PR #25835 完整报告

sgl-project/sglang

[JIT Kernel] Triton moe fused gate

合并时间：2026-06-30 23:26

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/25835

执行摘要

- 一句话：新增 Triton MoE 融合门控内核，替换 CUDA JIT
- 推荐动作：值得精读的 PR，尤其是 Triton 内核的迭代 top-k 选择策略和 PDL 用法。在采用前应关注数值回归问题，并确认模型准确性。设计上融合了正确性、性能和可维护性，但模型级测试的缺失是当前主要薄弱环节。

功能与动机

现有的 MoE 路由内核有 AOT CUDA 和 JIT CUDA 两个版本，维护成本高且 AOT 内核仅支持 sigmoid 评分函数、组大小限制为 32。Triton 实现可灵活支持任意评分函数和专家数，同时具备更好的跨平台兼容性和性能。PR 给出的性能数据显示 Triton 版本相比 JIT CUDA 快约 2 倍，且全面超越 AOT 和 PyTorch 实现。

实现拆解

1. 新增 Triton 内核：在 python/sglang/jit_kernel/moe_fused_gate.py 中添加 _router_triton_kernel，实现完整的门控逻辑：加载 scores 和 bias，根据评分函数计算激活值（sigmoid 或 sqrt(softplus)），加 bias，通过迭代 argmax 和屏蔽已选专家选择 top-k 路由专家，处理 fused shared experts，重归一化和缩放。同一文件中将原有 JIT CUDA 函数 moe_fused_gate 重命名为 moe_fused_gate_jit，并新增统一入口 moe_fused_gate（当前调用 Triton 版本）。
2. 添加单元测试：在 test/registered/jit/test_moe_fused_gate.py 中编写全面的参数化测试，验证 Triton 实现与参考实现、CUDA JIT 实现、生产实现 biased_grouped_topk_impl 的一致性。测试采用散列方式使比较次序无关。
3. 添加基准测试：test/registered/jit/benchmark/bench_moe_fused_gate.py 使用 jit_kernel benchmark marker 框架，对比 Triton、JIT CUDA、AOT CUDA 和 PyTorch 实现在不同专家数和 token 数下的延迟。
4. 集成到推理路径：修改 python/sglang/srt/layers/moe/topk.py 中的 biased_grouped_topk_gpu，当 num_expert_group == 1 时（对应 DeepSeek-V4 / Kimi-K2 的 ungrouped 模式），调用新的 Triton 入口 moe_fused_gate 替换原有的 AOT kimi_k2_moe_fused_gate。同时调整函数签名，删除旧的 AOT 导入。
5. 修复 CUDA JIT 内核边界 Bug：在 moe_fused_gate.cuh 中修复 moe_fused_gate_kernel_small_token 的 warp 读取问题，确保仅读取实际启动的 warp，

避免读取未初始化的共享内存。

关键文件：

- python/sglang/jit_kernel/moe_fused_gate.py（模块 门控内核；类别 source；类型 core-logic；符号 moe_fused_gate, moe_fused_gate_jit, _router_triton_kernel）：核心文件：添加 Triton 路由内核 _router_triton_kernel、统一入口 moe_fused_gate，并将原 CUDA JIT 重命名为 moe_fused_gate_jit。
- test/registered/jit/test_moe_fused_gate.py（模块 门控测试；类别 test；类型 test-coverage；符号 _scatter_by_expert, _reference_gate, _make_inputs, test_moe_fused_gate_matches_reference）：深度单元测试：通过三种参照验证 Triton 内核正确性。
- test/registered/jit/benchmark/bench_moe_fused_gate.py（模块 性能基准；类别 test；类型 test-coverage；符号 torch_router, benchmark）：基准测试：在多种 token/ 专家数下对比 Triton、JIT、AOT、PyTorch 性能。
- python/sglang/srt/layers/moe/topk.py（模块 路由层；类别 source；类型 dependency-wiring；符号 biased_grouped_topk_gpu, biased_topk_jit_kernel_impl）：集成点：在 biased_grouped_topk_gpu 中替换 AOT 内核为 Triton 实现，删除过期导入。
- python/sglang/jit_kernel/csrc/moe/moe_fused_gate.cuh（模块 CUDA 内核；类别 other；类型 core-logic）：修复 small-token kernel 的 warp 边界读取问题。

关键符号：_router_triton_kernel, moe_fused_gate, moe_fused_gate_jit, biased_grouped_topk_gpu, _reference_gate, _scatter_by_expert, _make_inputs, test_moe_fused_gate_matches_reference, test_moe_fused_gate_matches_cuda_jit, test_moe_fused_gate_matches_production_impl, test_moe_fused_gate_shapes_and_dtypes, torch_router, benchmark

