PR #30387 完整报告

sgl-project/sglang

Fix zero expert routed ids for MoE backends

合并时间：2026-07-08 21:23

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/30387

执行摘要

- 一句话：修复零专家路由传入非法负数 id 导致 CUDA 崩溃
- 推荐动作：建议合入，但后续需跟进 review 中提出的命名问题和非 identity 路径保护。

功能与动机

LongCat-Flash 中 zero experts 表示为 top-k id >= num_experts，预处理将 id 设为 -1 并清零 scale。但某些 MoE 内核仍将 id 用作 gather/sort 输入，不处理负数 id，在高并发场景下出现 CUDA illegal memory access。改为合法 id 0 配合零 scale 在数学上等价，且避免传递负数 id。

实现拆解

1. 核心修复：在 python/sglang/srt/layers/moe/ep_moe/kernels.py 的 zero_experts_compute_triton 函数中，将零专家的 expert_indices 从 -1 改为 0，同时仍将对应 expert_scales 置零，确保路由专家的贡献被消除。
2. 新增单元测试：test/registered/moe/test_zero_experts.py，测试零专家路由后 expert_indices 均非负，且输出与理论值一致（零专家部分输出 hidden_states * 剩余 scale）。

关键文件：

- python/sglang/srt/layers/moe/ep_moe/kernels.py（模块 MoE 层；类别 source；类型 core-logic；符号 zero_experts_compute_triton）：核心修复文件，修改了零专家 id 从 -1 变为 0，避免 MoE 内核崩溃。
- test/registered/moe/test_zero_experts.py（模块 测试；类别 test；类型 test-coverage；符号 TestZeroExpertsComputeTriton, test_zero_expert_routes_keep_valid_ids）：新增的 CUDA 单元测试，验证零专家路由后 expert_indices 非负且输出正确。

关键符号：zero_experts_compute_triton

关键源码片段

python/sglang/srt/layers/moe/ep_moe/kernels.py

核心修复文件，修改了零专家 id 从 -1 变为 0，避免 MoE 内核崩溃。

```
def zero_experts_compute_triton(
```

```
    expert_indices, expert_scales, num_experts, zero_expert_type, hidden_states
):
    N = expert_indices.numel()
    top_k = expert_indices.size(-1)
    grid = lambda meta: (triton.cdiv(N, meta["BLOCK_SIZE"]),)

    if zero_expert_type == "identity":
        zero_expert_mask = expert_indices < num_experts
        zero_expert_scales = expert_scales.clone()
        zero_expert_scales[zero_expert_mask] = 0.0

    normal_expert_mask = expert_indices >= num_experts
    # 使用合法专家 id 0 代替 -1，避免 MoE 内核无法处理负数 id
    # 配合零 scale 仍能消除该路由专家的贡献，数学上等价
    expert_indices[normal_expert_mask] = 0
    expert_scales[normal_expert_mask] = 0.0

    output = torch.zeros_like(hidden_states).to(hidden_states.device)
    hidden_dim = hidden_states.size(-1)
    num_tokens = hidden_states.size(0)

    grid = lambda meta: (num_tokens * (hidden_dim // meta["BLOCK_SIZE"]),)
    compute_identity_kernel[grid](
        top_k,
        hidden_states,
        zero_expert_scales, # 注意：若 zero_expert_type 非 "identity"，此处 NameError
        num_tokens,
        output,
        hidden_dim,
        zero_expert_scales.stride(0),
        BLOCK_SIZE=256,
    )

    return output
```

test/registered/moe/test_zero_experts.py

新增的 CUDA 单元测试，验证零专家路由后 expert_indices 非负且输出正确。

```
import unittest
import torch
from sglang.srt.layers.moe.ep_moe.kernels import zero_experts_compute_triton
from sglang.test.ci.ci_register import register_cuda_ci
from sglang.test.test_utils import CustomTestCase

register_cuda_ci(est_time=5, stage="base-b", runner_config="1-gpu-small")

@unittest.skipIf(not torch.cuda.is_available(), "CUDA is required")
class TestZeroExpertsComputeTriton(CustomTestCase):
    def test_zero_expert_routes_keep_valid_ids(self):
```

```
        num_experts = 4
        # 构造 2 个 token，每个 top-3 路由，其中 id >= num_experts 视为零专家
        hidden_states = torch.arange(2 * 512, dtype=torch.float32, device="cuda").reshape(2,
        512)
        expert_indices = torch.tensor(
            [[0, 4, 1], [5, 2, 6]], dtype=torch.int64, device="cuda"
        )
        expert_scales = torch.tensor(
            [[0.1, 0.3, 0.2], [0.4, 0.5, 0.6]], dtype=torch.float32, device="cuda"
        )
        original_indices = expert_indices.clone()
        original_scales = expert_scales.clone()

        output = zero_experts_compute_triton(
            expert_indices=expert_indices,
            expert_scales=expert_scales,
            num_experts=num_experts,
            zero_expert_type="identity",
            hidden_states=hidden_states,
        )
        torch.cuda.synchronize()

        # 验证所有 expert_indices 非负
        self.assertTrue(torch.all(expert_indices >= 0).item())
        # 正常路由的 id 和 scale 保持不变
        normal_mask = original_indices < num_experts
        torch.testing.assert_close(expert_indices[normal_mask], original_indices[normal_mask])
        torch.testing.assert_close(expert_scales[normal_mask], original_scales[normal_mask])
        # 零专家的 id 被改为 0，scale 被改为 0
        zero_mask = original_indices >= num_experts
        torch.testing.assert_close(expert_indices[zero_mask], torch.zeros_like(expert_indices[zero_
        mask]))
        torch.testing.assert_close(expert_scales[zero_mask], torch.zeros_like(expert_scales[zero_
        mask]))
        # 输出应为 hidden_states * 剩余 scale 的和（仅保留正常路由的 scale）
        expected_scale = (original_scales * zero_mask.to(original_scales.dtype)).sum(dim=-1,
        keepdim=True)
        torch.testing.assert_close(output, hidden_states * expected_scale)

if __name__ == "__main__":
    unittest.main()
```

评论区精华

Review 中 gemini-code-assist[bot] 提出了两个中优先级问题：一是变量名 normal_expert_mask 和 zero_expert_mask 含义颠倒导致可读性差；二是当 zero_expert_type 不为 "identity" 时 zero_expert_scales 未定义可能导致 NameError。当前 PR 尚未处理这两点。

- 变量命名歧义：normal_expert_mask 和 zero_expert_mask 含义颠倒 (design): 未在 PR 中处理，建议后续重构命名。
- zero_expert_scales 在非 identity 模式下未定义导致 NameError (correctness): 未在 PR 中处理，当前仅支持 "identity" 模式。

风险与影响

- 风险：低风险：修复仅更改了 id 赋值（-1 -> 0）且已通过 H200 单测和 GSM8K 端到端验证；但变量命名反转可能使未来维护者困惑，且未处理的 NameError 分支在非 identity 模式下存在潜在运行时崩溃。
- 影响：直接影响使用 MoE 路由且存在零专家选择的推理场景（如 DeepSeek 系列）。修复合入后，LongCat-Flash-Lite FP8 等场景下 CUDA 非法访问崩溃消失，端到端负载完成。
- 风险标记：潜在未定义变量分支，变量命名混淆

关联脉络

- PR #9824：本 PR 的 bug 由 #9824 引入，PR body 中提及。