PR #28666 完整报告

sgl-project/sglang

[AMD] Fuse shared_expert_gate GEMV into the MoE append kernel (HIP/aiter)

合并时间：2026-08-14 12:32

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/28666

执行摘要

- 一句话：AMD 路径融合共享专家 GEMV 进 append 内核，消除独立启动
- 推荐动作：值得精读。一是 launch-bound vs compute-bound 的判断方法（GEMV 在 decode 下约 0 TFLOPs，调优不如融合）；二是 " 默认路径 byte-for-byte 不变 + 新参数互斥断言 " 的向后兼容策略；三是用双 checkpoint 隔离变量验证融合路径的测试设计。建议后续跟进 nightly MI35x 结果，并推动 AMD CI 基础设施改进以消除 gate 超时。

功能与动机

PR body 指出，AITER shared-expert-fusion 路径上 self.shared_expert_gate(hidden_states) 是一个 [M, hidden] × [hidden, 1] 矩阵 - 向量运算（Cijk 内核 decode 下约 8.9µs），其唯一输出只喂给后续 append 内核；在 decode batch size 下这是纯 kernel-launch 开销（约 0 TFLOPs），因此是融合进 append 内核的典型候选。作者在评论中进一步说明 qwen3.5 mxfp4-moe 的后续优化依赖本 PR，预期带来 1~2% e2e 提升。

实现拆解

1. 内核扩展（fused_moe_triton_kernels.py）：_fused_append_shared_experts_with_weights_kernel 新增 hidden_ptr、wgate_ptr、scale 运行时参数与 FUSE_GATE、HIDDEN、BLOCK_H 编译期常量。FUSE_GATE=True 时，每个 program（一个 token）把 hidden[pid, :] 与 W_gate[:] 在 fp32 中做点积得到 logit，再 tl.sigmoid(logit) * scale 广播到 BLOCK_S 个共享槽位，替换原先从 shared_weights_ptr 的 load。外层 fused_append_shared_experts_with_weights 增加 fuse_gate=False, hidden_states=None, gate_weight=None, scale=1.0 参数；默认 False 时路径与原先一致，fuse_gate=True 与 apply_sigmoid=True 互斥（assert），且强制要求传入 hidden 与 gate 权重；fuse_gate 分支将 gate_weight reshape 为一维、hidden_dim 向上取 2 的幂作为 BLOCK_H，并改用 num_warps=8 以支撑归约。
2. 模型接线（qwen2_moe.py）：新增 _shared_expert_scale()，把 1/ep_size 预缩放逻辑从 _get_shared_expert_weights 中抽出（Allreduce-EP 下后置 all_reduce 会把共享输出累加 ep_size 次，需预除以 ep_size）。_append_shared_to_topk_output 改为按后端分流：_use_aiter 时不再调用 _get_shared_expert_weights，以 fuse_gate=True 直接传入 hidden_states 与 shared_expert_gate.weight；CUDA 走原 _get_shared_expert_weights 路径不变。

3. 测试配套：新增 test/registered/moe/test_fused_append_shared_experts.py，用 _eager_append/_eager_gate 两个 eager 参考实现，验证 legacy（bitwise 相等）、apply_sigmoid、fuse_gate 三条路径；覆盖互斥断言、参数校验、s <= 0 no-op、routed 列保持；注册到 CUDA base-b 与 AMD stage-b。新增 test/registered/amd/accuracy/mi35x/test_qwen35_mxfp4_eval_mi35x.py（nightly）：两个 Qwen3.5-397B MXFP4 checkpoint（共享专家是否量化）分别做 GSM8K 精度门限（> 0.91）与 fuse_gate run-through（> 0.0）+ perf 基准，补齐 e2e 覆盖盲区。
4. 验收数据：作者在 PR body 给出 torch profiler 数据：独立 GEMV launch 从约 8.9μs 降为 0，append 内核从约 4.0μs 增至约 5.2μs，每 MoE 层约 12.9μs→约 5.2μs；GSM8K 精度在运行间方差内。单测全部 PASSED。

