PR #25663 完整报告

sgl-project/sglang

[MoE Refactor] [NPU] Refactor Ascend MoE implementation to reduce code duplication and align with community design

合并时间：2026-07-15 19:59

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/25663

执行摘要

- 一句话：重构 NPU MoE 为 5 组件解耦，对齐社区 All-to-All 架构
- 推荐动作：建议所有 NPU 后端开发者深度阅读此 PR，尤其是 5 组件的接口设计（base.py 中的 MoeRunnerCore 及其子类）和 AscendTPDispatcher 的 dispatch 流程。review 中关于循环导入的教训也值得其他模块借鉴。PR 本身合并后需密切监控 NPU MoE 各项 CI job 的稳定性。

功能与动机

当前 Ascend NPU 上的 MoE 实现存在两大问题：

1) 高代码重复 – MoE 前向逻辑在 10 多个量化文件中重复出现，每次修复或升级 kernel 都需要同步修改所有副本，维护开销极高； 2) 与社区设计偏离 – dispatch 和 grouped-GEMM 被硬编码在单片 apply 流程中，无法融入标准 All-to-All 后端和 MoE Runner 模式（见 Issue #8715 MoE Refactor 路线图）。

实现拆解

1. 删除遗留单片代码：移除 fused_moe_method_npu.py（1217 行），该文件包含了多个重复的 npu_fused_experts_* 函数。
2. 定义 5 个抽象组件及 NPU 实现：在 hardware_backend/npu/moe/ 下创建 init_routing.py、hidden_states_quant.py、matmul.py、activation.py、finalize_routing.py，每个组件提供 Base* 抽象类和一至多个 NPU 具体实现（如 NPUMoEInitRouting_v1/v2、HiddenStatesDynamicQuant、NPUSwigluQuant）。
3. 新建 Ascend MoE Runner：在 layers/moe/moe_runner/ascend.py 中创建 AscendRunnerInput/AscendRunnerOutput 数据和 AscendRunnerCore 类，负责在运行时根据模型配置和量化类型选择合适的组件组合，执行统一的 run 方法。
4. 新建 Ascend TP Dispatcher：在 layers/moe/token_dispatcher/ascend_tp.py 中创建 AscendTPDispatcher，封装初始路由和最终路由逻辑，对齐社区 BaseDispatcher 接口，支持 GGUF 下的 TP all-gather 包装。
5. 改造量化 Scheme：逐一修改 AWQ、ModelSlim（W4A4 / W4A8 / W8A8）等量化模块，使其通过 create_moe_runner 和 apply_weights/apply_without_routing_weights 接口与新 Runner 集成，权重处理逻辑被抽取为 _NPUMoEMethodBase 的工具方法。

6. 补充激活函数组件：在 activation.py 中实现 NPUSwiglu、NPUSwigluQuant、NPUSwigluDeepEPKernel、NPUGeluAndMul 等变体，均继承自 BaseActivation。
7. 测试与文档：更新 NPU 相关测试用例（如 test_npu_minimax_m2_5_w8a8_*），新增 TensorBoard/Waifu 等文档说明新架构。

