

PR #52263 完整报告

vllm-project/vllm

[ROCm][Quantization] Support AMD Quark per-block FP8 for fused MoE layers

合并时间: 2026-09-08 18:30

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/52263>

执行摘要

- 一句话: 为 Quark 量化后端添加 per-block FP8 MoE 支持, 使 MoE 模型加载和推理可使用 128x128 块量化尺度。
- 推荐动作: 这是一个高质量且重要的功能 PR, 值得精读。它不仅解决了具体的功能缺失, 还在审查过程中展现了对代码库跨后端兼容性的深入思考和良好的工程实践 (如提取公共函数)。重点关注 `QuarkW8A8Fp8MoEMethod` 的 `__init__` 和 `create_weights` 方法中对 per-block 路径的处理, 以及 `validate_fp8_block_shape_moe` 如何被三个不同的 MoE 后端复用。对于从事量化、MoE 或 AMD 生态开发的工程师, 此 PR 的设计和实现模式具有较高的借鉴价值。

功能与动机

PR #47972 已经为 Quark 的线性层添加了 per-block FP8 支持, 但 `QuarkW8A8Fp8MoEMethod` 仍然拒绝 `per_block` 和 `per_group` 的量化方案。这导致当使用 Quark 量化路由专家的 MoE 模型检查点时, 加载过程会失败, 报错为 `ValueError: For FP8 Fused MoE layers, only per-tensor and per-channel scales for weights and activations are supported. Found per_block, per_group`。因此, 需要扩展该方法以支持 per-block 量化, 使 MoE 模型的部署成为可能。

实现拆解

1. 扩展 `QuarkW8A8Fp8MoEMethod` 支持键与分派逻辑: 在 `vllm/model_executor/layers/quantization/quark/quark_moe.py` 中, 向 `QuarkW8A8Fp8MoEMethod` 类添加了 `kFp8Static128BlockSym` 和 `kFp8Dynamic128Sym` 到其支持的量化键列表。修改了 `__init__` 方法, 将权重和激活量化方案从二元判断扩展为三元 (`per_tensor/per_channel/per_block`), 并从 `weight_quant_key.scale.group_shape` 中提取块大小到 `self.weight_block_size`。
2. 分配并标记 per-block 权重尺度参数: 在 `create_weights` 方法中, 添加了 `per_block` 分支, 计算并分配形状为 `(num_experts, (intermediate_size_per_partition + block_n - 1) // block_n, (hidden_size + block_k - 1) // block_k)` 的 `w13_weight_scale` 和 `w2_weight_scale` 参数。关键地, 将尺度的 `quant_method` 属性设置为 `FusedMoeWeightScaleSupported.BLOCK.value`, 以确保 `RoutedExperts.weight_loader` 按块网格而非按行进行分片。

3. 提取公共块对齐验证函数：在 `vllm/model_executor/layers/quantization/utils/fp8_utils.py` 中新增了 `validate_fp8_block_shape_moe` 函数，封装了检查 `intermediate_size_per_partition` 是否可被 `block_n` 整除，以及在 `TP>1` 时是否可被 `block_k` 整除的逻辑。然后将 `vllm/model_executor/layers/quantization/fp8.py` (`Fp8MoEMethod`) 和 `vllm/model_executor/layers/quantization/compressed_tensors/compressed_tensors_moe/compressed_tensors_moe_w8a8_fp8.py` (`CompressedTensorsW8A8Fp8MoEMethod`) 中原有的重复验证代码替换为对此函数的调用。
4. 完善测试覆盖：在 `tests/quantization/test_quark.py` 中新增了单元测试和端到端测试。单元测试覆盖了方法初始化、权重形状创建和对齐验证拒绝场景。端到端测试加载公开的微调模型 `Adamji/tiny-qwen3-moe-fp8-per-block`，验证了 `QuarkW8A8Fp8MoEMethod` 被正确分派、权重尺度形状符合预期，并执行了前向传播。
5. 补充配置与集成：在测试的 `QTENSOR_CONFIGS` 中添加了 `fp8_w8a8_dynamic_block_fp32_moe` 条目，将新增的 MoE 方法纳入现有的共享分派与实例化测试矩阵。