关键文件：

- python/sglang/kernels/ops/moe/fused_moe_triton_kernels.py（模块 内核层；类别 source；类型 core-logic；符号 _fused_append_shared_experts_with_weights_kernel, fused_append_shared_experts_with_weights）：核心改动：append kernel 新增 FUSE_GATE 分支与 num_warps=8 的 GEMV 归约；外层 API 新增 fuse_gate/hidden_states/gate_weight 契约并保证默认路径不变。
- python/sglang/srt/models/qwen2_moe.py（模块 模型层；类别 source；类型 core-logic；符号 _shared_expert_scale, _append_shared_to_topk_output）：模型侧接线：抽取 _shared_expert_scale，_append_shared_to_topk_output 按 _use_aiter 分流，aiter 下不再调用 _get_shared_expert_weights 与独立 gate GEMV。
- test/registered/moe/test_fused_append_shared_experts.py（模块 单元测试；类别 test；类型 test-coverage；符号 TestFusedAppendSharedExperts, _eager_gate, test_fuse_gate_matches_eager, test_fuse_gate_and_apply_sigmoid_mutually_exclusive）：新增 kernel 级单测，覆盖 legacy/apply_sigmoid/fuse_gate 三条路径与 eager 参考的一致性、互斥与参数校验断言，注册到 CUDA base-b 与 AMD stage-b。
- test/registered/amd/accuracy/mi35x/test_qwen35_mxfp4_eval_mi35x.py（模块 夜测脚本；类别 test；类型 test-coverage；符号 TestQwen35Mxfp4MI35x, TestQwen35MoeMxfp4MI35x, _run_gsm8k, _run_perf）：新增 MI35x nightly 精度与性能测试，用双 checkpoint 隔离变量验证 fuse_gate e2e 路径与参考精度门限。

关键符号：_fused_append_shared_experts_with_weights_kernel, fused_append_shared_experts_with_weights, _shared_expert_scale, _append_shared_to_topk_output, _get_shared_expert_weights

关键源码片段

python/sglang/kernels/ops/moe/fused_moe_triton_kernels.py

核心改动：append kernel 新增 FUSE_GATE 分支与 num_warps=8 的 GEMV 归约；外层 API 新增 fuse_gate/hidden_states/gate_weight 契约并保证默认路径不变。

```
# python/sglang/kernels/ops/moe/fused_moe_triton_kernels.py
# 关键实现单元：fuse_gate 模式的外层启动函数（默认关闭，兼容 legacy 路径）
def fused_append_shared_experts_with_weights(
    topk_ids, topk_weights, shared_weights,
```

```
    num_fused_shared_experts, N=None,
    apply_sigmoid=False, fuse_gate=False,
    hidden_states=None, gate_weight=None, scale=1.0,
):
    # fuse_gate 路径已内置 sigmoid 与 scale，与 apply_sigmoid 互斥，避免双重激活
    assert not (fuse_gate and apply_sigmoid), (
        'fuse_gate already applies sigmoid in-kernel; do not also set apply_sigmoid'
    )
    assert N is not None, 'N (shared expert base id) must be provided'
    m, k = topk_ids.shape
    s = int(num_fused_shared_experts)
    if s <= 0:
        return topk_ids, topk_weights

    if fuse_gate:
        # GEMV 场景：要求 hidden_states 与 gate_weight，gate_weight 展平为一维
        assert hidden_states is not None and gate_weight is not None, (
            'fuse_gate=True requires hidden_states and gate_weight'
        )
        hidden_arg = hidden_states.contiguous()
        wgate_arg = gate_weight.reshape(-1).contiguous()
        hidden_dim = hidden_arg.shape[1] # 每 token 归约长度
        block_h = triton.next_power_of_2(hidden_dim)
        shared_arg = topk_weights # FUSE_GATE 下不再读 shared_weights
        num_warps = 8 # 点积归约需要更多线程
    else:
        # legacy/apply_sigmoid：保持原语义（dtype 转换、unsqueeze、expand）
        shared_weights_2d = (
            shared_weights if apply_sigmoid else shared_weights.to(topk_weights.dtype)
        )
        if shared_weights_2d.ndim == 1:
            shared_weights_2d = shared_weights_2d.unsqueeze(-1)
        if shared_weights_2d.shape[1] < s:
            shared_weights_2d = shared_weights_2d.expand(m, s)
        shared_arg = shared_weights_2d.contiguous()
        hidden_arg = topk_weights # 占位，避免 kernel 签名分支
        wgate_arg = topk_weights
        hidden_dim = 1
        block_h = 1
        num_warps = 1

    # ... 输出张量分配与 block_k/block_s 计算省略，与原有逻辑一致 ...
    _fused_append_shared_experts_with_weights_kernel[(m,)](
        topk_ids, topk_weights, shared_arg, out_ids, out_weights,
        hidden_arg, wgate_arg,
        N_BASE=N, scale=scale, K=k, S=s,
        BLOCK_K=block_k, BLOCK_S=block_s,
        APPLY_SIGMOID=apply_sigmoid,
        FUSE_GATE=fuse_gate, HIDDEN=hidden_dim, BLOCK_H=block_h,
```

```
        num_warps=num_warps,
    )
    return out_ids, out_weights

# 内核内部：每个 program 处理一个 token
# FUSE_GATE 分支：核内完成 hidden[pid, :] · W_gate[:] 的 fp32 点积
if FUSE_GATE:
    offs_h = tl.arange(0, BLOCK_H)
    mask_h = offs_h < HIDDEN
    h = tl.load(hidden_ptr + pid * HIDDEN + offs_h, mask=mask_h, other=0.0).to(tl.float32)
    w = tl.load(wgate_ptr + offs_h, mask=mask_h, other=0.0).to(tl.float32)
    logit = tl.sum(h * w) # 标量 logit（fp32 归约）
    shared_val = tl.sigmoid(logit) * scale # sigmoid + 1/ep_size 预缩放
    shared_ws = tl.zeros((BLOCK_S,), dtype=tl.float32) + shared_val # 广播到全部共享槽
else:
    # 旧路径：加载预计算权重，必要时应用 apply_sigmoid
    shared_ws = tl.load(shared_weights_ptr + pid * S + offs_s, mask=mask_s)
    if APPLY_SIGMOID:
        shared_ws = tl.sigmoid(shared_ws.to(tl.float32)) * scale
```

