PR #49096 完整报告

vllm-project/vllm

Fix Humming non-gated MoE

合并时间：2026-07-28 06:56

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/49096

执行摘要

- 一句话：修复 Humming MoE 非门控专家形状假设
- 推荐动作：本 PR 值得精读，因为它展示了如何在已有的门控假设后端中安全地引入非门控支持，通过 is_gated 属性做条件分支，并配套形状契约测试确保正确性。对于 MoE 量化后端的开发者具有参考价值。

功能与动机

支持 NemotronH 等使用非门控 squared-ReLU 专家的模型在 Humming MoE 后端上运行。此前 Humming 假设每个 w13 包含两个门控投影，对非门控激活产生错误的权重和缓冲区形状。

实现拆解

1. 在 humming_utils.py 中根据激活类型调整 w13 堆叠逻辑：将 _convert_sublayer_to_humming 中的无条件堆叠拆分为 w13 且 is_gated 时才拆成两半。同时修正 convert_to_humming_moe_kernel_format 中的 sublayer_configs，使 w13 的 shape_n 根据 is_gated 取 intermediate_size 或 intermediate_size*2，w2 的 shape_k 固定为 intermediate_size 而不是原先非门控时的翻转。
2. 在 fused_humming_moe.py 中统一问题大小计算：将 moe_problem_size 返回的中间维度从 meta1.shape_n // 2 改为 self.layer.intermediate_size_per_partition，因为该值表示逻辑中间宽度，对门控和非门控一致。get_buffer_metas 中引入 gate_up_size 变量，门控时为 N*2，非门控时为 N*1；down_input_size 固定为 N，不再依赖是否门控，确保缓冲区形状正确。
3. 新增形状契约测试：在 tests/kernels/moe/test_moe.py 中添加 test_humming_gated_non_gated_shape_contract，使用参数化的 MoEActivation.SILU（门控）和 MoEActivation.RELU2_NO_MUL（非门控），构造虚拟 MoE 层并验证转换后的 meta 形状、缓冲区形状、moe_problem_size 返回值是否符合预期。

关键文件：

- vllm/model_executor/layers/fused_moe/experts/fused_humming_moe.py（模块 MoE 专家层；类别 source；类型 core-logic；符号 moe_problem_size, get_buffer_metas）：核心计算路径，修改了问题大小计算和缓冲区 meta 定义，直接决定非门控激活的正确性
- tests/kernels/moe/test_moe.py（模块 MoE 测试；类别 test；类型 test-coverage；符号 test_humming_gated_non_gated_shape_contract）：新增形状契约测试，确保门控与非门

控场景的 meta 形状、缓冲区形状、问题大小均正确

- vllm/model_executor/layers/quantization/utils/humming_utils.py（模块 量化工具；类别 source；类型 data-contract；符号 _convert_sublayer_to_humming, convert_to_humming_moe_kernel_format）：权重转换格式逻辑，修改 w13 堆叠条件和非门控时 sublayer_configs 的 shape_n 和 shape_k

关键符号：moe_problem_size, get_buffer_metas, _convert_sublayer_to_humming, convert_to_humming_moe_kernel_format, test_humming_gated_non_gated_shape_contract

关键源码片段

vllm/model_executor/layers/fused_moe/experts/fused_humming_moe.py

核心计算路径，修改了问题大小计算和缓冲区 meta 定义，直接决定非门控激活的正确性

```
    def moe_problem_size(
        self,
        a1: torch.Tensor,
        w1: torch.Tensor,
        w2: torch.Tensor,
        topk_ids: torch.Tensor,
    ) -> tuple[int, int, int, int, int]:
        from vllm.utils.humming import HummingLayerMeta

        meta1: HummingLayerMeta = self.layer.humming_metas["w13"]
        meta2: HummingLayerMeta = self.layer.humming_metas["w2"]

        assert meta1.num_experts == meta2.num_experts

        num_experts = meta1.num_experts
        top_k = topk_ids.size(1)
        assert w1.size(0) == num_experts
        assert w2.size(0) == num_experts

        if not self.is_batched():
            num_tokens = a1.size(0)
            assert topk_ids.size(0) == num_tokens
        else:
            assert a1.dim() == 3
            assert a1.size(0) == num_experts
            num_tokens = a1.size(1)

        return (
            meta1.num_experts,
            num_tokens,
            # 逻辑中间宽度：之前是 meta1.shape_n // 2，但 shape_n 对于非门控可能已是
            intermediate_size
            self.layer.intermediate_size_per_partition,
```

```
            meta1.shape_k,
            top_k,
        )

    def get_buffer_metas(self, M: int, topk: int, activation: MoEActivation):
        from vllm.utils.humming import GemmType as HummingGemmType
        from vllm.utils.humming import dtypes

        num_experts = self.num_experts
        N = self.layer.intermediate_size_per_partition
        K = self.layer.hidden_size

        # ... 其他不变

        gate_up_size = N * (2 if activation.is_gated else 1) # 门控时为 2N，非门控时为 N
        down_input_size = N # 固定为 N，之前非门控时错误地设为 2N

        buffer_metas = {
            "quanted_gate_up_input": {
                "shape": (input_shape_m, K),
                "dtype": torch_dtype_map[a_dtype],
            },
            "gate_up_output": {
                "shape": (real_shape_m, gate_up_size), # 之前固定 N*2
                "dtype": torch_dtype_map[c_dtype],
            },
            "activation_output": {
                "shape": (real_shape_m, down_input_size), # 之前依赖 is_gated
                "dtype": torch_dtype_map[c_dtype],
            },
            # ... 其他缓冲区
        }
        return buffer_metas
```