关键文件：

- `vllm/model_executor/layers/quantization/quark/quark_moe.py` (模块 量化后端；类别 source；类型 core-logic；符号 `QuarkW8A8Fp8MoEMethod`)：核心实现文件，`QuarkW8A8Fp8MoEMethod` 类被扩展以支持 per-block 量化，包含关键的 `__init__`、`create_weights` 和 `get_fused_moe_quant_config` 方法变更。
- `vllm/model_executor/layers/quantization/utils/fp8_utils.py` (模块 量化工具；类别 source；类型 utility；符号 `validate_fp8_block_shape_moe`)：新增了 `validate_fp8_block_shape_moe` 公共函数，消除了三个 MoE 量化后端中的重复验证代码。
- `tests/quantization/test_quark.py` (模块 量化测试；类别 test；类型 test-coverage；符号 `_make_per_block_fp8_moe_method`, `test_quark_w8a8_fp8_moe_per_block_requires_dynamic_group_input`, `test_quark_w8a8_fp8_moe_per_block_weight_shapes`, `test_quark_w8a8_fp8_moe_per_block_rejects_misaligned_partition`)：测试文件，提供了对 per-block FP8 MoE 方法的新单元测试和端到端模型测试，确保功能正确性和集成性。

关键符号：`QuarkW8A8Fp8MoEMethod.init`, `QuarkW8A8Fp8MoEMethod.create_weights`, `QuarkW8A8Fp8MoEMethod.get_fused_moe_quant_config`, `validate_fp8_block_shape_moe`

关键源码片段

`vllm/model_executor/layers/quantization/quark/quark_moe.py`

核心实现文件，`QuarkW8A8Fp8MoEMethod` 类被扩展以支持 per-block 量化，包含关键的 `__init__`、`create_weights` 和 `get_fused_moe_quant_config` 方法变更。

```
class QuarkW8A8Fp8MoEMethod(QuarkMoEMethod):
    # 支持的激活量化键增加了 per-group 的动态键
    supported_activation_quant_keys = [
        kFp8StaticTensorSym,
        kFp8DynamicTensorSym,
```

```

        kFp8DynamicTokenSym,
        kFp8Dynamic128Sym, # 新增: 支持 per-group 动态激活量化
    ]
# 支持的权重量化键增加了 per-block 静态键
supported_weight_quant_keys = [
    kFp8StaticChannelSym,
    kFp8StaticTensorSym,
    kFp8Static128BlockSym, # 新增: 支持 128x128 块静态权重量化
]

def __init__(
    self,
    moe: FusedMoEConfig,
    weight_quant_key: QuantKey,
    activation_quant_key: QuantKey | None,
):
    super().__init__(moe, weight_quant_key, activation_quant_key)
    self.weight_dtype = "fp8"
    # 根据权重和激活的量化键, 确定量化方案字符串
    if weight_quant_key == kFp8StaticChannelSym:
        self.weight_qscheme = "per_channel"
    elif weight_quant_key == kFp8Static128BlockSym:
        self.weight_qscheme = "per_block" # 新增
    else:
        self.weight_qscheme = "per_tensor"
    if activation_quant_key == kFp8DynamicTokenSym:
        self.input_qscheme = "per_channel"
    elif activation_quant_key == kFp8Dynamic128Sym:
        self.input_qscheme = "per_group" # 新增
    else:
        self.input_qscheme = "per_tensor"
    # ... 其余初始化逻辑 ...
    # 检查组合是否合法 (per-tensor, per-channel, 或 per-block)
    per_block = (
        self.weight_qscheme == "per_block" and self.input_qscheme == "per_group"
    )
    if not (per_tensor or per_channel or per_block):
        raise ValueError(
            "For FP8 Fused MoE layers, only per-tensor, per-channel and "
            "per-block scales for weights and activations are supported. "
            f"Found {self.weight_qscheme}, {self.input_qscheme}"
        )
    # 从权重量化键的 group_shape 提取块大小
    self.weight_block_size: list[int] | None = (
        list(weight_quant_key.scale.group_shape) if per_block else None
    )
    self.block_quant = self.weight_block_size is not None
    # ... 设置 act_quant_group_shape 等 ...

