PR #25678 完整报告

sgl-project/sglang

[MoE Refactor] deprecate forward_npu and NpuFuseEPMoE

合并时间：2026-05-22 01:25

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/25678

执行摘要

- 一句话：废弃 NPU 专用 MoE 前向路径和调度器，统一至 FusedMoE 流水线
- 推荐动作：值得精读，特别是 forward_fuseep 作为 free function 绕过调度器的模式。设计决策如将 ascend_fuseep 路由到 StandardDispatcher 占位、在 quant_method 中拦截 DeepEP 输出等，展示了如何在统一架构中嵌入硬件专用路径。对于需要扩展 SGLang MoE 后端的开发者，此 PR 是一个很好的模板。

功能与动机

继续 MoE 重构路线图（#8715），弃用 DeepEPMoE.forward_npu 和 NpuFuseEPMoE，以便 NPU 路径符合统一的 FusedMoE.forward -> dispatcher.dispatch -> quant_method.apply -> dispatcher.combine 管道，减少代码冗余并降低维护成本。引用 PR body: 'Deprecates DeepEPMoE.forward_npu and the NpuFuseEPMoE class so the NPU paths fit the unified pipeline.'

实现拆解

1. 废弃专有前向方法：在 python/sglang/srt/layers/moe/ep_moe/layer.py 中删除了 DeepEPMoE.forward_npu 方法和 NpuFuseEPMoE 类。将 __init__ 中 _is_npu 分支的 deprecate_flag 从 False 改为 True，使 NPU + DeepEP 路径不再经过旧的 run_moe_core 分支，而是路由到基类 FusedMoE 的 quant_method.apply 路径。
2. 新增 ascend_fuseep 专用自由函数：创建了 python/sglang/srt/hardware_backend/npu/moe/fuseep.py，包含 forward_fuseep 和 process_fuseep_weights 等函数，替代旧的 NpuFuseEPDispatcher.dispatch 和 NpuFuseEPMoE._process_weights_after_loading。
3. 修改调度器创建和 Forward 跳转：在 python/sglang/srt/layers/moe/fused_moe_triton/layer.py 中，将 ascend_fuseep 路由到 StandardDispatcher（占位，实际不调用），并在 FusedMoE.forward 中增加早期返回调用 forward_fuseep。
4. 新增 NPU 量化方法拦截：在 python/sglang/srt/hardware_backend/npu/quantization/fused_moe_method_npu.py 中添加 maybe_apply_deepep_npu 和 maybe_apply_fuseep_weights，前者拦截 DeepEP 分派输出，后者在权重加载后应用 FuseEP 布局（当前仅限 NPUW8A8Int8DynamicMoEMethod，因 FUSED_DEEP_MOE 内核仅支持 W8A8 INT8）。

5. BF16 未量化 DeepEP 路径：在 python/sglang/srt/layers/quantization/unquant.py 的 UnquantizedFusedMoEMethod 中添加 _forward_npu_deepep 分支，处理未量化的 BF16 DeepEP 输入。
6. 删除不再需要的调度器：删除 python/sglang/srt/layers/moe/token_dispatcher/fuseep.py 并从 __init__.py 中移除导出。
7. Review 后的精简：根据 reviewer 反馈回退了在 CompressedTensorsFusedMoEMethod 和 ModelSlimFusedMoEMethod 中添加的 wrapper-level 钩子，因为这些方法通过 scheme->kernel 委托链已可到达 NPU 方法中的拦截逻辑。

