PR #28450 完整报告

sgl-project/sglang

[AMD] Fuse shared-expert append + DeepEP remap into one Triton kernel

合并时间：2026-06-25 16:22

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/28450

执行摘要

- 一句话：融合共享专家追加与 DeepEP 重映射为单个 Triton 内核
- 推荐动作：本 PR 值得相关人员精读，特别是关注 AMD MoE 推理性能优化的工程师。其设计体现了“每层一次 Triton launch”的极致融合思路，且对数值一致性有严格验证（bit-identical 断言）。代码风格和 review 中提出的修改建议（如导入简化、异常代替 assert）可作为团队代码规范参考。建议在 AMD 生产环境充分测试后，考虑将 SGLANG_MORI_NO_PAD_MASK 默认开启以最大化性能。

功能与动机

On the AMD aiter / DeepEP-class MoE path, every layer runs the shared-expert append (fused_append_shared_experts) immediately followed by the eager DeepEP interleaved remap (_remap_topk_for_deepep). The remap is a sequence of small, launch-latency-bound elementwise ops (floor-div, add, arange, fill, copy) that each cost a kernel launch. With one shared-expert append + remap per MoE layer across many layers, this launch overhead is pure dispatch cost on top of work the append kernel is already doing. This PR fuses the append and the remap into the single Triton kernel that already iterates over the topk rows, so the remap math runs on the rows already resident in registers instead of as extra eager launches.

实现拆解

1. 新增融合内核：在 fused_moe_triton_kernels.py 中添加 _fused_append_remap_shared_experts_deepep_kernel 及其包装函数 fused_append_remap_shared_experts_deepep。该内核在单次 Triton launch 中完成共享专家追加和 DeepEP ID 重映射，将原分步操作中的 6 次 eager 内核启动（floor-div、add、arange、fill、copy）合并为一次，计算在寄存器内完成。
2. 连接 TopK 后处理：在 topk.py 的 _post_process_topk_ids 中，当满足 aiter+DeepEP 条件时，调用新融合内核替代原有的 fused_append_shared_experts() + _remap_topk_for_deepep() 序列。同时添加 _fill_padded_rows_kernel 和 _fill_padded_rows 函数，用一个 Triton launch 实现填充行的清零 / 填充，替代 arange + (>=) + index_put_ 的 eager 序列。
3. 增强输入校验：_fill_padded_rows 使用显式 raise（而非 assert）进行输入校验，确保在 python -O 环境下仍能正确报错，避免无提示的 Triton 错误。

4. 可选 HIP 填充掩码跳过：引入环境变量 SGLANG_MORI_NO_PAD_MASK（默认关闭），可跳过 HIP 上的填充路由权重掩码。此开关影响数值精度，需经验证后开启。测试中使用 get_parallel().override() 替代已移除的函数 patch，确保 EP 拓扑正确。
5. 新增单元测试：在 test_fused_append_remap_deepep.py 中注册 4 个测试用例，覆盖与 golden reference 的 match、与原有 eager 路径的等价性、共享权重在 aiter 路径上为 1.0、以及无共享专家时的空操作。测试同时在 CUDA 和 AMD CI 中注册。

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 _fused_append_remap_shared_experts_deepep_kernel, fused_append_remap_shared_experts_deepep）：核心变更文件，添加了融合追加 + 重映射的 Triton 内核及其包装函数，是性能优化的核心实现。
- python/sglang/srt/layers/moe/topk.py（模块 TopK 路由；类别 source；类型 dependency-wiring；符号 _fill_padded_rows_kernel, _can_fuse_padded_region, _fill_padded_rows）：修改了 topk 后处理逻辑，接入融合内核并新增填充行 Triton kernel。同时引入了环境变量和输入校验改进。
- test/registered/moe/test_fused_append_remap_deepep.py（模块 测试；类别 test；类型 test-coverage；符号 _reference_append_remap, TestFusedAppendRemapDeepEP, _make_inputs, _shared_id_base）：新增的单元测试文件，全面覆盖融合内核的正确性、与 eager 等价性、共享权重逻辑和空操作场景，注册了 AMD 和 CUDA CI。

关键符号：fused_append_remap_shared_experts_deepep, _fused_append_remap_shared_experts_deepep_kernel, _fill_padded_rows_kernel, _fill_padded_rows, _can_fuse_padded_region