python/sglang/srt/models/qwen2_moe.py

模型侧接线：抽取 _shared_expert_scale，_append_shared_to_topk_output 按 _use_aiter 分流，aiter 下不再调用 _get_shared_expert_weights 与独立 gate GEMV。

```
# python/sglang/srt/models/qwen2_moe.py
# aiter 路径：把 shared_expert_gate GEMV、sigmoid、1/ep_size 缩放全部交给 append 内核
def _shared_expert_scale(self) -> float:
    # 1/ep_size 预缩放：Allreduce-EP 下每个 rank 都算同一共享输出，
    # 后置 all_reduce 会累加 ep_size 次，这里预除以 ep_size 抵消
    #  （与 _get_shared_expert_weights 中的逻辑保持一致）
    moe_ep_size = get_parallel().moe_ep_size
    if moe_ep_size > 1 and not is_deepep_class_backend():
        return 1.0 / float(moe_ep_size)
    return 1.0


def _append_shared_to_topk_output(self, topk_output, hidden_states):
    # 把共享专家 id 与权重追加到 topk 输出后再进入 fused MoE
    if not self.enable_shared_expert_fusion or self.shared_expert_gate is None:
        return topk_output

    from sglang.kernels.ops.moe.fused_moe_triton_kernels import (
        fused_append_shared_experts_with_weights,
    )

    if _use_aiter:
        # HIP/aiter：消除独立 gate GEMM launch，GEMV + sigmoid + scale 全部
        # 在 append 内核内完成；此路径不再调用 _get_shared_expert_weights
        fused_topk_ids, fused_topk_weights = fused_append_shared_experts_with_weights(
```

```
            topk_output.topk_ids,
            topk_output.topk_weights,
            None,
            self.num_fused_shared_experts,
            N=self.num_experts,
            fuse_gate=True,
            hidden_states=hidden_states,
            gate_weight=self.shared_expert_gate.weight,
            scale=self._shared_expert_scale(),
        )
    else:
        # CUDA: 保持 legacy 行为，sigmoid + scale 由 eager 路径预先算好
        shared = self._get_shared_expert_weights(hidden_states)
        if shared is None:
            return topk_output
        shared_weights, _ = shared
        fused_topk_ids, fused_topk_weights = fused_append_shared_experts_with_weights(
            topk_output.topk_ids,
            topk_output.topk_weights,
            shared_weights,
            self.num_fused_shared_experts,
            N=self.num_experts,
        )
    return StandardTopKOutput(
        topk_weights=fused_topk_weights,
        topk_ids=fused_topk_ids,
        router_logits=topk_output.router_logits,
    )
```

评论区精华

1. HaiShaw 在 CHANGES_REQUESTED 中指出初版丢弃了 apply_sigmoid 逻辑（"the logic of apply_sigmoid is dropped, please try to retain it."），作者随后恢复，并在 kernel 入口增加 fuse_gate 与 apply_sigmoid 互斥断言与对应单测，最终 APPROVED。
2. amd-bot 多轮 CI 状态反复强调 " 绿色 ≠ 已验证 ": PR 核心代码只在 AMD aiter 上执行，而多个 AMD stage 因 wait-for-stage-a-amd 超时被跳过；最终 kernel 单测在 CUDA 与 AMD 均通过，但 MXFP4 模型级 e2e 只在 nightly 套件中运行。
3. yichiche 澄清 B200 失败为上游已知 flake（test_spec_ngram 连接拒绝，issue #17050），与 PR 无关，并说明该 PR 是 qwen3.5 mxfp4-moe 进一步优化的前置，预期 e2e 提升 1~2%。

