PR #39684 完整报告

vllm-project/vllm

[Compilation] Refactor SiluMul activation+quant Fusion Pass

合并时间：2026-04-23 21:10

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/39684

执行摘要

- 一句话：重构 SiluMul 激活与量化融合 pass，统一注册框架
- 推荐动作：该 PR 值得所有关心 vllm 编译优化和代码架构的开发者精读。
- 展示了如何将现有代码平滑迁移到 VllmPatternReplacement / VllmFusionPatternMatcherPass 框架。
- 设计上通过 pattern 和 replacement 属性分离模式定义，避免了手工调用 register_replacement，提高了可读性。
- 测试中的特殊处理虽然临时，但作者已承诺统一，社区可以跟进。
- 建议学习其拆分思路，未来新 fusion pass 应直接使用该模式。

功能与动机

PR 描述明确说明：'Cleaned up the SiluMul activation+quant fusion passes to align with the existing pattern matcher infrastructure.' 目的是使这些融合 pass 与 vllm 已有的模式匹配器基础设施对齐，统一代码风格，减少重复样板代码。

实现拆解

重构 SiluMul 激活 + 量化融合 Pass

1. 重构 ActivationQuantPattern 基类

- 文件：vllm/compilation/passes/fusion/act_quant_fusion.py
  - 将基类从 ABC 改为继承 VllmPatternReplacement，移除抽象方法 register，取而代之的是 pattern 和 replacement 属性，由子类通过 @property 提供。
  - 这样统一了融合模式的声明方式，与框架内其他模式保持一致。

2. 更新所有具体模式类

- 文件：act_quant_fusion.py（SiluMulFp8StaticQuantPattern, SiluMulNvfp4QuantPattern, SiluMulBlockQuantPattern）
  - 每个子类将原来的 register 方法拆解为 pattern 和 replacement 属性，内部定义嵌套函数 _pattern 和 _replacement。同时利用闭包捕获 self，避免手动传递 pm_pass。
- 例如 SiluMulFp8StaticQuantPattern 中：

3. 重构融合 Pass 类

- 文件：act_quant_fusion.py 中的 ActivationQuantFusionPass 和 rocm_aiter_fusion.py 中的 RocmAiterSiluMulFp8GroupQuantFusionPass
  - 改为继承 VllmFusionPatternMatcherPass，在构造函数中通过 super().__init__(config, pass_name) 设置 pass 名称，然后调用 self.register(pattern_instance) 注册每个模式。
  - 去除了手动创建 PatternMatcherPass、调用 register_replacement 以及 __call__ / uuid 等样板代码，因为父类已经处理。
- 例如 ActivationQuantFusionPass 的新构造函数：

4. 测试配套调整

- 文件：tests/compile/fusions_e2e/conftest.py
  - 由于重构后 act_quant_fusion 的日志匹配方式变化（不再输出每范围的匹配数），在 e2e 测试中添加了特殊分支来处理 act_quant_fusion 断言。
- 文件：tests/compile/passes/test_silu_mul_quant_fusion.py
  - 修复了 TestSiluMulGroupFp8QuantModel 中 w8a8_block_fp8_linear 调用时传递的冗余参数 self.w 和 self.wscale，这些参数已由内部处理，不再需要显式传入。

5. 总结

- 整体改动净减 103 行，代码更加简洁。
- 模式注册统一采用 VllmPatternReplacement 和 VllmFusionPatternMatcherPass，为后续添加新的激活 + 量化融合提供了标准接口。

