PR #43841 完整报告

vllm-project/vllm

[CPU] Migrate cpu_awq into awq_marlin

合并时间：2026-05-28 22:36

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/43841

执行摘要

- 一句话：CPU AWQ 迁移至 awq_marlin 并支持 W4A8
- 推荐动作：建议关注 cpu.py 中新增的 _process_gptq_weights_w4a8 函数，了解 W4A8 权重的重排逻辑。对于维护者，PR 展示了如何通过平台条件扩展重构量化后端。

功能与动机

PR 目的如 body 所述：弃用 cpu_awq 并复用 awq_marlin，以及为所有 WNA16 方法启用 W4A8 支持。这可以减少重复代码、统一量化后端，并扩展 CPU 上的量化精度选项。

实现拆解

1. 删除独立 CPU AWQ 配置和线性方法文件 vllm/model_executor/layers/quantization/cpu_wna16.py，该文件包含 CPUAWQConfig 类和相关权重处理逻辑。
2. 在 awq_marlin.py 中将 is_awq_marlin_compatible 方法中的平台检查从 is_cuda_alike() 扩展为 is_cuda_alike() or is_cpu()，使 AWQ Marlin 配置兼容 CPU 平台。
3. 重构 cpu.py 中的混合精度核：将原 _process_gptq_weights 方法拆分为 _process_gptq_weights_w4a16（处理 W4A16 格式）和新增 _process_gptq_weights_w4a8（处理 W4A8 格式）。新函数支持 W4A8 的权重重排和零点的伪造分配。
4. 清理量化方法和配置注册：从 __init__.py 的 QuantizationMethods、导入列表和 method_to_config 字典中移除 cpu_awq 和 CPUAWQConfig；从 config/model.py 的有效量化方法列表以及 v1/metrics/perf.py 的性能权重映射中移除 cpu_awq。
5. 更新 CPU 硬件测试配置 .buildkite/hardware_tests/cpu.yaml，将测试从 cpu_awq 替换为 awq_marlin。

关键文件：

- vllm/model_executor/layers/quantization/cpu_wna16.py（模块 量化层；类别 source；类型 deletion；符号 CPUAWQConfig, init, repr, get_name）：该文件是整个迁移的核心删除对象，包含 CPU AWQ 的所有配置和线性方法实现，被完全移除以统一量化路径。
- vllm/model_executor/kernels/linear/mixed_precision/cpu.py（模块 核函数；类别 source；类型 core-logic；符号 _process_gptq_weights_w4a16, _process_gptq_weights_w4a8）：该文件包含 CPU 混合精度线性核，被重构以支持 W4A8 和 W4A16 两种格式，新增 _process_gptq_weights_w4a8 方法，是功能扩展的关键文件。

- vllm/model_executor/layers/quantization/__init__.py（模块 量化层；类别 source；类型 data-contract）：入口文件，移除 cpu_awq 注册和导入，是迁移的必要清理步骤。
- vllm/model_executor/layers/quantization/awq_marlin.py（模块 量化层；类别 source；类型 data-contract；符号 is_awq_marlin_compatible）：扩展平台兼容性，允许 CPU 使用 awq_marlin 路径，是迁移的核心注册点。
- vllm/config/model.py（模块 配置；类别 source；类型 data-contract）：从有效量化方法列表中移除 cpu_awq，确保用户无法再指定该方法。
- vllm/v1/metrics/perf.py（模块 性能指标；类别 source；类型 core-logic）：从性能权重映射中移除 cpu_awq，与配置清理同步。
- .buildkite/hardware_tests/cpu.yaml（模块 CPU 平台；类别 test；类型 test-coverage）：更新 CPU 硬件测试，将 cpu_awq 替换为 awq_marlin，确保 CI 覆盖迁移后的路径。

关键符号：CPUAWQConfig, CPUWNA16LinearKernel._process_gptq_weights_w4a16, CPUWNA16LinearKernel._process_gptq_weights_w4a8, CPUWNA16LinearKernel.process_weights_after_loading, AWQMarlinConfig.is_awq_marlin_compatible