关键文件：

- python/sglang/srt/hardware_backend/npu/quantization/fused_moe_method_npu.py（模块 旧 MoE 单片；类别 source；类型 deletion；符号 npu_fused_experts_w4a4, npu_fused_experts, npu_fused_experts_w8a8_decode, npu_fused_moe_without_routing_weights_bf16）：旧单片 MoE 实现的核心文件，包含所有重复的 npu_fused_experts_* 方法，共 1217 行，已被完全删除，是重构的首要目标。
- python/sglang/srt/hardware_backend/npu/quantization/moe_methods.py（模块 MoE 量化统一入口；类别 source；类型 dependency-wiring；符号 fused_moe_npu, _NPUMoEMethodBase, init, _set_dispatcher_output_dtype）：新增核心文件，包含 _NPUMoEMethodBase 基类和所有 NPU 量化方法的统一入口（apply/apply_without_routing_weights），以及废弃的 fused_moe_npu 兼容函数。
- python/sglang/srt/layers/moe/moe_runner/ascend.py（模块 Ascend Runner；类别 source；类型 core-logic；符号 AscendRunnerInput, runner_backend, AscendRunnerOutput, AscendRunnerCore）：新增的 Ascend MoE Runner 核心实现，包含 AscendRunnerInput/AscendRunnerOutput 数据类、AscendRunnerCore（继承自 MoeRunnerCore）以及 run 方法，是整个组件化架构的执行中枢。
- python/sglang/srt/hardware_backend/npu/moe/activation.py（模块 激活组件；类别 source；类型 dependency-wiring；符号 BaseActivation, _apply_activation, NPUSwiglu, NPUSwigluQuant）：新增的激活函数组件抽象基类和多种 NPU 具体实现（Swiglu、SwigluQuant、DeepEP 等），支撑 Runner 在运行时动态选择。
- python/sglang/srt/layers/moe/token_dispatcher/ascend_tp.py（模块 Ascend 调度器；类别 source；类型 dependency-wiring；符号 AscendTPDispatchOutput, format, AscendTPCombineInput, AscendTPDispatcher）：新增的 Ascend TP Dispatcher，封装了初始路由（npu_moe_init_routing_v2）和最终路由（NPUFinalizeRouting），对齐社区 BaseDispatcher 接口。
- python/sglang/srt/layers/quantization/modelslim/schemes/modelslim_w4a4_int4_moe.py（模块 ModelSlim W4A4；类别 source；类型 data-contract；符号 _create_weight_params, create_moe_runner, apply_weights, apply_without_routing_weights）：ModelSlim W4A4 量化 Scheme 改造，从旧 NPUW4A4Int4DynamicMoEMethod 切换到新 NPUW4A4Int4MoEMethod，并支持新 Runner 的权重创建接口。

关键符号：npu_fused_experts_w4a4, npu_fused_experts, npu_fused_experts_w8a8_decode, fused_moe_npu, _NPUMoEMethodBase.init, AscendRunnerCore.init, AscendRunnerCore.run, BaseActivation._apply_activation, NPUSwiglu._apply_activation, NPUSwigluQuant._apply_activation, NPUSwigluDeepEPKernel._apply_activation, AscendTPDispatcher.dispatch, ModelSlimW4A4Int4MoE.create_weights, ModelSlimW4A4Int4MoE.apply_weights

关键源码片段

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

新增核心文件，包含 _NPUMoEMethodBase 基类和所有 NPU 量化方法的统一入口（apply/apply_without_routing_weights），以及废弃的 fused_moe_npu 兼容函数。

```
# DEPRECATED METHOD – 保留用于向后兼容
# TODO: Remove in future releases
def fused_moe_npu(
    x,
    w1,
    w2,
    topk_output,
    moe_runner_config,
):
    logger.warning_once(
        "The fused_moe_npu method deprecated and will be removed in future releases"
    )
    topk_weights, topk_ids, _ = topk_output
    original_dtype = x.dtype
    num_tokens = x.shape[0]
    topk_weights = topk_weights.to(x.dtype)
    topk_ids = topk_ids.to(torch.int32)
    num_experts = w1.shape[0]
    top_k = topk_weights.shape[-1]
    row_idx_len = num_tokens * top_k
    row_idx = (
        torch.arange(0, row_idx_len, dtype=torch.int32, device=topk_weights.device)
        .view(top_k, -1)
        .permute(1, 0)
        .contiguous()
    )

    # 使用 NPU 原生 init_routing v1
    hidden_states, expanded_row_idx, expanded_expert_idx = (
        torch.ops.npu.npu_moe_init_routing(
            x, row_idx=row_idx, expert_idx=topk_ids, active_num=num_tokens
        )
    )
    expert_tokens = torch.ops.npu.npu_moe_compute_expert_tokens(
        expanded_expert_idx, num_experts
    ).to(torch.int64)

    # gmm1: gate_up_proj，注意 weight 需 permute
    hidden_states = torch.ops.npu.npu_grouped_matmul(
        x=[hidden_states],
        weight=[w1.permute(0, 2, 1)],
        bias=None,
        split_item=2,
```

```
        group_list_type=0,
        group_type=0,
        group_list=expert_tokens,
        output_dtype=original_dtype,
    )[0]

    # 根据配置选择激活函数
    if moe_runner_config.activation == "silu":
        hidden_states = torch.ops.npu.npu_swiglu(hidden_states)
    else:
        from sglang.srt.layers.activation import GeluAndMul
        hidden_states = GeluAndMul()(hidden_states)

    # gmm2: down_proj
    hidden_states = torch.ops.npu.npu_grouped_matmul(
        x=[hidden_states],
        weight=[w2.permute(0, 2, 1)],
        bias=None,
        split_item=2,
        group_list_type=0,
        group_type=0,
        group_list=expert_tokens,
        output_dtype=original_dtype,
    )[0]

    final_hidden_states = torch.ops.npu.npu_moe_finalize_routing(
        hidden_states,
        skip1=None,
        skip2=None,
        bias=None,
        scales=topk_weights,
        expanded_src_to_dst_row=expanded_row_idx,
        export_for_source_row=topk_ids,
    )
    return final_hidden_states
```