```

vllm/model_executor/layers/quantization/utils/fp8_utils.py

新增了 `validate_fp8_block_shape_moe` 公共函数，消除了三个 MoE 量化后端中的重复验证代码。

```
def validate_fp8_block_shape_moe(
    intermediate_size_per_partition: int,
    block_size: list[int],
) -> None:
    """Validate fused MoE block quantization shapes for tensor parallelism.

    确保 gate/up 权重的输出维度 (intermediate_size_per_partition) 可被 block_n 整除,
    并且在 TP>1 时, down 权重的输入维度也可被 block_k 整除。
    """
    from vllm.distributed import get_tensor_model_parallel_world_size

    tp_size = get_tensor_model_parallel_world_size()
    block_n, block_k = block_size[0], block_size[1]

    # 检查 gate/up 权重维度是否与块宽度对齐 (列并行所需)
    if intermediate_size_per_partition % block_n != 0:
        raise ValueError(
            f"The output_size of gate's and up's weight = "
            f"{intermediate_size_per_partition} is not divisible by "
            f"weight quantization block_n = {block_n}."
        )
    # 在 TP>1 时, 检查 down 权重维度是否与块深度对齐 (行并行所需)
    if tp_size > 1 and intermediate_size_per_partition % block_k != 0:
        raise ValueError(
            f"The input_size of down's weight = "
            f"{intermediate_size_per_partition} is not divisible by "
            f"weight quantization block_k = {block_k}."
        )
```

tests/quantization/test_quark.py

测试文件，提供了对 per-block FP8 MoE 方法的新单元测试和端到端模型测试，确保功能正确性和集成性。

```
# 辅助函数，用于创建 per-block FP8 MoE 方法实例进行单元测试
def _make_per_block_fp8_moe_method(
    activation_quant_key: QuantKey = kFp8Dynamic128Sym,
) -> QuarkW8A8Fp8MoEMethod:
    return QuarkW8A8Fp8MoEMethod(
        _make_test_moe_config(),
        kFp8Static128BlockSym, # 使用 per-block 权重量化键
        activation_quant_key,
    )

# 测试 per-block 方法要求动态组输入
def test_quark_w8a8_fp8_moe_per_block_requires_dynamic_group_input():
```

```

with (
    patch(
        f"{QUARK_MOE_MODULE}.select_fp8_moe_backend",
        return_value=(Mock(), Mock()),
    ),
    patch(f"{QUARK_MOE_MODULE}.get_current_vllm_config"),
    pytest.raises(ValueError, match="per-block scales"),
):
    # 传入静态张量激活键，应触发错误
    _make_per_block_fp8_moe_method(kFp8StaticTensorSym)

# 测试端到端加载 per-block MoE 模型
@pytest.mark.parametrize("tp", [1])
def test_quark_fp8_w8a8_per_block_moe(vllm_runner, tp):
    """Test per-block (128x128) FP8 MoE quantization with a tiny Qwen3 MoE model."""
    model_path = "Adamji/tiny-qwen3-moe-fp8-per-block"
    with vllm_runner(
        model_path,
        enforce_eager=True,
        tensor_parallel_size=tp,
        gpu_memory_utilization=0.1,
    ) as llm:
        def check_model(model):
            experts = model.model.layers[0].mlp.experts
            method = experts._quant_method
            # 验证分派到正确的 MoE 方法
            assert isinstance(method, QuarkW8A8Fp8MoEMethod)
            # 验证方法识别为 per-block 量化
            assert method.weight_qscheme == "per_block"
            # ... 验证权重尺度形状 ...
        llm.apply_model(check_model)
        # 执行一次生成以验证前向传播
        output = llm.generate_greedy("Hello", max_tokens=4)
        assert output

```

评论区精华

- 权重尺度命名争议 (fxmarty-amd vs jimmy-adams): fxmarty-amd 质疑将 per-block 尺度注册为 `w13_weight_scale_inv / w2_weight_scale_inv` 的做法，指出下游的 Humming 后端和共享转换器期望旧名 `w13_weight_scale / w2_weight_scale`。jimmy-adams 最终澄清，最终实现（与最初提交不同）统一注册为 `w13_weight_scale / w2_weight_scale`，符合 vLLM FP8 MoE oracle 和 Humming 的预期，且与 Quark 导出的名称一致，因此解决了 Humming 兼容性问题。
- 代码重复与重构建议 (fxmarty-amd vs jimmy-adams): fxmarty-amd 指出三个 MoE 方法中存在重复的块对齐检查代码，并建议提取为静态方法。jimmy-adams 同意重构，并选择将其提取为公共函数 `validate_fp8_block_shape_moe` 放在 `fp8_utils.py` 中，理由是这与其线性层对应的 `validate_fp8_block_shape` 放在同一位置，且避免了引入不必要的模块间依赖。

