PR #31323 完整报告

sgl-project/sglang

[AMD] [GLM5] Fuse shared-expert append into aiter grouped-topk (skip per-layer append kernel)

合并时间：2026-08-17 09:22

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/31323

执行摘要

- 一句话：GLM5 共享专家 append 融合进 aiter topk，消除每层内核
- 推荐动作：值得精读，重点看三点设计：① 用持久预填充 buffer + row stride 部分列写入消除每层 kernel 启动，并与 CUDA graph 地址稳定性耦合；② 容量超限降级回 fallback 的“只影响覆盖、不影响正确性”设计哲学；③ review 中“条件镜像易碎、改用 shape 检测”的健壮性权衡，以及“server args 未初始化时优雅降级”的工程细节。同时应把“最终被 PR#35105 回滚”作为教训：特殊硬件路径优化应配套直接针对该分支的单元测试与更小粒度的 CI 验证，避免只依赖 nightly 精度套件。

功能与动机

PR body 指出：在非 EP aiter grouped-topk MoE 路径下，每个 decoder layer 都要运行独立的 _fused_append_shared_experts 内核把 shared expert 追加进 top-k ids/weights。这个 PR 的目标是“pre-populate a persistent top-k buffer's shared-expert columns once and let the aiter kernel write only the routed columns via row stride, so the per-layer append kernel is removed”，并且强调该做法与 plain append“Bit-identical”——shared 列使用与 aiter append 相同常数 fused_shared_experts_scaling_factor，因此对任何 shared-expert scaling 都正确。性能动机方面，PR 给出了 MI355X TP4 上 i1024/o1024 与 i8192/o1024 两组吞吐 /TPOT 对比，结论是“removes one kernel launch per layer (3 kernels → fewer) and is bit-identical, so it is a clean simplification with a slight net-positive tendency at low concurrency”。

实现拆解

1. 新增持久 buffer 与容量计算 helper（topk.py 顶部）：新增 _AITER_TOPK_FUSE_SHARED_MAX_TOKENS_CAP = 131072、_aiter_topk_fuse_shared_max_tokens_cache 和全局 dict _aiter_topk_fuse_shared_bufs。_get_aiter_topk_fuse_shared_max_tokens() 从 runtime_context.get_server_args() 读取 chunked_prefill_size / max_prefill_tokens（下限 8192、上限 131072）决定持久 buffer 的行数，结果缓存；未初始化时降级返回安全上限且不缓存，避免把“仅影响覆盖范围”的 sizing helper 变成硬崩溃。_get_aiter_topk_fuse_shared_buf() 按 (topk_routed, n_shared, num_experts, shared_weight, device) 缓存 [M, topk_routed + n_shared] 的 weight/id 张量，shared 列一次性预填充，固定 M 保证 CUDA graph 重放时地址稳定。

2. 改造 biased_grouped_topk_gpu 的 aiter 分支：新增参数 fused_shared_experts_scaling_factor。满足 num_fused_shared_experts > 0、moe_ep_size == 1、token <= max_tokens 时启用 _shared_fuse：从持久 buffer 取切片 full_w[:token, :topk] 交给 aiter_biased_grouped_topk 只写 routed 列，随后返回全宽视图 full_w[:token], full_ids[:token]；不满足条件时仍走原 torch.empty 临时 buffer 路径，由后续 append 补齐。该分支同时作用于 prefill 与 decode。
3. _post_process_topk_ids 用 shape 检测跳过 append：原 _aiter_append 分支无条件调用 fused_append_shared_experts；现改为先判断 topk_ids.shape[1] == topk_config.top_k——若持久 buffer 已返回全宽（routed + shared）则跳过 append，否则照旧追加。采用 shape 检查而非镜像上游 _shared_fuse 条件（含 token <= MAX 边界），避免条件漂移导致静默缺少 shared expert。
4. 配套与测试情况：PR 早期新增的 get_global_server_args() 调用点触发 NVIDIA lint guardrail test_legacy_global_ratchet.py 失败，经 commit 99cbde2 改用 runtime_context.get_server_args() 后 CUDA/AMD CI 全绿；CPU 变体 biased_grouped_topk_cpu 曾为 API 对齐添加未使用参数，最终 commit 189b90ff 删除。本 PR 未附带直接单元测试，融合路径主要依赖 AMD nightly 精度套件（GSM8K）验证。