python/sglang/srt/layers/moe/moe_runner/ascend.py

新增的 Ascend MoE Runner 核心实现，包含 AscendRunnerInput/AscendRunnerOutput 数据类、AscendRunnerCore（继承自 MoeRunnerCore）以及 run 方法，是整个组件化架构的执行中枢。

```
@dataclass
class AscendRunnerInput(RunnerInput):
    """NPU runner 的输入 bundle，包含已排好序的 hidden_states 和专家计数信息。"""
    hidden_states: torch.Tensor
    hidden_states_scale: Optional[torch.Tensor] # 非量化时为 None
    expert_tokens: torch.Tensor # 每个专家分配的 token 数
    group_list_type: int # 0 或 1，传给 NPU grouped matmul 操作
```

```
    @property
    def runner_backend(self) -> MoeRunnerBackend:
        return MoeRunnerBackend.ASCEND


class AscendRunnerCore(MoeRunnerCore):
    runner_backend = MoeRunnerBackend.ASCEND

    def __init__(self, config: MoeRunnerConfig):
        super().__init__(config)
        kernel = config.layer.w2_kernel

        if get_moe_a2a_backend().is_deepep():
            # DeepEP 路径：使用统一 kernel 决定是否需要量化
            is_quant_kernel = isinstance(
                kernel, (NPUW4A8Int8MoEMethod, NPUW8A8Int8MoEMethod)
            )
            self.activation = NPUSwigluDeepEPKernel(need_quant=is_quant_kernel)
        else:
            # 非 DeepEP（ascend_tp）路径
            if isinstance(kernel, (NPUW4A8Int8MoEMethod, NPUW8A8Int8MoEMethod)):
                inner = NPUSwigluQuant()
            else:
                # 根据配置选择具体的激活变体
                if config.activation == "npu_swiglu_oai":
                    inner = NPUSwigluOAI(moe_runner_config=config)
                elif config.activation == "silu":
                    if config.gemm1_clamp_limit is not None:
                        inner = NPUSwigluStepAndMul(clamp_limit=config.gemm1_clamp_limit)
                    else:
                        inner = NPUSwiglu()
                else:
                    inner = NPUGeluAndMul()

            if getattr(config, "use_tp_all_gather_activation", False):
                self.activation = AllGatherActivationWrapper(inner, dim=-1)
            else:
                self.activation = inner

    def run(self, runner_input: AscendRunnerInput, quant_info: AscendQuantInfo,
            running_state: dict, hooks=None) -> AscendRunnerOutput:
        """执行 MoE 层：w13 -> 激活 -> w2 -> 最终化。"""
        x = runner_input.hidden_states
        original_dtype = torch.float16 if x.dtype == torch.float16 else torch.bfloat16
        expert_tokens = runner_input.expert_tokens
        group_list_type = runner_input.group_list_type

        # w13 (gate & up) 投影
        hidden_states = self.config.layer.w13_kernel.apply(
```

```
        quant_info, x, expert_tokens,
        pertoken_scale=runner_input.hidden_states_scale,
        output_dtype=original_dtype,
        weight_prefix="w13"
    )

    # 激活函数（可能包含量化）
    hidden_states, pertoken_scale = self.activation._apply_activation(
        hidden_states, expert_tokens, group_list_type
    )

    # w2 (down) 投影
    hidden_states = self.config.layer.w2_kernel.apply(
        quant_info, hidden_states, expert_tokens,
        pertoken_scale=pertoken_scale,
        output_dtype=original_dtype,
        weight_prefix="w2"
    )

    # 最终化路由（合并结果还原 token 顺序）
    final_hidden_states = self.config.layer.finalize_kernel._finalize_routing(
        hidden_states, quant_info, runner_input.topk_weights,
        runner_input.expanded_row_idx, runner_input.topk_ids
    )

    return AscendRunnerOutput(hidden_states=final_hidden_states)
```

