PR #26502 完整报告

sgl-project/sglang

perf(gemma4): single-launch fused router (topk + softmax + scale)

合并时间：2026-06-02 16:00

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/26502

执行摘要

- 一句话：融合 Gemma4 路由为单次 Triton kernel，decode 吞吐提升 5.6%
- 推荐动作：建议技术团队精读该 PR，尤其关注：(1) int64 键打包实现单次排序的设计技巧；(2) 如何通过条件判断保持与现有量化路径的兼容；(3) 将 vLLM 算法重写为 SGLang 代码风格的方法。对于非 Gemma4 用户，该 PR 虽不直接受益，但其 fused routing 模式可推广至其他 MoE 路由场景。

功能与动机

在 Gemma-4-26B 等模型中，路由函数在每次 MoE 前向中发射四个 GPU kernel，累计占用约 5.8% 的 decode GPU 时间。vLLM 的 fused kernel 将此降到 ~1.1%，本 PR 将其移植至 SGLang，以提升 Gemma4 模型的推理吞吐。

实现拆解

1. 新增融合 kernel 和封装：在 python/sglang/srt/layers/gemma4_fused_ops.py 中添加 Triton kernel _gemma4_routing_kernel 和 Python 封装函数 gemma4_fused_routing。kernel 通过将 logit 的比特表示与专家 ID 打包为 int64 键，一次 tl.sort 实现降序排列，随后完成 fp32 softmax 和 scale 乘法。
2. 修改模型路由入口：在 python/sglang/srt/models/gemma4_causal.py 的 Gemma4MoE.__init__ 中的 routing_function 内加入条件判断，对 CUDA 上的 fp16/bf16/fp32 输入调用 fused 路由，否则回退原有 torch 实现。
3. 添加单元测试：新建 test/registered/kernels/test_gemma4_fused_routing.py，通过参数化覆盖 47 种形状 / 数据类型组合，验证与参考实现的一致性；包含零 token 边界和 scale 应用验证。
4. CI 注册：在测试文件中通过 register_cuda_ci 注册到 CI 流水线，确保构建自动执行。

关键文件：

- python/sglang/srt/layers/gemma4_fused_ops.py（模块 路由内核；类别 source；类型 core-logic；符号 _gemma4_routing_kernel, gemma4_fused_routing）：核心文件，包含 Triton kernel 及其封装函数，是性能提升的直接来源。
- test/registered/kernels/test_gemma4_fused_routing.py（模块 路由测试；类别 test；类型 test-coverage；符号 fused_routing, _reference, test_matches_reference, test_zero_tokens）：新测试文件，覆盖 47 种形状 / 数据类型组合，验证 fused kernel 与

参考实现一致性，包含边界和 scale 测试。

- python/sglang/srt/models/gemma4_causal.py（模块 模型定义；类别 source；类型 entrypoint；符号 routing_function）：模型文件中的路由函数入口，新增对 fused kernel 的条件调用，作为回退保护。