tests/kernels/moe/test_moe.py

新增形状契约测试，确保门控与非门控场景的 meta 形状、缓冲区形状、问题大小均正确

```
@pytest.mark.parametrize(
    "activation",
    [
        MoEActivation.SILU,
        MoEActivation.RELU2_NO_MUL,
    ],
    ids=["gated", "non_gated"],
)
def test_humming_gated_non_gated_shape_contract(activation: MoEActivation):
    pytest.importorskip("humming")
    from vllm.model_executor.layers.fused_moe.experts.fused_humming_moe import (
        HummingIndexedExperts,
```

```
)
from vllm.model_executor.layers.quantization.utils import humming_utils
from vllm.utils import humming

top_k, num_experts = 6, 12
hidden_size, intermediate_size = 2688, 1856
# 根据激活类型计算 gate_up_size 和 num_w13_stacks
gate_up_size = intermediate_size * 2 if activation.is_gated else intermediate_size
num_w13_stacks = 2 if activation.is_gated else 1

# 创建虚拟 MoE 配置和层
moe_config = make_dummy_moe_config(
    num_experts=num_experts,
    experts_per_token=top_k,
    hidden_dim=hidden_size,
    intermediate_size=intermediate_size,
    activation=activation,
)
layer = torch.nn.Module()
layer.moe_config = moe_config
layer.params_dtype = torch.bfloat16

# 注册权重参数
weight_schema = humming.ModeloptNvfp4WeightSchema()
for sublayer_name, shape_n, shape_k, stack_size in (
    ("w13", gate_up_size, hidden_size, num_w13_stacks),
    ("w2", hidden_size, intermediate_size, 1),
):
    tensor_attrs = weight_schema.get_tensors_attrs(
        shape_n=shape_n,
        shape_k=shape_k,
        param_dtype=layer.params_dtype,
        num_experts=num_experts,
        stack_size=stack_size,
    )
    for tensor_name, attrs in tensor_attrs.items():
        layer.register_parameter(
            f"{sublayer_name}_{tensor_name}",
            Parameter(torch.ones(attrs["shape"], dtype=attrs["dtype"], device="cuda"), requires_
            grad=False),
        )

# 转换为 Humming 格式
humming_utils.convert_to_humming_moe_kernel_format(
    layer,
    weight_schema=weight_schema,
    input_schema=humming.HummingInputSchema(a_dtype=humming.dtypes.bfloat16),
)
```

```
    # 验证 meta 形状
    w13_meta, w2_meta = (layer.humming_metas[name] for name in ("w13", "w2"))
    assert w13_meta.shape_n - w13_meta.pad_shape_n == gate_up_size
    assert w2_meta.shape_k - w2_meta.pad_shape_k == intermediate_size

    # 创建专家模块并验证缓冲区形状
    layer.local_num_experts = layer.global_num_experts = num_experts
    layer.hidden_size = hidden_size
    layer.intermediate_size_per_partition = intermediate_size
    quant_config = humming_utils.get_humming_moe_quant_config(layer)
    experts = HummingIndexedExperts(layer, moe_config, quant_config)

    buffer_metas, _ = experts.get_buffer_metas(
        M=1, topk=top_k, activation=moe_config.activation,
    )
    assert buffer_metas["gate_up_output"]["shape"][-1] == gate_up_size
    assert buffer_metas["activation_output"]["shape"][-1] == intermediate_size
    assert experts.moe_problem_size(
        a1=torch.empty(1, hidden_size),
        w1=torch.empty(num_experts, 1),
        w2=torch.empty(num_experts, 1),
        topk_ids=torch.empty(1, top_k, dtype=torch.long),
    ) == (num_experts, 1, intermediate_size, hidden_size, top_k)
```