评论区精华

- ch-wan 指出 P0 问题：顶层 import Ascend runner 导致循环导入，import sglang.srt.layers.moe 直接失败。作者随后将 import 移至函数内部或延迟导入，解决了核心模块加载问题。
- ch-wan 的回归分析：将新旧 W4A4 打包路径、AWQ 权重处理等差异逐一对比，确认新增的 SGLANG_NPU_W4A4_NEW_PACKING 门控避免对现有方案的影响，AWQ 恢复了 #10158 的正确实现。
- ping1jing2 的命名 & 风格建议：要求将 torch_npu 统一改为 ascend_tp，避免引入新词汇；为 awq_kernels.py 中的 magic number 添加详细注释；建议用 __call__ 替代 forward 等。作者均配合修改。
- gemini-code-assist 类型提示修复：指出 init_routing.py 中抽象方法返回值标注为 3 元组但实现返回 4 元组，建议统一为 Tuple[..., Optional[torch.Tensor]]。
  - 顶层 import 导致循环导入（P0） (correctness): 作者将 import 移至函数内部或使用延迟导入，解决了循环依赖。
  - W4A4 新打包路径回归分析 (design): 确定为有意扩展，门控机制避免了回归风险。
  - 命名风格统一（torch_npu → ascend_tp） (style): 已按建议修改。
  - Magic Number 注释与公式说明 (documentation): 作者补充了详细注释。

风险与影响

- 风险：
  1. 循环导入风险（已修复）：顶层 import 曾导致部分环境无法导入 sglang.srt.layers.moe。修复后需确保无残留。
  2. W4A4 新打包路径兼容性：W4A4 的通用逐前缀 matmul 合约受环境变量 SGLANG_NPU_W4A4_NEW_PACKING 门控，旧路径仍受支持，但若门控逻辑有误可能导致静默精度回退。
  3. DeepEP / FuseEP 集成：新 Runner 的 DeepEP 路径使用了 NPUSwigluDeepEPKernel，但该路径的测试覆盖不足，可能存在运行时 shape 或 dtype 不匹配。
  4. 性能回归：组件化增加了间接层（虚函数调用、配置分支），在极端高频的 MoE 前向路径可能引入微小开销，需 benchmark 验证。
  5. 多量化方案回归：影响 AWQ、ModelSlim、GGUF 等全部 NPU 量化方案，每个方案都需在 NPU 硬件上重新验证精度与吞吐。 - 影响：影响范围：所有使用 Ascend NPU 的 MoE 模型（DeepSeek、Qwen、MiniMax 等），涉及推理核心路径。用户影响：对外 API 和 CLi 参数保持兼容，旧 fused_moe_npu 函数仍保留并标记废弃，用户无感知。团队影响：新架构大幅降低后续添加量化方法或路由 kernel 的维护成本；代码结构清晰，组件可独立测试。
- 风险标记：核心路径变更，循环导入修复，新打包路径门控，多量化兼容性

关联脉络

- PR #8715 [Roadmap] MoE Refactor: 此 PR 是该路线图在 NPU 后端的具体落地，实现了组件化拆分和对齐社区架构的目标。
- PR #10158 [AWQ] AWQ weight processing fix: PR 描述中明确提到 AWQ 处理恢复到了 #10158 的实现，因此是回归回归参考点。