关键文件：

- python/sglang/srt/layers/moe/topk.py（模块 专家路由；类别 source；类型 core-logic；符号 _get_aiter_topk_fuse_shared_max_tokens, _get_aiter_topk_fuse_shared_buf, biased_grouped_topk_gpu, _post_process_topk_ids）：唯一变更文件，MoE 路由核心模块。新增持久 topk buffer 与容量计算 helper，改造 biased_grouped_topk_gpu 的 aiter 融合分支，并在 _post_process_topk_ids 中通过 shape 检测跳过 per-layer append，整体消除了 AMD aiter 路径每层一次的内核启动。

关键符号：_get_aiter_topk_fuse_shared_max_tokens, _get_aiter_topk_fuse_shared_buf, biased_grouped_topk_gpu, _post_process_topk_ids

关键源码片段

python/sglang/srt/layers/moe/topk.py

唯一变更文件，MoE 路由核心模块。新增持久 topk buffer 与容量计算 helper，改造 biased_grouped_topk_gpu 的 aiter 融合分支，并在 _post_process_topk_ids 中通过 shape 检测跳过 per-layer append，整体消除了 AMD aiter 路径每层一次的内核启动。

```
# python/sglang/srt/layers/moe/topk.py

# 持久 topk buffer 的上限与缓存：仅用于 aiter 非 EP 融合路径的容量，不影响正确性
_AITER_TOPK_FUSE_SHARED_MAX_TOKENS_CAP = 131072
_aiter_topk_fuse_shared_bufs: dict = {}


def _get_aiter_topk_fuse_shared_buf(
    topk_routed: int, n_shared: int, num_experts: int, shared_weight: float, device
):
    """按 (topk_routed, n_shared, num_experts, shared_weight, device) 缓存持久 buffer。
```

```
    buffer 形状固定为 [M, topk_routed + n_shared]，shared 列只预填充一次：
    id = num_experts + i，weight = shared_weight。固定 M 保证 CUDA graph 重放时
    张量地址稳定，这是该优化能与 CUDA graph 配合的关键前提。
    """
    key = (topk_routed, n_shared, num_experts, float(shared_weight), str(device))
    buf = _aiter_topk_fuse_shared_bufs.get(key)
    if buf is None:
        total = topk_routed + n_shared
        M = _get_aiter_topk_fuse_shared_max_tokens()
        w = torch.empty((M, total), dtype=torch.float32, device=device)
        ids = torch.empty((M, total), dtype=torch.int32, device=device)
        # 预填充 shared 列：id 从 num_experts 开始连续编号，weight 为统一常数
        ids[:, topk_routed:] = torch.arange(
            num_experts, num_experts + n_shared, dtype=torch.int32, device=device
        ).unsqueeze(0)
        w[:, topk_routed:] = shared_weight
        buf = (w, ids)
        _aiter_topk_fuse_shared_bufs[key] = buf
    return buf

# biased_grouped_topk_gpu 的 aiter 分支（节选）
elif _use_aiter:
    assert not apply_routed_scaling_factor_on_output, "Not implemented"
    token = gating_output.shape[0]
    device = gating_output.device
    # 仅非 EP + 有 fused shared experts + token 数在持久 buffer 容量内时启用融合
    _shared_fuse = (
        num_fused_shared_experts > 0
        and get_parallel().moe_ep_size == 1
        and token <= _get_aiter_topk_fuse_shared_max_tokens()
    )
    if _shared_fuse:
        # shared 列权重必须与 _post_process_topk_ids 里 append 内核写入的常数一致，
        # 才能保证与旧路径 bit-identical
        _shared_w = (
            1.0
            if fused_shared_experts_scaling_factor is None
            else fused_shared_experts_scaling_factor
        )
        full_w, full_ids = _get_aiter_topk_fuse_shared_buf(
            topk, num_fused_shared_experts, num_experts, _shared_w, device
        )
        # aiter 内核只写 routed 列（[:, :topk]），shared 列保持预填充值不受扰动
        topk_weights = full_w[:token, :topk]
        topk_ids = full_ids[:token, :topk]
    else:
        # 大 prefill 或非适用场景回退普通临时 buffer，后续由常规 append 路径补 shared
        topk_weights = torch.empty((token, topk), dtype=torch.float32, device=device)
        topk_ids = torch.empty((token, topk), dtype=torch.int32, device=device)
```

```
    aiter_biased_grouped_topk(
        gating_output,
        correction_bias.to(dtype=gating_output.dtype),
        topk_weights,
        topk_ids,
        num_expert_group,
        topk_group,
        renormalize,
        routed_scaling_factor if routed_scaling_factor is not None else 1.0,
    )
    if _shared_fuse:
        # 返回全宽视图（routed 已写入 + shared 已预填充），_post_process 据此跳过 append
        return full_w[:token], full_ids[:token]
    return topk_weights, topk_ids

# _post_process_topk_ids 的 aiter append 分支（节选）
elif _aiter_append:
    # 用 shape 判断 shared expert 是否已在 biased_grouped_topk_gpu 中融合：
    # 融合时 topk_ids 已是全宽（routed + shared），shape[1] == top_k；
    # 回退路径只有 routed 列，必须在此处 append。相比镜像上游 _shared_fuse 条件，
    # shape 检查对调用链变化更健壮，不会因条件漂移产生静默缺共享专家的问题。
    _shared_fused_in_topk = (
        num_fused_shared_experts > 0
        and get_parallel().moe_ep_size == 1
        and topk_ids.shape[1] == topk_config.top_k
    )
    if not _shared_fused_in_topk:
        M, N = router_logits.shape
        scale_factor = (
            1.0
            if fused_shared_experts_scaling_factor is None
            else fused_shared_experts_scaling_factor
        )
        # 延迟导入避免循环依赖
        from sglang.kernels.ops.moe.fused_moe_triton_kernels import (
            fused_append_shared_experts,
        )

        topk_ids, topk_weights = fused_append_shared_experts(
            topk_ids,
            topk_weights,
            num_fused_shared_experts,
            scale_factor,
            N, # base id for shared experts
        )
```