关键符号：_gemma4_routing_kernel, gemma4_fused_routing, routing_function

关键源码片段

test/registered/kernels/test_gemma4_fused_routing.py

新测试文件，覆盖 47 种形状 / 数据类型组合，验证 fused kernel 与参考实现一致性，包含边界和 scale 测试。

```
def _reference(gating_output: torch.Tensor, per_expert_scale: torch.Tensor, topk: int):
    """The previous (now fallback) torch routing function from gemma4_causal.py."""
    topk_logits, topk_ids = torch.topk(gating_output, k=topk, dim=-1)
    topk_weights = torch.nn.functional.softmax(topk_logits, dim=-1)
    topk_weights = topk_weights * per_expert_scale[topk_ids].to(topk_weights.dtype)
    return topk_weights.to(torch.float32), topk_ids.to(torch.int32)


@pytest.mark.parametrize("dtype", [torch.bfloat16, torch.float16, torch.float32])
@pytest.mark.parametrize("T", [1, 7, 64, 128, 1024])
@pytest.mark.parametrize("E,K", [(128, 8), (64, 4), (256, 8)])
def test_matches_reference(fused_routing, dtype, T, E, K):
    torch.manual_seed(0)
    g = torch.randn(T, E, dtype=dtype, device="cuda")
    s = torch.rand(E, dtype=dtype, device="cuda") * 2.0

    ref_w, ref_i = _reference(g, s, K)
    out_w, out_i = fused_routing(g, s, K)

    assert out_w.dtype == torch.float32
    assert out_i.dtype == torch.int32
    assert out_w.shape == (T, K)
    assert out_i.shape == (T, K)

    # 根据输入 dtype 设置容忍度（fused kernel 使用 fp32 softmax，参考使用输入 dtype）
    if dtype == torch.bfloat16:
        atol, rtol = 5e-3, 5e-3
    elif dtype == torch.float16:
        atol, rtol = 1e-3, 1e-3
    else:
        atol, rtol = 1e-5, 1e-5

    if (out_i != ref_i).any():
        # tie-break 可能导致顺序不同，但集合必须相同
        ref_set = ref_i.sort(dim=-1).values
        out_set = out_i.sort(dim=-1).values
```

```
        assert torch.equal(out_set, ref_set), "fused routing picked a different top-K set"
        torch.testing.assert_close(
            out_w.sum(dim=-1).to(torch.float32),
            ref_w.sum(dim=-1).to(torch.float32),
            atol=atol, rtol=rtol,
        )
    else:
        torch.testing.assert_close(out_w, ref_w, atol=atol, rtol=rtol)


def test_scale_applied(fused_routing):
    """Weights must include per_expert_scale[topk_ids]."""
    torch.manual_seed(1)
    T, E, K = 4, 128, 8
    g = torch.randn(T, E, dtype=torch.bfloat16, device="cuda")
    s = torch.rand(E, dtype=torch.bfloat16, device="cuda") * 3.0

    out_w, out_i = fused_routing(g, s, K)
    ref_w, ref_i = _reference(g, s, K)
    torch.testing.assert_close(out_w, ref_w, atol=5e-3, rtol=5e-3)
    assert torch.equal(out_i, ref_i)
```

python/sglang/srt/models/gemma4_causal.py

模型文件中的路由函数入口，新增对 fused kernel 的条件调用，作为回退保护。

```
def routing_function(
    hidden_states: torch.Tensor,
    gating_output: torch.Tensor,
    topk: int,
    renormalize: bool, # Gemma4 始终 True；softmax 恒等式仅在重归一化时成立
) -> tuple[torch.Tensor, torch.Tensor]:
    # 快速路径：如果输入是 CUDA 且 dtype 支持，调用单个 Triton kernel
    if (
        gating_output.is_cuda
        and gating_output.dim() == 2
        and gating_output.dtype
        in (torch.float16, torch.bfloat16, torch.float32)
    ):
        return gemma4_fused_routing(gating_output, per_expert_scale, topk)

    # 回退路径：使用 torch.topk + softmax + scale
    topk_logits, topk_ids = torch.topk(gating_output, k=topk, dim=-1)
    topk_weights = torch.nn.functional.softmax(topk_logits, dim=-1)
    topk_weights = topk_weights * per_expert_scale[topk_ids].to(topk_weights.dtype)
    return topk_weights.to(torch.float32), topk_ids.to(torch.int32)
```

评论区精华

Review 讨论聚焦在代码风格和兼容性上：(1) BBuf 建议将 tl.exp2 替换为 tl.exp，因为无须兼容过旧 Triton 版本；(2) 为 assert 添加失败说明信息；(3) 移除不必要的内联注释。作者均予以采纳，无未解决争议。

- 使用 tl.exp 替代 tl.exp2 (style): 作者采纳，改为 tl.exp。
- 添加 assert 失败信息 (style): 作者添加了描述信息。
- 移除不必要的注释 (style): 作者移除。

风险与影响

- 风险：主要风险在 kernel 对 GPU 架构的依赖及精度一致性。但已通过以下方式控制：(1) 仅在 CUDA 且 dtype 为 fp16/bf16/fp32 时启用快速路径，否则安全回退；(2) E ≤ 1024 的断言保护 warp 假设；(3) 47 个单元测试、随机等价性探针及全量 MMLU 评估均证实与旧实现行为一致。量化兼容性方面，仅当路由函数未被 bypass 时触发，无副作用。整体风险较低。
- 影响：对用户：Gemma4 模型解码吞吐提升 4-6%，首 token 延迟不受影响，token 间延迟略有降低。对系统：无额外内存或启动开销（num_warps=1）。对团队：新增约 110 行 Python+Triton 代码和维护负担，但 kernel 粒度小、逻辑清晰。测试覆盖充分，回归风险低。
- 风险标记：GPU 专属，E≤1024，测试充分，回退保障

关联脉络

- 暂无明显关联 PR