若未来格式变更可能导致静默跳过但会被 unloaded_params 捕获。测试仅覆盖 H100 4-GPU TP=4 场景，其他配置可能遗漏回归。
- 影响：正面影响：Gemma4 模型用户获得 prefill 阶段约 2% 延迟改善，per-expert FP8 检查点现在可用。负面影响：无已知负面影响，融合 kernel 与回退路径均保持数值一致（bit-exact 验证）。社区：修复了一个可能影响 FP8 MoE 检查点的通用问题。
- 风险标记：条件激活路径，正则假设风险，测试覆盖局限

关联脉络

- PR #23976 Support Gemma3/4 + Eagle3: 同一模型系列支持，修改了相同的 *gemma4_causal* 和 *gemma4_mm* 文件。