关键文件：

- python/sglang/srt/hardware_backend/npu/moe/fuseep.py（模块 NPU 后端；类别 source；类型 dependency-wiring；符号 _get_fuseep_buffer, forward_fuseep, _permute_w13_weight_scale, _reshape_w13_weight）：新增文件，定义 forward_fuseep 和 process_fuseep_weights 等核心自由函数，替代废弃的 NpuFuseEPDispatcher 和 NpuFuseEPMoE。
- python/sglang/srt/layers/moe/ep_moe/layer.py（模块 MoE 核心；类别 source；类型 dependency-wiring；符号 forward_npu, NpuFuseEPMoE, init, forward）：核心修改文件，删除 forward_npu 方法和 NpuFuseEPMoE 类，调整 deprecate_flag 使 NPU 路径统一至基类。
- python/sglang/srt/layers/moe/token_dispatcher/fuseep.py（模块 调度器；类别 source；类型 deletion；符号 FuseEPDispatchOutput, format, FuseEPCombineInput, NpuFuseEPDispatcher）：删除文件，移除 NpuFuseEPDispatcher 类和相关数据结构，这些已由新自由函数替代。
- python/sglang/srt/hardware_backend/npu/quantization/fused_moe_method_npu.py（模块 量化方法；类别 source；类型 core-logic；符号 maybe_apply_deepep_npu, maybe_apply_fuseep_weights, _maybe_apply_deepep, _maybe_apply_fuseep_weights）：核心逻辑修改，添加 maybe_apply_deepep_npu 和 maybe_apply_fuseep_weights 辅助函数，拦截 DeepEP 分派输出和应用 FuseEP 权重布局。
- python/sglang/srt/layers/moe/fused_moe_triton/layer.py（模块 MoE 核心；类别 source；类型 dependency-wiring）：调整调度器创建和 forward 跳转，将 ascend_fuseep 路由到 StandardDispatcher 并在 forward 中增加早期返回。
- python/sglang/srt/layers/quantization/unquant.py（模块 量化方法；类别 source；类型 core-logic；符号 _forward_npu_deepep）：添加 BF16 未量化 DeepEP 路径 _forward_npu_deepep，处理老的 forward_npu 中 BF16 分支。
- python/sglang/srt/layers/moe/token_dispatcher/__init__.py（模块 调度器；类别 source；类型 dependency-wiring）：移除对废弃调度器类的导出。
- python/sglang/srt/layers/quantization/compressed_tensors/compressed_tensors.py（模块 量化方法；类别 source；类型 core-logic）：回滚 wrapper-level 的 fuseep 权重钩子，遵循 reviewer 建议。

关键符号：forward_fuseep, maybe_apply_deepep_npu, maybe_apply_fuseep_weights, _forward_npu_deepep, process_fuseep_weights, create_moe_dispatcher

关键源码片段

python/sglang/srt/hardware_backend/npu/moe/fuseep.py

新增文件，定义 forward_fuseep 和 process_fuseep_weights 等核心自由函数，替代废弃的 NpuFuseEPDispatcher 和 NpuFuseEPMoE。

```
# _get_fuseep_buffer: 获取 DeepEP buffer，强制低延迟模式
def _get_fuseep_buffer(layer: "FusedMoE"):
    DeepEPBuffer.set_dispatch_mode_as_low_latency()
    return DeepEPBuffer.get_deepep_buffer(
        get_tp_group().device_group,
        layer.hidden_size,
        _PARAMS_BYTES, # bf16，Ascend 不支持 fp16
        DeepEPMode.LOW_LATENCY,
        envs.SGLANG_DEEPEP_NUM_MAX_DISPATCH_TOKENS_PER_RANK.get(),
        layer.num_experts,
    )


# forward_fuseep: 驱动 fused dispatch + GEMM + combine 操作
def forward_fuseep(
    layer: "FusedMoE",
    hidden_states: torch.Tensor,
    topk_output: "TopKOutput",
) -> torch.Tensor:
    buf = _get_fuseep_buffer(layer)
    hidden_states, _ = buf.fused_deep_moe(
        hidden_states,
        topk_idx=topk_output.topk_ids,
        topk_weights=topk_output.topk_weights,
        gmm1_permuted_weight=layer.w13_weight,
        gmm1_permuted_weight_scale=layer.w13_weight_scale,
        gmm2_weight=layer.w2_weight,
        gmm2_weight_scale=layer.w2_weight_scale,
        num_max_dispatch_tokens_per_rank=(
            envs.SGLANG_DEEPEP_NUM_MAX_DISPATCH_TOKENS_PER_RANK.get()
        ),
        num_experts=layer.num_experts,
        fuse_mode=envs.SGLANG_NPU_FUSED_MOE_MODE.get(),
    )
    return hidden_states
```