关键源码片段

python/sglang/jit_kernel/moe_fused_gate.py

核心文件：添加 Triton 路由内核 _router_triton_kernel、统一入口 moe_fused_gate，并将原 CUDA JIT 重命名为 moe_fused_gate_jit。

```
# 入口函数 moe_fused_gate（Triton 版本）
from sglang.jit_kernel.utils import is_arch_support_pdl


def moe_fused_gate(
    input: torch.Tensor, bias: torch.Tensor, topk: int,
    scoring_func: str = "sigmoid", num_fused_shared_experts: int = 0,
    renormalize: bool = True, routed_scaling_factor: float = 1.0,
    apply_routed_scaling_factor_on_output: bool = False,
) -> Tuple[torch.Tensor, torch.Tensor]:
    # 验证输入合法性
    assert input.dtype == torch.float32
    assert bias.dtype == torch.float32
    # ... 省略其他断言
    num_rows, num_experts = input.shape
```

```
    output = torch.empty(num_rows, topk, dtype=torch.float32, device=input.device)
    indices = torch.empty(num_rows, topk, dtype=torch.int32, device=input.device)

    # 选择 PDL 启用取决于架构支持
    use_pdl = is_arch_support_pdl()
    # 确定 BLOCK_N 和 BLOCK_K（为 2 的幂）
    BLOCK_N = triton.next_power_of_2(num_experts)
    BLOCK_K = triton.next_power_of_2(topk)

    # 启动 Triton kernel，每个程序处理一行
    _router_triton_kernel[(num_rows,)](
        input, bias, output, indices,
        num_rows, routed_scaling_factor,
        num_experts, topk, topk - num_fused_shared_experts,
        BLOCK_N, BLOCK_K,
        scoring_func_int, renormalize,
        apply_routed_scaling_factor_on_output, use_pdl,
        input.stride(0), input.stride(1),
        output.stride(0), output.stride(1),
        indices.stride(0), indices.stride(1),
    )
    return output, indices

# Triton kernel 核心（节选：迭代 top-k 选择）
@triton.jit
def _router_triton_kernel(_, _, _, _, M, _, N, K, K_ROUTED, BLOCK_N, BLOCK_K,
                          SCORING_FUNC, RENORMALIZE, APPLY_SCALE, USE_PDL,
                          stride_sm, stride_sn, stride_wm, stride_wk, stride_im, stride_ik):
    pid = tl.program_id(0)
    if pid >= M: return
    offs_n = tl.arange(0, BLOCK_N)
    mask_n = offs_n < N
    bias = tl.load(bias_ptr + offs_n, mask=mask_n, other=0.0).to(tl.float32)
    if USE_PDL:
        tl.extra.cuda.gdc_wait() # 等待上游依赖（例如 GEMM）完成
    scores = tl.load(scores_ptr + pid * stride_sm + offs_n * stride_sn, mask=mask_n, other=0.0).
    to(tl.float32)
    # 评分函数：sigmoid 或 sqrt(softplus)
    if SCORING_FUNC == 0:
        activated = tl.sigmoid(scores)
    else:
        sp = tl.where(scores > 20.0, scores, tl.log(1.0 + tl.exp(scores)))
        activated = tl.sqrt(sp)
    biased = activated + bias
    biased = tl.where(mask_n, biased, -float('inf'))
    # 迭代选择 top-K_ROUTED 个专家
    offs_k = tl.arange(0, BLOCK_K)
    selected_vals = tl.zeros([BLOCK_K], dtype=tl.float32)
    selected_idx = tl.zeros([BLOCK_K], dtype=tl.int32)
```

```
    cur = biased
    for k in tl.static_range(K_ROUTED):
        max_val = tl.max(cur, axis=0)
        is_max = cur == max_val
        lane_id = tl.where(is_max, offs_n, N + 1)
        win_lane = tl.min(lane_id, axis=0).to(tl.int32)
        win_activated = tl.sum(tl.where(offs_n == win_lane, activated, 0.0), axis=0)
        slot = offs_k == k
        selected_vals = tl.where(slot, win_activated, selected_vals)
        selected_idx = tl.where(slot, win_lane, selected_idx)
        cur = tl.where(offs_n == win_lane, -float('inf'), cur) # 屏蔽已选专家
    # ... 后续处理 fused shared experts 和重归一化
```