关键源码片段

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

核心变更文件，添加了融合追加+重映射的Triton内核及其包装函数，是性能优化的核心实现。

```
# 在 fused_moe_triton_kernels.py 中新增
@triton.jit
def _fused_append_remap_shared_experts_deepep_kernel(
    topk_ids_ptr,
    topk_weights_ptr,
    out_ids_ptr,
    out_weights_ptr,
    shared_id_base, # 运行时标量：ep_rank * num_local_experts + num_local_routed
    num_local_routed, # 运行时标量：每 rank 的路由专家数（用于间隙插入）
    scale_factor, # 运行时标量：共享专家权重
    K: tl.constexpr,
    S: tl.constexpr,
):
    """在单次遍历中追加共享专家并应用 DeepEP 交错重映射。
```

```
    等效于依次执行 fused_append_shared_experts() 和 _remap_topk_for_deepep()，
    但重映射计算在已载入寄存器的行上完成，仅需少量 ALU 操作，
    避免了每层约 6 次额外 eager 内核启动（div_floor / add / arange / fill / copy）。

    路由 ID:  e -> e + e // num_local_routed   （为共享槽位插入间隙）
    共享 ID:  shared_id_base + arange(S)       （每个共享槽位一个 ID）
    共享权重: scale_factor                     （aiter 上为 1.0；否则为 1/rsf）
    """
    pid = tl.program_id(0)
    ids_row_ptr = pid * K
    out_ids_row_ptr = pid * (K + S)
    offs_k = tl.arange(0, K)
    ids = tl.load(topk_ids_ptr + ids_row_ptr + offs_k)
    ws = tl.load(topk_weights_ptr + ids_row_ptr + offs_k)
    # DeepEP 交错布局：将每个路由 ID 移过其前面的共享槽位
    ids = ids + ids // num_local_routed
    tl.store(out_ids_ptr + out_ids_row_ptr + offs_k, ids)
    tl.store(out_weights_ptr + out_ids_row_ptr + offs_k, ws)
    offs_s = tl.arange(0, S)
    shared_ids = tl.cast(shared_id_base + offs_s, ids.dtype)
    shared_ws = tl.full([S], scale_factor, dtype=ws.dtype)
    tl.store(out_ids_ptr + out_ids_row_ptr + K + offs_s, shared_ids)
    tl.store(out_weights_ptr + out_ids_row_ptr + K + offs_s, shared_ws)


def fused_append_remap_shared_experts_deepep(
    topk_ids,
    topk_weights,
    num_fused_shared_experts,
    scale_factor,
    shared_id_base,
    num_local_routed,
):
    """融合追加+DeepEP 重映射的包装函数。

    替换 aiter/DeepEP 路径上的 fused_append_shared_experts() + _remap_topk_for_deepep() 对。
    主机端计算标量 remap 参数，使内核保持无分支。
    """
    m, k = topk_ids.shape
    s = int(num_fused_shared_experts)
    if s <= 0:
        return topk_ids, topk_weights
    out_ids = torch.empty((m, k + s), dtype=topk_ids.dtype, device=topk_ids.device)
    out_weights = torch.empty(
        (m, k + s), dtype=topk_weights.dtype, device=topk_weights.device
    )
    _fused_append_remap_shared_experts_deepep_kernel[(m,)](
        topk_ids,
        topk_weights,
```

```
        out_ids,
        out_weights,
        shared_id_base,
        num_local_routed,
        scale_factor,
        K=k,
        S=s,
        num_warps=1,
    )
    return out_ids, out_weights
```

