PR #44763 完整报告

vllm-project/vllm

Add weights padding for fp8 per-block online quantization

合并时间：2026-06-23 23:08

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/44763

执行摘要

- 一句话：FP8 per-block 在线量化权重补齐
- 推荐动作：该 PR 值得合并，修复了明确的 bug。建议后续添加单元测试（如模拟不可整除的维度，验证 _zero_padding 和 maybe_roundup_sizes 的正确性），以及对所有 online MoE 量化方法统一考虑维度对齐问题。

功能与动机

FP8 per-block 在线量化要求隐藏维度能被 block size（128）整除，但模型如 Qwen3-30B-A3B 在 TP=4 时 intermediate size 为 192，不满足整除条件，导致 silu_and_mul_per_block_quant 抛出 RuntimeError: hidden_size must be divisible by group_size。PR 描述中包含了完整的错误堆栈。

实现拆解

1. 新增导入：在 vllm/model_executor/layers/quantization/online/fp8.py 中添加 from vllm.utils.math_utils import round_up。
2. 重写 maybe_roundup_sizes：在 Fp8PerBlockOnlineMoEMethod 中重写父类方法，先调用父类逻辑，再将 hidden_size 和 intermediate_size_per_partition 分别向上取整到 weight_block_size[0]（即 128），使得后续创建的张量尺寸满足 block 整除要求。
3. 新增 _zero_padding：在 process_weights_after_loading 开始时调用。该方法将权重张量中超出原始 unpadded 尺寸的区域置零，确保新增的填充部分不会影响推理结果。具体处理 w13_weight（分为上下两半，分别对应 gate 和 up 投影）、w2_weight、以及可选的 w13_bias 和 w2_bias。
4. 调用集成：在 process_weights_after_loading 中，加载权重后先执行 _zero_padding，再进行 FP8 量化等后续处理。
5. 测试与验证：PR 提供了基于 Qwen3-30B-A3B 的手动测试命令和 gsm8k lm_eval 结果（TP=2 无填充 vs TP=4 有填充），两个配置下 accuracy 接近，表明填充不影响模型精度。

关键文件：

- vllm/model_executor/layers/quantization/online/fp8.py（模块 量化层；类别 source；类型 data-contract；符号 maybe_roundup_sizes, _zero_padding）：唯一修改的文件；在 Fp8PerBlockOnlineMoEMethod 中添加 maybe_roundup_sizes 和 _zero_padding 方法，并修改 process_weights_after_loading 调用链。

关键符号：maybe_roundup_sizes, _zero_padding

关键源码片段

vllm/model_executor/layers/quantization/online/fp8.py

唯一修改的文件；在 Fp8PerBlockOnlineMoEMethod 中添加 maybe_roundup_sizes 和 _zero_padding 方法，并修改 process_weights_after_loading 调用链。

```
# vllm/model_executor/layers/quantization/online/fp8.py
# 新增导入 round_up
from vllm.utils.math_utils import round_up


class Fp8PerBlockOnlineMoEMethod(_Fp8OnlineMoEBase):
    # ... 省略构造函数 ...

    def maybe_roundup_sizes(
        self,
        hidden_size: int,
        intermediate_size_per_partition: int,
        act_dtype: torch.dtype,
        moe_parallel_config,
    ) -> tuple[int, int]:
        # 先调用父类逻辑
        hidden_size, intermediate_size_per_partition = super().maybe_roundup_sizes(
            hidden_size=hidden_size,
            intermediate_size_per_partition=intermediate_size_per_partition,
            act_dtype=act_dtype,
            moe_parallel_config=moe_parallel_config,
        )
        assert self.weight_block_size is not None
        block_size = self.weight_block_size[0] # 即 128
        # 将维度向上取整到 block_size 的整数倍
        return (
            round_up(hidden_size, block_size),
            round_up(intermediate_size_per_partition, block_size),
        )

    def _zero_padding(self, layer: Module) -> None:
        """将权重张量中填充区域（超出 unpadded 尺寸的部分）置零。"""
        hidden_size = layer.moe_config.hidden_dim_unpadded
        intermediate_size = layer.moe_config.intermediate_size_per_partition_unpadded

        w13_half_size = layer.w13_weight.shape[1] // 2
        # 处理 w13_weight：前一半和后一半分别清零填充部分
        if w13_half_size > intermediate_size:
            layer.w13_weight[:, intermediate_size:w13_half_size, :] = 0
            layer.w13_weight[
                :, w13_half_size + intermediate_size : 2 * w13_half_size, :
```

```
            ] = 0
        if layer.w13_weight.shape[2] > hidden_size:
            layer.w13_weight[:, :, hidden_size:] = 0

        if layer.w2_weight.shape[1] > hidden_size:
            layer.w2_weight[:, hidden_size:, :] = 0
        if layer.w2_weight.shape[2] > intermediate_size:
            layer.w2_weight[:, :, intermediate_size:] = 0

        # 处理可选的 bias
        if getattr(layer, "w13_bias", None) is not None:
            w13_bias_half_size = layer.w13_bias.shape[1] // 2
            if w13_bias_half_size > intermediate_size:
                layer.w13_bias[:, intermediate_size:w13_bias_half_size] = 0
                layer.w13_bias[
                    :, w13_bias_half_size + intermediate_size : 2 * w13_bias_half_size
                ] = 0
        if (
            getattr(layer, "w2_bias", None) is not None
            and layer.w2_bias.shape[1] > hidden_size
        ):
            layer.w2_bias[:, hidden_size:] = 0

    def process_weights_after_loading(self, layer: Module) -> None:
        if getattr(layer, "_already_called_process_weights_after_loading", False):
            return
        # 先清零填充区域，再进行后续量化
        self._zero_padding(layer)
        # 后续 FP8 量化逻辑保持不变 ...
```