关键文件：

- vllm/compilation/passes/fusion/act_quant_fusion.py（模块 编译层；类别 source；类型 core-logic；符号 ActivationQuantPattern, SiluMulFp8StaticQuantPattern, SiluMulNvfp4QuantPattern, SiluMulBlockQuantPattern）：核心重构文件，修改了所有 Activation+Quant 融合模式和 pass 类
- vllm/compilation/passes/fusion/rocm_aiter_fusion.py（模块 ROCm 编译；类别 source；类型 core-logic；符号 AiterSiluMulFp8GroupQuantPattern, RocmAiterSiluMulFp8GroupQuantFusionPass）：重构了 ROCm AITER SiluMul+Fp8 组量化融合模式和 pass
- tests/compile/fusions_e2e/conftest.py（模块 e2e 测试；类别 test；类型 test-coverage）：添加 act_quant_fusion 特殊处理以适应新的匹配方式
- tests/compile/passes/test_silu_mul_quant_fusion.py（模块 单元测试；类别 test；类型 bugfix）：修复冗余参数调用

关键符号：ActivationQuantPattern.init, SiluMulFp8StaticQuantPattern.pattern, SiluMulFp8StaticQuantPattern.replacement, ActivationQuantFusionPass.init, AiterSiluMulFp8GroupQuantPattern.pattern, AiterSiluMulFp8GroupQuantPattern.replacement, RocmAiterSiluMulFp8GroupQuantFusionPass.init

关键源码片段

vllm/compilation/passes/fusion/act_quant_fusion.py

核心重构文件，修改了所有 Activation+Quant 融合模式和 pass 类

```
class ActivationQuantPattern(VllmPatternReplacement):
    """
    Base class for Activation+Quant fusions.
    Should not be used directly.

    # 现在继承 VllmPatternReplacement 而非 ABC,
    # 子类通过定义 pattern 和 replacement 属性来声明模式。
    """
    def __init__(self, quant_key: QuantKey) -> None:
        self.quant_key = quant_key
        self.quant_dtype = quant_key.dtype
        # 验证量化操作和融合操作存在
        assert self.quant_key in QUANT_OPS
        self.QUANT_OP = QUANT_OPS[self.quant_key]
        assert self.quant_key in FUSED_OPS
        self.FUSED_OP = FUSED_OPS[self.quant_key]
        self.silu_and_mul_matcher = MatcherSiluAndMul()

    def empty_quant(self, *args: Any, **kwargs: Any) -> torch.Tensor:
        kwargs = {"dtype": self.quant_dtype, "device": "cuda", **kwargs}
        return torch.empty(*args, **kwargs)


class SiluMulFp8StaticQuantPattern(ActivationQuantPattern):
    """
    Fusion for SiluMul+Fp8StaticQuant Pattern.
    # pattern 属性返回匹配函数，replacement 属性返回替换函数。
    # 框架自动调用 get_inputs 生成假张量来追踪模式。
    """
    def __init__(self) -> None:
        super().__init__(kFp8StaticTensorSym)
        self.quant_matcher = MatcherQuantFP8(kFp8StaticTensorSym)

    def get_inputs(self) -> list[torch.Tensor]:
        scale = self.quant_matcher.inputs()[1]
        return [*self.silu_and_mul_matcher.inputs(), scale]

    @property
    def pattern(self):
        def _pattern(input: torch.Tensor, scale: torch.Tensor) -> torch.Tensor:
            # 匹配两个连续操作：silu_and_mul 后接 fp8 量化
            result_silu_mul = self.silu_and_mul_matcher(input)
            result_quant = self.quant_matcher(result_silu_mul, scale)
            return result_quant[0]
        return _pattern
```

```
    @property
    def replacement(self):
        def _replacement(input: torch.Tensor, scale: torch.Tensor) -> torch.Tensor:
            # 用一个融合算子代替
            d = input.shape[-1] // 2
            output_shape = input.shape[:-1] + (d,)
            result = torch.empty(output_shape, device=input.device, dtype=self.quant_dtype)
            at = auto_functionalized(self.FUSED_OP, result=result, input=input, scale=scale)
            return at[1]
        return _replacement
```