评论区精华

- gemini-code-assist[bot]（high priority）：指出 get_global_server_args() 在全局 server args 未发布前调用会抛 ValueError（如离线单测、初始化脚本），建议 try/except 并返回安全 fallback 不缓存。Jacob0226 回复“Solved in commit abb96d9”，随后 commit 99cbde2 进一步改用 runtime_context.get_server_args() 以遵守 legacy accessor ratchet guardrail。
- gemini-code-assist[bot]（high priority）：批评“镜像 biased_grouped_topk_gpu 的复杂条件”过于脆弱，“can lead to silent correctness bugs or missing shared experts if other routing paths are used”，建议改用 topk_ids.shape[1] == topk_config.top_k 检测。Jacob0226 在 commit 1f4fe926 应用该建议，并附上“shape check is robust to the exact fast-path conditions”的说明。
- amd-bot CI 报告：判定“Do not merge”，指出 guardrail 失败导致 fast-fail 跳过全部 NVIDIA GPU 阶段（base-b/c、extra-*），且融合路径未被任何 PR CI 测试执行（仅 nightly AMD 精度套件覆盖）。修复 guardrail 并 rerun 后 Jacob0226 贴图确认 CUDA 与 AMD CI 全绿。
- clintg6（APPROVED）：独立复测“no impact on accuracy (95.3%) and 1% uplift on TPOT/TPUT for ISL/OSL 1k/1k”，并验证 gemini 两条建议均无问题。
- HaiShaw：“CI signal green. Nit: remove unused param.”——要求删除 CPU 变体上为 API parity 添加但未使用的 fused_shared_experts_scaling_factor 参数，最终以 commit 189b90ff 删除。
  - get_global_server_args() 未初始化时崩溃风险 (correctness): Jacob0226 在 commit abb96d9 加 try/except 降级，随后 commit 99cbde2 改用 runtime_context.get_server_args() 以通过 legacy accessor ratchet guardrail。
  - 镜像条件脆弱，改用 shape 检测融合状态 (design): Jacob0226 在 commit 1f4fe926 应用 shape 检查替代条件镜像，并说明其对 fast-path 条件变化更健壮。
  - PR CI 不完整且变更未被 PR CI 覆盖 (testing): 修复 guardrail 后 rerun-failed-ci，CUDA 与 AMD CI 全绿；但直接针对融合路径的单元测试始终缺失。
  - CPU 变体新增未使用参数 (style): 最终 commit 189b90ff 删除该参数。