python/sglang/srt/hardware_backend/npu/quantization/fused_moe_method_npu.py

核心逻辑修改，添加 maybe_apply_deepep_npu 和 maybe_apply_fuseep_weights 辅助函数，拦截 DeepEP 分派输出和应用 FuseEP 权重布局。

```
# maybe_apply_deepep_npu: 拦截 DeepEP 分派输出，路由到 NPU 的计算路径
```

```
def maybe_apply_deepep_npu(
    quant_method,
    layer: torch.nn.Module,
    dispatch_output: "DispatchOutput",
) -> Optional["CombineInput"]:
    # 如果不是 DeepEP 格式，返回 None，让调用者继续标准路径
    if not dispatch_output.format.is_deepep():
        return None

    # Ascend 的 Dispatch & Combine 不支持 fp16，统一输出为 bf16
    output_dtype = torch.bfloat16
    group_list_type = 1

    # 根据分派格式（normal vs low-latency）解析不同字段
    if DispatchOutputChecker.format_is_deepep_normal(dispatch_output):
        (
            hidden_states,
            hidden_states_scale,
            _,
            _,
            num_recv_tokens_per_expert,
        ) = dispatch_output
        group_list = torch.tensor(
            num_recv_tokens_per_expert,
            dtype=torch.int64,
            device=hidden_states.device,
        )
        combine_cls = DeepEPNormalCombineInput
    else:
        (
            hidden_states,
            hidden_states_scale,
            _,
            _,
            group_list,
            _,
        ) = dispatch_output
        group_list = group_list.to(torch.int64)
        combine_cls = DeepEPLLCombineInput

    # 调用 quant_method 的 apply_without_routing_weights 计算
    hidden_states = quant_method.apply_without_routing_weights(
        layer,
        hidden_states,
        hidden_states_scale,
        group_list_type,
        group_list,
        output_dtype,
    )
```

```
    # 包装成正确的 CombineInput 返回
    return combine_cls(
        hidden_states=hidden_states,
        topk_ids=dispatch_output.topk_ids,
        topk_weights=dispatch_output.topk_weights,
    )
```

评论区精华

主要讨论集中在：

- FuseEP 仅支持 W8A8 INT8：Reviewer @OrangeRedeng 指出 FUSED_DEEP_MOE 内核文档明确只支持 W8A8 INT8，因此不应在 W4A4、W4A8 等方法中添加 _maybe_apply_fuseep_weights（#25678#discussion_r3264992853）。作者采纳意见，在 commit e8897eac0 中回退了对非 W8A8 方法的修改。
- Wrapper-level 钩子不必要：Reviewer 认为 CompressedTensorsFusedMoEMethod 和 ModelSlimFusedMoEMethod 不需要直接添加 maybe_apply_fuseep_weights，因为底层 NPU 量化方法（通过 scheme->kernel 链）已经处理（#25678#discussion_r3265176447）。作者同意并回退。
- Forward 构造器方向：@OrangeRedeng 提到正在开发 PR #25663，允许以构造器方式组装 forward，当收到 deepep 分派输出时取消单独量化路径。作者表示会关注。
  - FuseEP INT8 限制 (correctness): 作者接受并限定在 NPUW8A8Int8DynamicMoEMethod 中保留 _maybe_apply_fuseep_weights，其他方法回退。
  - 避免 wrapper 层侵入 (design): 作者回退了 CompressedTensorsFusedMoEMethod 和 ModelSlimFusedMoEMethod 中的修改，保持 wrapper 层清洁。
  - Forward 构造器方向 (design): 作者表示会关注后续 PR，当前设计足够。

风险与影响

- 风险：具体风险包括：
  1. NPU DeepEP 路径回归：重构后 NPU + DeepEP 走向 FusedMoE.forward_impl，增加了 use_symmetric_memory 上下文和 slice-to-origin_hidden_states_dim，虽然 PR 作者认为是 no-op，但需 NPU 验证。Reviewer @OrangeRedeng 在 A3 服务器上确认 DeepEP 工作正常。
  2. FuseEP 路径环境依赖：Reviewer 发现一个环境变量组合错误导致 FuseEP 结果异常，表明该路径对环境变量敏感，本 PR 未改善该敏感性问题。
  3. 测试覆盖不足：作者无 NPU 访问权限，没有运行完整的精度测试，依赖 NPU CI 和 reviewer 手动验证。虽然 reviewer 最终确认工作正常，但 CI 测试可能未覆盖所有模型和配置组合。
  4. 配置兼容性：废弃了 NpuFuseEPDispatcher，但 create_moe_dispatcher 中 ascend_fuseep 分支被替换为 StandardDispatcher，如果外部代码依赖旧的调度器类型，可能出现兼容问题（但调度实际在 forward_fuseep 中完成，影响有限）。 - 影响： - 用户影响：对使用 Ascend NPU 运行 MoE 模型的用户，此 PR 不改变外部行为，但内

部路径统一可能带来更好的兼容性和维护性。使用 --moe-a2a-backend ascend_fuseep 的用户将自动切换到新的 forward_fuseep 路径。使用 DeepEP 分派的用户路径也统一。 - 系统影响：删除约 364 行代码，新增约 359 行，净减少约 5 行，整体规模稳定。模块间职责更清晰：NPU 专有逻辑集中在 hardware_backend/npu/ 下。 - 团队影响：降低 NPU 后端的维护成本，后续添加新量化方法或后端时只需遵循统一模式，不必再创建自定义调度器和 forward 方法。 - 风险标记：核心路径变更，NPU 环境敏感，无作者直接测试

关联脉络

- 暂无明显关联 PR