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

修改了 topk 后处理逻辑，接入融合内核并新增填充行 Triton kernel。同时引入了环境变量和输入校验改进。

```
# 在 topk.py 中新增
@triton.jit
def _fill_padded_rows_kernel(
    out_ptr,
    num_token_non_padded_ptr, # 设备内存中的标量，表示非填充 token 数
    n_cols,
    fill_value,
    stride_row,
    BLOCK_COLS: tl.constexpr,
):
    """将填充行（row >= num_token_non_padded）设置为 fill_value。

    每个程序处理一行；从设备内存读取有效 token 数，
    因此可安全捕获到 CUDA/HIP 图中。
    """
    row = tl.program_id(0)
    n_valid = tl.load(num_token_non_padded_ptr)
    if row >= n_valid:
        cols = tl.arange(0, BLOCK_COLS)
        mask = cols < n_cols
        ptrs = out_ptr + row * stride_row + cols
        fill = tl.full((BLOCK_COLS,), fill_value, dtype=out_ptr.dtype.element_ty)
        tl.store(ptrs, fill, mask=mask)


def _can_fuse_padded_region(x: torch.Tensor) -> bool:
    # 检查张量是否为行主序且连续列，满足内核要求
    return x.dim() == 2 and x.stride(1) == 1


def _fill_padded_rows(
    x: torch.Tensor,
    num_token_non_padded: torch.Tensor,
    fill_value,
```

```
) -> None:
    """使用单个 Triton launch 设置填充行。

    替换 eager 的 arange + (>=) + boolean index_put_ 序列，
    该序列每次调用会发起多次启动延迟受限的内核。
    """
    if not isinstance(num_token_non_padded, torch.Tensor):
        raise TypeError("num_token_non_padded must be a torch.Tensor")
    if num_token_non_padded.numel() != 1:
        raise ValueError(
            "num_token_non_padded must be a single-element tensor, got shape "
            f"{tuple(num_token_non_padded.shape)}"
        )
    if num_token_non_padded.dtype.is_floating_point:
        raise TypeError(
            "num_token_non_padded must be an integer tensor, got "
            f"{num_token_non_padded.dtype}"
        )
    if num_token_non_padded.device != x.device:
        raise ValueError("num_token_non_padded and x must be on the same device")
    n_rows, n_cols = x.shape
    _fill_padded_rows_kernel[(n_rows,)](
        x,
        num_token_non_padded,
        n_cols,
        fill_value,
        x.stride(0),
        BLOCK_COLS=triton.next_power_of_2(n_cols),
    )
```

test/registered/moe/test_fused_append_remap_deepep.py

新增的单元测试文件，全面覆盖融合内核的正确性、与 eager 等价性、共享权重逻辑和空操作场景，注册了 AMD 和 CUDA CI。

```
# 在 test_fused_append_remap_deepep.py 中
import unittest
import torch
from sglang.srt.layers.moe.moe_runner.triton_utils.fused_moe_triton_kernels import (
    fused_append_remap_shared_experts_deepep,
    fused_append_shared_experts,
)
from sglang.srt.layers.moe.topk import TopKConfig, _remap_topk_for_deepep
from sglang.srt.runtime_context import get_parallel
from sglang.srt.utils import get_device
from sglang.test.ci.ci_register import register_amd_ci, register_cuda_ci
from sglang.test.test_utils import CustomTestCase

register_cuda_ci(est_time=20, stage="base-b", runner_config="1-gpu-large")
register_amd_ci(est_time=20, suite="stage-b-test-1-gpu-small-amd")
```

```
def _reference_append_remap(
    topk_ids, topk_weights, s, scale_factor, shared_id_base, num_local_routed
):
    """纯 torch golden reference，反映内核的文档约定。"""
    m, k = topk_ids.shape
    out_ids = torch.empty((m, k + s), dtype=topk_ids.dtype, device=topk_ids.device)
    out_w = torch.empty(
        (m, k + s), dtype=topk_weights.dtype, device=topk_weights.device
    )
    out_ids[:, :k] = topk_ids + topk_ids // num_local_routed
    out_w[:, :k] = topk_weights
    shared = shared_id_base + torch.arange(s, device=topk_ids.device)
    out_ids[:, k:] = shared.to(topk_ids.dtype)
    out_w[:, k:] = scale_factor
    return out_ids, out_w

@unittest.skipUnless(torch.cuda.is_available(), ...)
class TestFusedAppendRemapDeepEP(CustomTestCase):
    CASES = [
        (1, 8, 256, 8, 0, 1),
        (4, 8, 256, 8, 7, 1),
        (17, 8, 264, 8, 3, 1),
        (128, 16, 128, 4, 2, 2),
    ]
    def test_matches_golden_reference(self):
        """验证内核输出与文档重映射数学一致。"""
        for m, k, npr, ep_size, ep_rank, s in self.CASES:
            with self.subTest(m=m, k=k, ...):
                shared_id_base, num_local_routed = self._shared_id_base(
                    npr, ep_size, ep_rank, s
                )
                topk_ids, topk_weights = self._make_inputs(m, k, npr)
                got_ids, got_w = fused_append_remap_shared_experts_deepep(
                    topk_ids, topk_weights, s, 1.0, shared_id_base, num_local_routed
                )
                exp_ids, exp_w = _reference_append_remap(
                    topk_ids, topk_weights, s, 1.0, shared_id_base, num_local_routed
                )
                self.assertTrue(torch.equal(got_ids, exp_ids))
                self.assertTrue(torch.allclose(got_w, exp_w))
    # 其他测试省略 ...
```