评论区精华

- 审核者 yewentao256 要求添加完整复现命令和错误输出，以及 lm_eval 指标以证明精度无损。提交者 yma11 按要求补充了复现命令和错误堆栈，并提供了 TP=2（无填充）和 TP=4（有填充）的 gsm8k 准确率对比图。
- 提交者 yma11 在第一个评论中询问：“我注意到离线也不支持 block 量化的权重填充，所以想知道这种断裂是否是预期的——因为填充会带来计算 / 内存开销？或者我们应该支持这些情况但添加警告？”此问题在后续讨论中未获直接回应，但最终方案选择了填充。
- 复现命令与错误输出 (correctness): 提交者 yma11 在 PR 描述中补充了完整命令和错误堆栈。
- 精度验证 (testing): 提交者提供了 TP=2（无填充）和 TP=4（有填充）的 gsm8k 准确率对比图，精度接近。
- 填充开销与设计取舍 (design): 最终实现了填充方案，但讨论中未深入探讨性能权衡。

风险与影响

- 风险:

- 回归风险：较小的风险。仅修改了 Fp8PerBlockOnlineMoEMethod 类，该路径仅在启用 --quantization=fp8_per_block 且 MoE 中间维度不能被 128 整除时才激活。对于原本就能整除的模型（如多数标准配置），行为与之前完全一致。
- 精度影响：_zero_padding 将填充区域置零，理论上不会影响原本有效区域的数学计算。提交者提供的 lm_eval 结果也证实精度无损。
- 性能风险：填充会增加张量尺寸，导致额外的计算和显存占用。但对于当前场景，填充量通常很小（例如从 192 补齐到 256），影响可接受。
- 无测试配套：PR 未添加单元测试或集成测试，仅靠手动验证。建议后续补充测试用例，覆盖不可整除的常见场景。

- 影响：
  - 用户影响：修复了 Qwen3-30B-A3B 等模型在 TP=4 且使用 fp8_per_block 量化时的崩溃问题。用户现在可以正常使用这些配置。
  - 系统影响：仅影响在线 FP8 per-block 量化路径，其他量化方法和非 MoE 层不受影响。
  - 团队影响：低。改动集中在单一文件，逻辑清晰，容易维护。
  - 风险标记：核心路径变更，缺少测试覆盖

关联脉络

- PR #43673 [ROCm][Perf] DSv3.2: fuse MLA Q concat+fp8-quant in forward_mqa: 同为 FP8 量化相关优化，但涉及不同模块。
- PR #45404 fix(moe_wna16): access tp_size via moe_config for RoutedExperts compatibility: 同为 MoE 量化 bugfix，涉及 MoE 权重处理。