- apply_sigmoid 逻辑被初版丢弃，需保留 (design): 作者恢复 apply_sigmoid，并在 kernel 入口增加 fuse_gate 与 apply_sigmoid 互斥断言，同时补单测 test_fuse_gate_and_apply_sigmoid_mutually_exclusive；后续 CI 重新跑通，最终 APPROVED。
- PR CI 无法覆盖 AMD aiter e2e 路径 (testing): 作者新增 nightly run-through 测试（test_qwen35_mxfp4_eval_mi35x.py）并在本地跑通（参考 0.933、fused 0.720）；PR

CI 上内核级已验证，模型级覆盖仍依赖 nightly。

- B200/NVIDIA CI 失败归因 (question): 确认失败为基础设施 / 上游 flake 后，作者申请合并，最终由 HaiShaw 合入。
- 双 checkpoint 隔离验证 fuse_gate 路径 (design): 该测试设计已实现，作者给出本地结果（参考 accuracy 0.933、fused run-through 0.720），用于夜间验证融合路径健康。

风险与影响

- 风险：
  1. 共享代码路径重构：_append_shared_to_topk_output 改为前后端分流，CUDA 分支虽然逻辑不变，但该函数是多个 Qwen MoE 变体的公共出口，回归风险集中在非 aiter 环境；NVIDIA CI 多次 fast-fail-skip，实际覆盖有限。
  2. 数值一致性：Triton tl.sigmoid 与 torch.sigmoid 存在 ULP 级差异，fp32 归约顺序与 cuBLAS 不同，测试用宽松容差（fp32 rtol=1e-4）；作者 GSM8K 结果在运行间方差内，但长尾场景需关注。
  3. 缩放逻辑双处维护：_shared_expert_scale 与 _get_shared_expert_weights 内联了相同的 1/ep_size 判断，未来若 EP 语义变化，两处可能漂移。
  4. 覆盖盲区：fuse_gate 的模型级 e2e 只在 nightly MI35x 测试中运行，PR CI 不覆盖；若 nightly 停摆，回归难被及时捕获。另外 num_warps=8 的新启动配置只在该路径生效，未在 CUDA 下验证编译行为。 - 影响：用户侧：AMD（HIP/aiter）部署 Qwen 系 MoE 模型 decode 性能提升，每 MoE 层约 7.5μs（层耗时下降约 58%），作者估计 e2e 提升 1~2%；CUDA 用户行为不变。系统侧：无新依赖；新增两个测试文件与 CI 注册；kernel API 扩展为向后兼容的默认关闭参数。团队侧：为后续 AMD kernel 融合优化立下模式（gate GEMV 进 append 内核），也暴露了 AMD stage 超时与 nightly-only 覆盖的基础设施问题。 - 风险标记：AMD-only 路径变更，e2e 覆盖依赖 nightly，内核数值非 bitwise 一致，EP 缩放逻辑双处维护

关联脉络

- PR #28658 [AMD] Fuse shared-expert sigmoid into the MoE append kernel (apply_sigmoid): 本 PR 的直接前置：测试 docstring 明确 apply_sigmoid 融合由 PR #28658 引入，fuse_gate 在其基础上把 gate GEMV 一并折叠；两者共享同一 kernel 与测试文件。
- PR #34768 [AMD] CI: pin antlr4-python3-runtime back after lmms-eval (unblocks ROCm 7.2 stage-b evals): AMD CI 基础设施修复，与本 PR 的 AMD stage 注册、gate 超时等问题同属 AMD 验证链路；该类修复直接决定本 PR 的 AMD 覆盖能否稳定运行。
- PR #25855 perf(jit_kernel/deepseek_v4): optimize paged_mqa_metadata: 同为 jit-kernel 性能优化范式（重写内核换 45.3× 收益），与本 PR 的 launch-bound 融合策略形成对比参考。
- PR #34592 [GDN] Honor configured linear-attn verify backend in the kernel dispatcher: 同为 AMD 后端 dispatch 逻辑修复（HIP/aiter 与 CUDA 分支），与本 PR 在 qwen2_moe _is_hip 分支附近的改动相邻。