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

权重转换格式逻辑，修改 w13 堆叠条件和非门控时 sublayer_configs 的 shape_n 和 shape_k

```
def _convert_sublayer_to_humming(
    layer: "RoutedExperts",
    sublayer_name: str,
    shape_n: int,
    shape_k: int,
    weight_schema: Any,
    input_schema: Any,
    num_experts: int,
    param_dtype: torch.dtype,
) -> tuple[Any, Any]:
    # ... 前置代码

    shape_k_stacks = [shape_k]
    shape_n_stacks = [shape_n]
    # 仅当是 w13 且激活为门控时才拆分为两个堆叠
    if sublayer_name == "w13" and layer.moe_config.activation.is_gated:
        shape_n_stacks = [shape_n // 2] * 2

    # 后续转换代码不变

def convert_to_humming_moe_kernel_format(
```

```
    layer: "RoutedExperts",
    weight_schema: Any = None,
    input_schema: Any = None,
    quant_config: Any = None,
    sublayer_configs: Optional[dict] = None,
    force_weight_schema: Any = None,
) -> None:
    # ... 前置代码

    # Build sublayer configs from layer properties if not provided
    if sublayer_configs is None:
        is_gated = layer.moe_config.activation.is_gated
        intermediate_size = layer.moe_config.intermediate_size_per_partition
        sublayer_configs = {
            "w13": {
                "shape_n": intermediate_size * (2 if is_gated else 1), # 之前固定 *2
                "shape_k": layer.moe_config.hidden_dim,
            },
            "w2": {
                "shape_n": layer.moe_config.hidden_dim,
                "shape_k": intermediate_size, # 之前非门控时错误地设为 intermediate_size*2
            },
        }

    # 后续处理代码不变
```

评论区精华

- amirkl94在 fused_humming_moe.py 中询问为什么 moe_problem_size 不能继续使用 meta1.shape_n // 2。作者 netanel-haber回应 Humming 后端实际不使用该返回值，但增加了注释说明其表示逻辑中间宽度，对门控和非门控均适用。
- amirkl94在 humming_utils.py 中评论 "Redundant"，指向新增的 intermediate_size 临时变量。作者未进一步回复，但最终合并，推测已 inline 解决。
- moe_problem_size 返回值中的中间宽度 (correctness): 保留使用 intermediate_size_per_partition，添加注释解释
- humming_utils.py 中新增 intermediate_size 变量 (style): 最终合并，可能认为可读性提升值得

风险与影响

- 风险：
  - 形状假设变更风险：moe_problem_size 和缓冲区定义直接决定 MoE 前向计算的正确性。本 PR 同时修改了三个文件中的形状逻辑，若非门控情况下 is_gated 判断不正确，可能导致静默数值错误。但新增的测试覆盖了两种激活类型，降低了风险。
  - 量化后端兼容性风险：修改了 humming_utils.py 中的转换逻辑，影响将模型权重转换为 Humming 格式的路径。对于已支持的门控模型，需要确保回归测试通过（PR 中已包含

门控测试）。
  - 性能影响风险：修复后非门控模型可正确使用 Humming 后端，从性能数据看有显著提升，但未提供回归测试确保门控模型性能不退化。不过改动集中在形状推导，对计算核心不构成直接性能风险。
- 影响：
  - 用户影响：使用 NemotronH 等非门控 MoE 模型的用户现在可以启用 --moe-backend humming 并获得比 Marlin 后端更高的吞吐和更低的首 Token 延迟。
  - 系统影响：仅影响 Humming MoE 后端，不影响其他 MoE 后端（如 Marlin、Triton）。新增测试增加了 CI 时间，但仅在使用 Humming 单元时运行。
  - 团队影响：为后续支持更多非门控激活函数（如 SquaredReLU）奠定了代码结构基础。
  - 风险标记：核心路径变更，形状计算变更

关联脉络

- PR #46765 [ROCm][Quantization][5/N] Refactor quark_moe w8a8-int8 w/ oracle: 都涉及 MoE 量化后端，同属 MoE 核心改进路径，修改了 fused_moe 和量化工具层