PR #35153 完整报告

vllm-project/vllm

[MoE Refactor] Make SharedExperts class for use with DefaultMoERunner

合并时间：2026-04-01 21:44

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/35153

执行摘要

- 一句话：引入 SharedExperts 类集中管理 MoE 共享专家执行，重构核心运行器。
- 推荐动作：建议深入阅读 SharedExperts 类的实现，理解其如何决策执行顺序和处理流同步；关注 DefaultMoERunner 中 forward_dispatch 的设计，这是整合新类的关键；同时，注意 review 中提及的待优化点，如所有权动态化，以把握未来演进方向。

功能与动机

PR body 指出目的是 'Move shared experts functionality into SharedExperts class'，以使 DefaultMoERunner 和模块化内核能委托 SharedExperts 执行共享专家，确保正确位置和单次执行。讨论中进一步强调需要解决共享专家所有权混乱和执行顺序问题，以支持重叠优化和代码清晰度。

实现拆解

1. 新增 SharedExperts 类：在 shared_experts.py 中定义 SharedExpertsOrder 枚举和 SharedExperts 类，负责共享专家的初始化、执行顺序决策和实际计算。关键符号包括 _determine_shared_experts_order 和 apply 方法，基于配置决定是否使用多流重叠。
2. 重构 DefaultMoERunner：修改 default_moe_runner.py，移除直接处理共享专家的逻辑，改为使用 SharedExperts 实例。引入 forward_dispatch 方法统一路由，并调整 DP 分块处理以集成共享专家。
3. 更新模块化内核：在 modular_kernel.py 中添加 _maybe_apply_shared_experts 方法，使模块化内核能调用 SharedExperts，支持异步执行模式。
4. 清理 FusedMoE 层：修改 layer.py，将 gate 和 shared_experts 的所有权移交给 runner，简化层的职责，移除冗余状态。
5. 测试与配置配套：同步更新了多个测试文件以验证新逻辑，包括 tests/kernels/moe 和 tests/lora/ 中的相关测试，确保重构不影响现有功能。

关键文件：

- vllm/model_executor/layers/fused_moe/runner/shared_experts.py（模块 MoE 运行器；类别 source；类型 core-logic；符号 SharedExpertsOrder, SharedExperts, init, _has_external_experts）：新增 SharedExperts 类，是重构的核心，定义了共享专家的执行逻辑和顺序枚举。

- vllm/model_executor/layers/fused_moe/runner/default_moe_runner.py（模块 MoE 运行器；类别 source；类型 core-logic；符号 _select_forward, use_dp_chunking, _maybe_setup_shared_experts_stream, _replace_quant_method）：核心运行器修改，整合 SharedExperts 并重构 forward 逻辑，影响 MoE 执行入口。
- vllm/model_executor/layers/fused_moe/modular_kernel.py（模块 模块化内核；类别 source；类型 data-contract；符号 supports_async, _maybe_apply_shared_experts, owns_shared_experts）：更新模块化内核以支持 SharedExperts，确保在异步模式下正确集成。
- vllm/model_executor/layers/fused_moe/layer.py（模块 MoE 层；类别 source；类型 data-contract；符号 _init_runner, shared_experts, gate）：清理 FusedMoE 层，移除对共享专家的直接管理，交由 runner 负责所有权。

关键符号：SharedExperts.init, SharedExperts._determine_shared_experts_order, SharedExperts.apply, DefaultMoERunner.forward_dispatch, DefaultMoERunner._apply_shared_experts, FusedMoEPrepareAndFinalizeModular.supports_async