- API 设计与内部化 (fxmarty-amd vs jimmy-adams): 在讨论将 `get_tensor_model_parallel_world_size()` 调用移入 `validate_fp8_block_shape_moe` 内部时, fxmarty-amd 建议此举, jimmy-adams 随后实现, 简化了调用者的接口。
- PR 描述与测试模型要求 (fxmarty-amd): fxmarty-amd 最初要求缩短 PR 描述并提供实际使用该功能的模型。jimmy-adams 后续更新了描述, 并提供了公开的测试模型链接及详尽的测试结果 (接受率、准确率), 满足了审查要求。
 - 权重尺度参数命名与后端兼容性 (design): 权重尺度参数统一命名为 `w13_weight_scale / w2_weight_scale`, 与 Quark 导出名称和下游组件 (FP8 MoE oracle, Humming) 的预期一致, 解决了兼容性问题。
 - 块对齐验证代码的复用与放置位置 (design): 验证逻辑被提取为 `validate_fp8_block_shape_moe` 函数并放置在 `fp8_utils.py` 中。三个 MoE 后端 (FP8, Compressed-Tensors, Quark) 均调用此函数, 消除了代码重复。
 - 公共函数的 API 设计: 是否内置 TP 大小查询 (design): `validate_fp8_block_shape_moe` 函数现在内部查询 TP 大小, 其签名简化为仅接受 `intermediate_size_per_partition` 和 `block_size`。

风险与影响

- 风险:
 1. 后端兼容性风险 (已缓解): 最初提交将 per-block 尺度命名为 `*_weight_scale_inv`, 可能与 Humming 后端和其他共享路径不兼容。经过审查讨论, 已修正为 `*_weight_scale`, 确保了兼容性。当前实现应能安全地与 TRITON FP8 MoE 后端和 Humming 后端协同工作。
 2. 块对齐与精度风险 (已缓解): 使用 per-block 量化要求维度严格对齐。PR 通过 `validate_fp8_block_shape_moe` 函数在权重创建时进行强制验证, 拒绝不兼容的模型配置 (如 `intermediate_size_per_partition` 不可被 128 整除), 避免了运行时错误或静默精度下降。
 3. 性能回归风险 (低): 变更主要集中在量化方法初始化和权重加载路径, 未修改核心计算内核。per-block 量化本身是性能优化, 而非开销。端到端测试中的高草稿接受率 (94.2%) 间接证明了量化尺度被正确加载和应用。
 4. 测试覆盖完整性: PR 提供了针对核心逻辑的单元测试 (形状、对齐验证) 和使用真实模型的端到端测试, 覆盖了主要功能路径。测试包括了对错误情况的验证 (如拒绝静态激活、不兼容的划分尺寸)。
- 影响:
 - 用户影响: 使使用 AMD Quark 工具链将 MoE 模型 (如 Qwen3.5-35B-A3B) 量化为 per-block FP8 格式后, 能够在 vLLM (ROCm) 上部署成为可能。这扩展了 vLLM 在 AMD 硬件上的模型支持范围和量化选项。
 - 系统影响: 增强了 Quark 量化后端的功能完整性, 使其与 FP8 和 Compressed-Tensors 后端在 per-block MoE 支持上对齐。公共验证函数的提取提升了量化子系统的代码质量和可维护性。

- 团队影响：为量化后端的开发者提供了清晰的 per-block MoE 实现范例和测试模板。讨论中关于参数命名和代码重复的讨论，有助于未来量化相关开发中更好地考虑跨后端兼容性和代码复用。
- 风险标记：后端兼容性（已通过审查修复），跨量化后端代码重复（已通过提取公共函数解决）

关联脉络

- PR #47972 [Quantization][Quark] Support per-block FP8 for Quark linear layers: 本 PR 的直接前置工作。PR #47972 为 Quark 的线性层（非 MoE）添加了 per-block FP8 支持，引入了 QuarkW8A8Fp8PerBlock 类和 kFp8Static128BlockSym 键。本 PR 将该能力扩展到了 MoE 路由专家层，使得整个模型（线性层 + MoE 层）都能使用 per-block 量化。
- PR #53319 [Kernel] Add NVFP4 support to the torch linear backend: 作为近期量化内核增强的关联 PR。虽然技术路径不同（NVFP4 vs FP8 per-block），但两者都代表了为特定硬件和量化方案优化线性 / MoE 层性能的持续努力，反映了 vLLM 在量化内核方面的活跃演进。