test/registered/jit/test_moe_fused_gate.py

深度单元测试：通过三种参照验证 Triton 内核正确性。

```
# 参考实现：只使用 PyTorch 操作，清晰定义门控语义
def _reference_gate(
    scores: torch.Tensor, bias: torch.Tensor, topk: int,
    scoring_func: str, num_fused_shared_experts: int,
    renormalize: bool, routed_scaling_factor: float,
    apply_routed_scaling_factor_on_output: bool,
) -> Tuple[torch.Tensor, torch.Tensor]:
    if scoring_func == "sigmoid":
        activated = scores.sigmoid()
    else:
        activated = torch.nn.functional.softplus(scores).sqrt()
    biased = activated + bias.unsqueeze(0)
    num_experts = scores.size(1)
    num_routed = topk - num_fused_shared_experts
    # 迭代 argmax 选择路由专家（匹配 Triton 算法）
    bs = biased.size(0)
    work = biased.clone()
    arange = torch.arange(num_experts, device=scores.device).unsqueeze(0)
    routed_idx = torch.empty(bs, num_routed, dtype=torch.int32, device=scores.device)
    routed_wgt = torch.empty(bs, num_routed, dtype=torch.float32, device=scores.device)
    for k in range(num_routed):
        vals, _ = work.max(dim=1, keepdim=True)
        lane = torch.where(work == vals, arange, num_experts + 1)
        winner = lane.min(dim=1).values.to(torch.int32)
        routed_idx[:, k] = winner
        routed_wgt[:, k] = activated.gather(1, winner.long().unsqueeze(1)).squeeze(1)
        work.scatter_(1, winner.long().unsqueeze(1), float("-inf")) # 屏蔽已选专家
    # 构建完整 output, indices（包括 fused shared）
    # ... 后续组装和重归一化
    return weights, indices
```

评论区精华

1. 数值回归报告：合并后用户 qiushixiaoyu 报告在 DeepSeek-V4 sqrtsoftplus 上发现准确率下降，bisect 指向此 PR，需要进一步修复。
2. PDL 内存顺序：yuan-luo 建议将 gdc_launch_dependents 放在最终 store 之后；DarkSharpness 认为下游应在 PDL wait 前完成读取，且 FlashInfer 采用类似模式。最终接受当前实现。
3. 测试覆盖：ispobock 建议增加 PDL on/off 路径的单元测试；BBuf 在审批时要求添加模型准确率测试。这些尚未在 PR 中完成。
4. 代码质量：gemini-code-assist 建议使用 tl.sigmoid 和 tl.log1p 并清理注释，已采纳。

- 数值回归报告 (DeepSeek-V4 sqrtsoftplus) (correctness): 需要进一步调查并修复回归，PR 已合并但回归未修复。
- PDL gdc_launch_dependents 放置位置 (correctness): 保持当前实现，不移动 PDL 信号。
- 需要覆盖 PDL on/off 的单元测试 (testing): 承认需要，但 PR 未实现，可能后续添加。
- 添加模型准确率测试 (testing): BBuf 虽审批但要求作为后续工作。

风险与影响

- 风险：
  - 数值精度风险：Triton 内核的 sqrtsoftplus 实现可能与 CUDA 版本存在数值差异，已在 DeepSeek-V4 上引发准确率回归。需要仔细对照双精度参考并修复。
  - PDL 内存顺序依赖：当前 Triton kernel 在 PDL 信号后继续写 global memory，若下游不通过 PDL wait 同步可能读到不完整结果。虽然讨论中认为下游负责同步，但仍存在隐患。
  - 迁移风险：替换 AOT kernel 可能带来行为差异（tie-breaking、inf 处理等），测试覆盖了主要配置但未覆盖所有边界条件。
  - 模型级验证缺失：仅依赖单元测试和 benchmark，未在完整模型推理链中验证，可能导致精度回归未被及时捕获。
- 影响：
  - 用户：使用 DeepSeek-V4、Kimi-K2 等 MoE 模型（且 num_expert_group == 1）的用户将自动获得性能提升，但可能经历数值变化导致准确率波动。建议升级后验证模型输出。
  - 系统：统一了 MoE 路由内核路线，减少 AOT 和 JIT 双线维护成本。未来新增评分函数或优化可通过修改 Triton 内核实现。
  - 团队：Triton 的实现更易阅读和修改，降低了 MoE 路由模块的贡献门槛。但需要跟进已报告的数值回归并提供修复。
  - 风险标记：数值精度回归，PDL 内存顺序依赖，缺少 PDL 覆盖测试，模型级测试缺失

关联脉络

- PR #29909 [Bugfix][NPU] Fix Hunyuan3 model where MoE's routing_scaling_ratio is missing on NPU: 同属 MoE 路由模块，修复 NPU 上 missing routed_scaling_factor，与本 PR 的 MoE 路由路径相关。

- PR #30828 Make the mxfp8 MoE runner backend list extensible: 重构了 mxfp8 MoE 后端列表选择，与本 PR 的 MoE 路由内核替换有间接依赖。