PR #23707 完整报告

sgl-project/sglang

[MoE] Deprecate act_and_mul_triton; fold filter_expert into JIT silu/gelu_and_mul

合并时间：2026-04-26 16:41

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/23707

执行摘要

- 一句话：废弃 Triton act_and_mul，将 filter_expert 合并至 JIT CUDA 激活核
- 推荐动作：该 PR 设计清晰、测试充分，值得精读。重点关注 if constexpr 在 CUDA kernel 中实现零开销抽象的模式，以及过滤路径在不活跃 token 上的 work-stealing 效果。对于 AMD CI 的覆盖，建议后续 PR 跟进。

功能与动机

act_and_mul_triton 与已有的 JIT CUDA silu_and_mul / gelu_and_mul 功能重复，唯一区别是支持 filter_expert 跳过 -1 expert。JIT 核更快，合并可消除冗余内核并减少约 100 行 Triton 代码。

实现拆解

1. CUDA kernel 模板化扩展— 在 activation.cuh 中向 ActivationParams 添加 expert_ids 和 expert_step 字段；为 act_and_mul_kernel 增加编译时 kFilterExpert 模板参数，通过 if constexpr(kFilterExpert) 决定是否加载 expert 判定；暴露 run_activation_filtered 主机方法。
2. Python 前端扩展— 在 activation.py 中新增 _run_activation_filtered_inplace（注册为 custom_op），并修改 run_activation / silu_and_mul / gelu_and_mul / gelu_tanh_and_mul 以接受可选的 expert_ids 和 expert_step 参数；当 expert_ids 非 None 时路由到 filtered 路径。
3. MoE runner 调用替换— 在 fused_moe.py 的 _fused_moe_kernel_sequence 中，将 act_and_mul_triton(...) 调用替换为 silu_and_mul / gelu_and_mul 并传入 expert_ids 与 expert_step。仅 CUDA 路径走 filtered JIT kernel；HIP/XPU 保持 unfiltered 路径（由下游 down kernel 对 filtered 行写入零来弥补）。
4. 删除冗余代码— 移除 fused_moe_triton_kernels.py 中的 tanh、_apply_activation、act_and_mul_kernel（Triton jit）以及 act_and_mul_triton 封装，共约 106 行。
5. 测试与基准— 在 test_activation.py 添加 3 个 filter_expert 测试（常规随机跳过、全部跳过、全部不跳过验证比特精确相等）；在 bench_activation.py 添加 filter 对比基准，覆盖不同 skip_ratio。

关键文件：

- python/sglang/srt/layers/moe/moe_runner/triton_utils/fused_moe_triton_kernels.py（模块 MoE 内核；类别 source；类型 core-logic；符号 tanh, _apply_activation, act_and_mul_kernel, act_and_mul_triton）：核心删除文件：移除约 106 行 Triton 重复代码（tanh, _apply_activation, act_and_mul_kernel, act_and_mul_triton），是本次重构的主要目标。
- python/sglang/jit_kernel/activation.py（模块 JIT 激活层；类别 source；类型 core-logic；符号 _run_activation_filtered_inplace）：新增 filtered 激活函数注册与路由，是 JIT 侧的扩展核心。
- python/sglang/srt/layers/moe/moe_runner/triton_utils/fused_moe.py（模块 MoE 执行器；类别 source；类型 core-logic）：MoE runner 调用侧，将 act_and_mul_triton 替换为带 expert_ids 的 JIT 激活函数，是功能替换的关键衔接。
- python/sglang/jit_kernel/csrc/elementwise/activation.cuh（模块 CUDA 核；类别 other；类型 core-logic）：CUDA kernel 端模板化改造，新增 kFilterExpert 编译时分支和 run_activation_filtered 入口。
- python/sglang/jit_kernel/tests/test_activation.py（模块 激活测试；类别 test；类型 test-coverage；符号 test_activation_filter_expert, test_activation_filter_expert_all_skipped, test_activation_filter_expert_none_skipped）：新增 filter_expert 测试套件，覆盖常规跳过、全部跳过、全部不跳过三种场景，none_skipped 验证比特精确相等。
- python/sglang/jit_kernel/benchmark/bench_activation.py（模块 激活基准；类别 source；类型 benchmark；符号 _make_expert_ids, benchmark_filter, f）：新增 filter_expert 基准测试，比较过滤与未过滤路径在不同 skip ratio 下的延迟。

关键符号：_run_activation_filtered_inplace, run_activation, silu_and_mul, gelu_and_mul, gelu_tanh_and_mul, act_and_mul_kernel (CUDA template), select_kernel, launch, test_activation_filter_expert, test_activation_filter_expert_all_skipped, test_activation_filter_expert_none_skipped, benchmark_filter