关键源码片段

vllm/model_executor/layers/fused_moe/runner/shared_experts.py

新增 SharedExperts 类，是重构的核心，定义了共享专家的执行逻辑和顺序枚举。

```
from enum import IntEnum
import torch
import vllm.envs as envs

class SharedExpertsOrder(IntEnum):
    # 枚举定义共享专家的执行顺序：无共享专家、外部执行、无重叠、MK内部重叠、多流重叠
    NONE = 0
    EXTERNAL = 1 # 可能因 torch.compile 原因暂时保留，未来考虑合并或移除
    NO_OVERLAP = 2 # 无重叠，在模块化内核前防御性调用
    MK_INTERNAL_OVERLAPPED = 3 # 在 DP/EP 的调度/组合中重叠，由模块化内核调用
    MULTI_STREAM_OVERLAPPED = 4 # 在门控、路由、专家计算的辅助流中重叠

class SharedExperts:
    def __init__(self, layer, moe_config, quant_method, reduce_results, enable_dbo):
        # 初始化共享专家管理类，处理 DBO（动态批处理优化）和 CUDA 流设置
        self._layer = layer # 底层 PyTorch 模块（共享专家 MLP）
        self._moe_config = moe_config # MoE 配置
        self._quant_method = quant_method # 量化方法，必须是 FusedMoEMethodBase 类型
        self._reduce_results = reduce_results # 是否需结果归约
        self._use_dp_chunking = moe_config.moe_parallel_config.use_dp_chunking # DP 分块标志
        self.enable_dbo = enable_dbo # DBO 启用状态
        self._output = [None, None] # 输出缓存列表，支持 DBO 按 ubatch id 索引
        # 设置 CUDA 流以实现重叠执行，可通过环境变量 VLLM_DISABLE_SHARED_EXPERTS_
        STREAM 禁用
        if envs.VLLM_DISABLE_SHARED_EXPERTS_STREAM:
```

```
            self._stream = None
            logger.debug_once("Disabling MoE shared_experts cuda stream", scope="local")
        else:
            self._stream = aux_stream() # 获取辅助流，非 CUDA 类平台返回 None
            if self._stream is not None:
                logger.debug_once("Enabled separate cuda stream for MoE shared_experts", scope=
                "local")

    def _determine_shared_experts_order(self, hidden_states):
        # 基于配置、量化方法和输入张量决定共享专家的执行顺序
        if self._has_external_experts and not self._use_dp_chunking:
            return SharedExpertsOrder.EXTERNAL # 外部执行路径
        if self._quant_method.mk_owns_shared_expert:
            return SharedExpertsOrder.MK_INTERNAL_OVERLAPPED # 模块化内核内部重叠
        should_run_shared_in_aux_stream = (
            current_platform.is_cuda()
            and not self._use_dp_chunking
            and self._stream is not None
            and hidden_states.shape[0] <= envs.VLLM_SHARED_EXPERTS_STREAM_TOKEN_
            THRESHOLD
        )
        if should_run_shared_in_aux_stream:
            return SharedExpertsOrder.MULTI_STREAM_OVERLAPPED # 多流重叠
        else:
            return SharedExpertsOrder.NO_OVERLAP # 无重叠

    def apply(self, shared_experts_input, order):
        # 根据指定顺序执行共享专家计算，处理 DBO 缓存和结果归约
        if order == SharedExpertsOrder.EXTERNAL:
            # 外部执行：直接调用底层模块并可选进行张量模型并行归约
            output = self._layer(shared_experts_input)
            if self._reduce_results and get_tensor_model_parallel_world_size() > 1:
                output = tensor_model_parallel_all_reduce(output)
            return output
        # 其他顺序（如 NO_OVERLAP、MK_INTERNAL_OVERLAPPED、MULTI_STREAM_
        OVERLAPPED）
        # 涉及流同步和重叠逻辑，具体实现省略以保持简洁
        # ...
```

vllm/model_executor/layers/fused_moe/runner/default_moe_runner.py

核心运行器修改，整合 SharedExperts 并重构 forward 逻辑，影响 MoE 执行入口。

```
def forward_dispatch(self, layer, hidden_states, router_logits, shared_experts_input):
    # 统一分发 MoE 前向计算，根据是否启用 DP 分块选择具体实现
    if self.use_dp_chunking:
        return self.forward_impl_chunked(layer, hidden_states, router_logits, shared_experts_
        input)
    else:
        return self.forward_impl(layer, hidden_states, router_logits, shared_experts_input)
```

```
def _apply_shared_experts(self, shared_experts_input, order):
    # 应用共享专家计算，委托给 SharedExperts 实例处理
    if self.shared_experts is not None:
        return self.shared_experts.apply(shared_experts_input, order)
    return None # 无共享专家时返回 None

def forward_impl(self, layer, hidden_states, router_logits, shared_experts_input):
    # 标准 forward 实现，整合共享专家执行和路由逻辑
    if self.shared_experts is not None:
        # 决定共享专家执行顺序并应用
        order = self.shared_experts._determine_shared_experts_order(shared_experts_input)
        shared_output = self._apply_shared_experts(shared_experts_input, order)
        # 结合路由专家输出（简化示意）
        fused_output = self._run_routed_experts(hidden_states, router_logits)
        return shared_output, fused_output
    else:
        return self._run_routed_experts(hidden_states, router_logits)
```

评论区精华

review 中，robertgshaw2-redhat 指出共享专家所有权混乱（'OWNED BY EXTERNAL - OWNED BY RUNNER - OWNED BY MK'），建议未来动态决策；bnellnm 同意在后续 PR 处理。关于执行顺序，讨论了 SharedExpertsOrder.EXTERNAL 的必要性，认为可能因 torch.compile 原因暂时保留。此外，对流同步时机（'syncing the streams here defeats the purpose'）和 API 设计（如 SharedExperts 类命名混淆）有深入交锋，最终决定在后续迭代优化。

- 共享专家所有权设计 (design): 决定在后续 PR 中优化所有权模型，当前实现作为中间状态。
- 执行顺序和流同步 (performance): 保留 EXTERNAL 顺序以待 torch.compile 兼容，流同步逻辑调整。
- API 返回类型变更 (correctness): 未在评论中明确解决，需确保所有调用站点更新或类型准确。

风险与影响

- 风险：技术风险包括：回归风险，因为核心 MoE 执行路径变更，可能影响模型正确性和性能；测试覆盖虽全面，但新逻辑在边缘条件（如 DP 分块、多流重叠）下需进一步验证；兼容性问题，SharedExperts 类的引入可能破坏依赖旧接口的第三方代码；执行顺序逻辑复杂，易在并发场景下引入竞态条件。
- 影响：对用户而言，MoE 模型推理的底层行为保持不变，但性能可能因共享专家重叠优化而提升；对系统，重构提高了代码模块化和可维护性，为未来功能扩展（如更灵活的重叠策略）奠定基础；对团队，开发者需适应新 API，但清晰的职责分离将降低后续开发复杂度。
- 风险标记：核心路径变更，所有权逻辑复杂，测试覆盖需验证，执行顺序边缘条件

关联脉络

- PR #33049 [MoE Refactor] Initial MoERunner refactor: 基础重构，为本 PR 的前置工作，建立了 MoERunner 框架。
- PR #35949 Final PR for MoE refactor: 本 PR 的最终版本，关联性强，可能进一步优化 SharedExperts 集成。