vllm/compilation/passes/fusion/rocm_aiter_fusion.py

重构了 ROCm AITER SiluMul+Fp8 组量化融合模式和 pass

```
class AiterSiluMulFp8GroupQuantPattern(VllmPatternReplacement):
    """
    This pattern fuses aiter silu_and_mul & group fp8 quant custom
    ops into an aiter silu_and_mul_group_fp8_quant op.

    # 不再继承 ActivationQuantPattern，直接继承 VllmPatternReplacement
    # 使用 property pattern 和 replacement 声明模式。
    """
    FUSED_SILU_MUL_QUANT_OP = rocm_aiter_ops.get_act_mul_fused_fp8_group_quant_op()

    def __init__(self) -> None:
        self.silu_and_mul_matcher = MatcherSiluAndMul()
        self.quant_matcher = MatcherQuantFP8(quant_key=kFp8Dynamic128Sym, match_rocm_
        aiter=True)

    def get_inputs(self) -> list[torch.Tensor]:
        return [self.silu_and_mul_matcher.inputs()[0]]

    @property
    def pattern(self):
        def _pattern(input: torch.Tensor) -> tuple[torch.Tensor, torch.Tensor]:
            at1 = self.silu_and_mul_matcher(input)
            at2 = self.quant_matcher(at1)
            return at2[0], at2[1]
        return _pattern

    @property
    def replacement(self):
        def _replacement(input: torch.Tensor) -> tuple[torch.Tensor, torch.Tensor]:
            at = self.FUSED_SILU_MUL_QUANT_OP(x=input, group_size=128)
            return at[0], at[1]
        return _replacement


class RocmAiterSiluMulFp8GroupQuantFusionPass(VllmFusionPatternMatcherPass):
```

```
    """
    This pass fuses a pre-defined set of custom ops into fused ops.

    # 继承 VllmFusionPatternMatcherPass，构造函数极大简化。
    # 不再需要自己创建 PatternMatcherPass、实现 __call__ 和 uuid。
    """
    def __init__(self, config: VllmConfig) -> None:
        super().__init__(config, "rocm_aiter_silu_mul_fp8_group_quant_fusion_pass")
        self.register(AiterSiluMulFp8GroupQuantPattern())
        self.dump_patterns(config, self.pm_pass)
```

评论区精华

Review 中 ProExpertProg 在 [conftest.py 行 253] 评论：

> "We should unify these somehow so we don't have to special case every time" 作者 BadrBasowid 回应： "yeah ill fix that in the next PR"

这指出了测试中的临时特殊处理应当被统一解决，已计划后续 PR 完善。

- 测试特殊处理应统一 (design): 临时方案已接受，后续 PR 统一。

风险与影响

- 风险：本 PR 为纯重构，不改变融合的数学行为。风险主要来源于以下方面：
  - 编译 pass 注册变更：新模式继承自 VllmFusionPatternMatcherPass，如果父类的实现存在未发现的 bug，可能导致模式匹配失败或误匹配。不过已有 80 个单元测试通过，覆盖了主要的融合场景。
  - 测试特殊处理：conftest.py 中对 act_quant_fusion 添加了特殊断言路径，后续若统一方案未及时落实，可能导致其他测试维护负担。
  - ROCm 平台：rocm_aiter_fusion.py 同样重构，如果 ROCm 特有的 AITER 算子行为发生变化，可能影响 ROCm 用户的融合效果。
  - 量化模式精度：重构未修改算子本身，但若传入参数或张量形状推断有误，可能会导致精度回退。建议合并前运行完整量化模型 *e2e* 测试。
  - 影响：
  - 用户影响：无直接功能变化，用户无需改变使用方式。
  - 系统影响：编译图融合的代码路径变短，可能轻微提升编译稳定性。
  - 开发团队影响：统一了融合 pass 的编写范式，降低新融合的开发门槛。后续如需增加新的 activation+quant 组合，只需编写继承 VllmPatternReplacement 的类并注册到 pass 即可。
  - 测试影响：*e2e* 测试中对 act_quant_fusion 进行了适配，保证了覆盖率；待后续 PR 统一特殊处理。
  - 兼容性：完全向后兼容，无破坏性变更。
  - 风险标记：编译核心路径变更，测试适配待统一，ROCm 平台影响

关联脉络

- 暂无明显关联 PR