关键源码片段

python/sglang/jit_kernel/activation.py

新增 filtered 激活函数注册与路由，是 JIT 侧的扩展核心。

```
@register_custom_op(mutates_args=["out"])
def _run_activation_filtered_inplace(
    op_name: str,
    input: torch.Tensor,
    out: torch.Tensor,
    expert_ids: torch.Tensor,
    expert_step: int,
) -> None:
    # 将输入 reshape 为 2D：每行包含 gate 和 up 输出（各占一半）
    hidden_size = input.shape[-1] // 2
    module = _jit_activation_module(input.dtype)
    input_2d = input.view(-1, hidden_size * 2)
```

```
    out_2d = out.view(-1, hidden_size)
    # 调用 CUDA JIT kernel 的 run_activation_filtered 入口
    module.run_activation_filtered(input_2d, out_2d, expert_ids, expert_step, op_name)


def run_activation(
    op_name: str,
    input: torch.Tensor,
    out: Optional[torch.Tensor],
    expert_ids: Optional[torch.Tensor] = None,
    expert_step: int = 1,
) -> torch.Tensor:
    """Apply ``op_name`` activation followed by element-wise multiplication.

    When ``expert_ids`` is provided, output rows are skipped for tokens whose
    routed expert id is ``-1``. ``expert_step`` is 1 for per-token routing and
    ``BLOCK_SIZE_M`` for sorted/TMA routing — i.e. ``expert_ids[token_id //
    expert_step]`` is consulted before computing each row.
    """
    assert op_name in SUPPORTED_ACTIVATIONS, f"Unsupported activation: {op_name}"
    hidden_size = input.shape[-1] // 2
    if out is None:
        out = input.new_empty(*input.shape[:-1], hidden_size)
    if expert_ids is None:
        _run_activation_inplace(op_name, input, out) # 旧路径，无过滤
    else:
        _run_activation_filtered_inplace(op_name, input, out, expert_ids, expert_step)
    return out
```

python/sglang/srt/layers/moe/moe_runner/triton_utils/fused_moe.py

MoE runner 调用侧，将 act_and_mul_triton 替换为带 expert_ids 的 JIT 激活函数，是功能替换的关键衔接。

```
    # 原代码：根据 filter_expert 调用 act_and_mul_triton 或 silu_and_mul
    # 新代码：仅 CUDA 时调用带 expert_ids 的 JIT kernel，HIP/XPU 走无过滤路径
    if filter_expert and _is_cuda:
        silu_and_mul(
            intermediate_cache1.view(-1, N),
            intermediate_cache2,
            expert_ids=(expert_ids if down_moe_use_tma else topk_ids.view(-1)),
            expert_step=(config["BLOCK_SIZE_M"] if down_moe_use_tma else 1),
        )
    else:
        # HIP/XPU 或非 filter_expert: 下游 down kernel 会自动对过滤行写零，安全避免计算
        silu_and_mul(intermediate_cache1.view(-1, N), intermediate_cache2)
```

评论区精华

Gemini Code Assist 建议添加 expert_ids 校验：在 activation.cuh 的 launch 函数中，应对 expert_ids 张量进行设备一致性（与 input/out 同设备）和维度（1D）的运行时检查，以避免潜在的段错误或非法内存访问。该评论状态为 COMMENTED，未见对应的修改提交，建议未在本次 PR 中采纳。

- 添加 expert_ids 张量的设备与维度检查 (correctness): 评论处于 COMMENTED 状态，未见对应代码修改，建议未在本 PR 中采纳。

风险与影响

- 风险：
  1. HIP/XPU 兼容性：filter_expert 路径在 HIP/XPU 上走 unfiltered AOT kernel，但下游 down kernel 仍会写入零，结果正确但可能浪费少量算力；若 AMD CI 未长期覆盖此路径，可能引入隐蔽错误。
  2. 删除代码依赖风险：_apply_activation 和 tanh 经作者 grep 确认无其他调用者，但若未来合入其他分支可能有隐患。
  3. filter_expert 逻辑正确性：新 JIT kernel 的 none_skipped 测试验证了与 unfiltered 路径比特精确相等，但复杂 TMA 路由路径下的 expert_step 计算仍依赖下游正确传递参数。 - 影响：用户侧：MoE 模型推理（特别是 DeepSeek、Qwen3-MoE 等使用 EP+filter_expert 的模型）在 decode 阶段可获得 2~3 倍激活计算加速，高 expert 跳过率时效果更明显；无需用户干预。系统侧：代码精简约 106 行，消除 Triton 冗余内核，降低维护成本；JIT kernel 模板化设计便于未来扩展其他激活函数。团队侧：后续可基于此模式将更多 Triton 内核收敛到 JIT CUDA。 - 风险标记：缺少 AMD CI 覆盖，HIP/XPU 回退路径未测试，删除 Triton 代码依赖

关联脉络

- PR #23731 Fix Qwen3 MoE double-reduce when DP attention + EP + reduce_scatterv: 同为 MoE 路径的修复和优化，与当前 PR 关注的 filter_expert 路径在同一 MoE 控制流中。
- PR #23732 Apply should_use_dp_reduce_scatterv guard to remaining MoE models: 扩展 MoE 保护到更多模型，与当前 PR 的 MoE kernel 修改在相同模块中。
- PR #23734 Fix Qwen3 MoE: also guard EP all-reduce with not use_reduce_scatter: 继续修复 Qwen3 MoE 的问题，与当前 PR 的 MoE 激活 kernel 替换有间接关联。