评论区精华

Review 中主要讨论了以下问题：

- Triton 导入方式：HaiShaw 指出环境总是有 triton，建议将导入移到文件顶部（参考 fp8_utils.py），作者随后清理。

- 测试 patching 错误：Copilot 发现测试中使用 patch.object 模拟 get_moe_expert_parallel_world_size/rank 无效，因为被测试函数实际调用 get_parallel().moe_ep_size。建议改用 get_parallel().override(...)。作者提交修复。
- 内核文档不精确：Copilot 指出内核 docstring 说 Shared IDs: shared_id_base + s 但实现是 shared_id_base + arange(S)，作者更新了文档。
- 注释误导：Copilot 指出注释提及不存在的 _remap_topk_for_deepep aiter branch，作者修改了注释。
- 输入校验使用 assert：Copilot 提醒 _fill_padded_rows 使用 assert 在 python -O 下会被跳过，应改为显式异常，作者改用 raise。 以上问题均已在后续提交中解决，PR 最终获得 HaiShaw 批准。
- Triton 导入方式简化 (style): 作者后续提交（acd93c5）将导入移至顶部并移除 _HAS_TRITON 标志。
- 测试中错误地 patching 已移除的函数 (testing): 作者提交（ab70cdd）使用 override() 上下文管理器修复，确保 EP 拓扑正确。
- 内核 docstring 中共享 ID 描述不精确 (documentation): 作者提交（06412fe）修正 docstring。
- 注释错误引用不存在的 aiter 分支 (documentation): 作者提交（06412fe）修改注释，避免误导。
- 输入校验使用 assert 可能被跳过 (correctness): 作者提交（06412fe）改用 raise TypeError/ValueError。

风险与影响

- 风险：
  1. 数值精度风险：虽然融合内核在构造上保持位一致，但 SGLANG_MORI_NO_PAD_MASK 环境变量（默认关闭）会改变填充掩码行为，需经过精确度验证才能默认启用。PR 已在 DeepSeek-R1 上验证 GSM8K 准确率 95.0%，风险较低。
  2. 平台特定风险：新内核仅用于 AMD aiter + DeepEP 路径，其他平台（CUDA、NPU 等）不受影响。若 AMD 环境配置不正确（如缺少 aiter），将回退到原路径。
  3. 测试覆盖：新增单元测试验证了多种形状和 EP 配置下的等价性，但未覆盖端到端多 layer 场景。注册的 CI 测试在 AMD 和 CUDA 上运行，但 SGLANG_MORI_NO_PAD_MASK=1 路径的测试未包含，需注意。
  4. 性能回归可能：融合内核使用 num_warps=1，对于极大 token 数可能效率略低；但原 append 内核也是 num_warps=1，且消除了额外 launch，整体应更优。 - 影响：直接提升 AMD MI350 系列上 DeepSeek-R1 等模型的 MoE 推理吞吐量，降低每层 MoE 路由的 kernel launch 开销。影响范围限定于启用了 SGLANG_USE_AITER + SGLANG_AITER_MOE 的 AMD HIP 路径，不影响其他后端。对于使用 DeepEP 通信的 MoE 模型，融合内核可减少约 6 次 / 层的显式设备同步开销，在高频 MoE 层场景下收益明显。同时新增的 _fill_padded_rows 内核可被其他路径复用（如 NPU 也引入了类似 mask_topk_ids 分支）。环境变量 SGLANG_MORI_NO_PAD_MASK 为未来零开销掩码

提供了可选途径。 - 风险标记：核心路径变更，AMD 专有优化，新环境变量默认关闭，缺少覆盖 SGLANG_MORI_NO_PAD_MASK 的测试

关联脉络

- PR #28237 [AMD] fix(moe): correct fused shared-expert scaling on aiter/DeepEP path (mori all-to-all): 同一个 AMD MoE 路径上的修复，修正了 fused shared-expert 权重缩放，本 PR 在此基础上进一步融合 append+remap 内核，解决了相同路径的性能和正确性累积问题。