风险与影响

- 风险：
  - 已被回滚（最大现实风险）：本 PR 合并后，PR#35105 以“回滚 #31323，恢复逐层 append，解除 CI 阻塞”为由 revert 了本改动，说明该优化在 main 上引发了 CI 阻塞（Issue 评论区 hzh0425 也指出某次 run 疑似被本 PR 弄挂）。恢复逐层 append 后风险解除。
  - 全局可变状态缓存：_aiter_topk_fuse_shared_bufs 是模块级 dict，进程生命周期内不释放；每个 key 对应约 [131072, topk + n_shared] * 8B 的显存，多配置组合（不同 topk/n_shared/device/shared_weight）会累积占用。
  - shape 检查的隐式契约：_post_process_topk_ids 用 topk_ids.shape[1] == topk_config.top_k 判断是否已融合，隐含“任何上游路径都不会先改变 topk_ids 宽度”的假设；未来若引入 remap/reshape 路径可能误判。

- CUDA graph 地址稳定依赖：持久 buffer 固定 M 是地址稳定的前提，但 fast path 覆盖的 graph 捕获尺寸若超过 buffer 行数会静默走 fallback，依赖 _post_process 的 shape 检查保持两路径一致，缺少直接测试锁定。
- 测试覆盖缺口：无直接单元测试；biased_grouped_topk_gpu 的 aiter 分支与 _post_process_topk_ids 的跳过逻辑没有针对融合 / 回退 /EP 模式的分支级验证。
- 影响：影响范围限定在 AMD gfx950 / MI355X + aiter + 非 EP（moe_ep_size == 1）+ 带 fused shared experts（GLM-5.2）的 MoE 路由路径：消除每层一次 append 内核启动，TPUT +0.4%–0.9%、TPOT −0.1%–1.4%（PR 自测数据，clintg6 独立复测约 1%）。其他架构（CUDA/MUSA/CPU）、EP 模式及非适用路径一律 fallback 到原有 per-layer append，数值与行为不受影响（PR 声称 bit-identical）。对团队而言，该 PR 暴露了 AMD 专属路径优化与全局 CI 质量门（legacy accessor ratchet、NVIDIA GPU 阶段）之间的摩擦：一轮 guardrail 修复、一次 rerun、最终被回滚，说明这类改动需要更强的 CI 覆盖与更小的 main 集成面。
- 风险标记：已被回滚（PR#35105），缺少直接测试覆盖，全局缓存增加显存，CI 门禁摩擦，shape 检查依赖隐式契约

关联脉络

- PR #35105 Revert "[AMD] [GLM5] Fuse shared-expert append into aiter grouped-topk (skip per-layer append kernel)": 直接回滚本 PR，恢复逐层 append 以解除 main CI 阻塞，是本 PR 合并后的关键后续。
- PR #31324 [AMD] [GLM5] Skip DSA decode indexer when kv_len <= index_topk (dense k-only fast path): 同为 GLM5 在 AMD 上的性能优化线，展示了该模型系列持续通过消除冗余内核 / 路径来降延迟的演进方向。