关键源码片段

vllm/model_executor/kernels/linear/mixed_precision/cpu.py

该文件包含 CPU 混合精度线性核，被重构以支持 W4A8 和 W4A16 两种格式，新增 _process_gptq_weights_w4a8 方法，是功能扩展的关键文件。

```
# vllm/model_executor/kernels/linear/mixed_precision/cpu.py
# 处理 W4A8 格式的 GPTQ 权重（4 比特权重、8 比特激活）
# 假设张量布局：weight_packed 的 input_dim=0，output_dim=1，packed_dim=0
# scale 和 zero_point 的 input_dim=0，output_dim=1
def _process_gptq_weights_w4a8(self, layer: torch.nn.Module):
    packed_weight = getattr(layer, self.w_q_name)
    scales = getattr(layer, self.w_s_name)
    group_num = scales.data.size(0)
    zp_output_size = scales.data.size(1) // 8
    if self.config.zero_points:
        assert self.w_zp_name
        packed_zp = getattr(layer, self.w_zp_name)
    else:
        # w4a8 内核始终需要 zero_point，若无则伪造一个全 1 的零
        assert self.w_zp_name
        fake_zp_value = torch.ones(group_num, zp_output_size, dtype=torch.int32) * -
        2004318072
        packed_zp = torch.nn.Parameter(fake_zp_value, requires_grad=False)
        setattr(layer, self.w_zp_name, packed_zp)
    # 由于 convert_weight_packed_scale_zp 在 GPTQ 格式上有 bug，
    # 这里直接手动重排为 AWQ 兼容格式
    weight = unpack_quantized_values_into_int32(packed_weight, self.config.weight_type, 0)
    input_size, output_size = weight.size()
    # 将 8 个 4 比特元素合并为 32 比特块后重新排列顺序
```

```
    weight = weight.view(input_size, output_size // 8, 8)
    weight = weight[:, :, (0, 2, 4, 6, 1, 3, 5, 7)].reshape(input_size, output_size)
    weight = pack_quantized_values_into_int32(weight, self.config.weight_type, 1).contiguous()
    # 对 zero_point 做类似重排并移除奇数通道
    zp = unpack_quantized_values_into_int32(packed_zp, self.config.weight_type, 1)
    zp = zp.view(group_num, output_size // 8, 8)
    # 后续还有 scale 和 zp 的进一步处理，此处省略
```

vllm/model_executor/layers/quantization/awq_marlin.py

扩展平台兼容性，允许 CPU 使用 awq_marlin 路径，是迁移的核心注册点。

```
@classmethod
def is_awq_marlin_compatible(cls, quant_config: dict[str, Any]) -> bool:
    # 从量化配置中提取关键参数
    quant_method = quant_config.get("quant_method", "").lower()
    num_bits = quant_config.get("bits")
    group_size = quant_config.get("group_size")
    zero_point = quant_config.get("zero_point")
    # 现在允许在 CUDA 和 CPU 平台上使用
    if not (current_platform.is_cuda_alike() or current_platform.is_cpu()):
        return False
    if quant_method != "awq":
        return False
    if num_bits is None or group_size is None or zero_point is None:
        return False
    if num_bits not in cls.TYPE_MAP:
        return False
    return check_marlin_supported(
        quant_type=cls.TYPE_MAP[num_bits],
        group_size=group_size,
        has_zp=zero_point,
    )
```

评论区精华

该 PR 没有 review 评论，仅获得批准（jikunshang）。因此无讨论精华。

- 暂无高价值评论线程

风险与影响

- 风险：主要风险包括：删除 cpu_awq 可能导致使用该旧方法的模型加载失败；新实现的 W4A8 核函数可能缺少充分测试覆盖；awq_marlin 在 CPU 上的兼容性边界情况可能未完全覆盖。建议关注回归测试结果。
- 影响：对用户的影响：AWQ 量化模型在 CPU 上仍可使用，但量化方法名从 cpu_awq 变为 awq_marlin，可能需要更新配置。对系统的影响：代码量减少约 350 行，统一量化路径有助于减少维护成本。对团队的影响：后续量化相关开发可集中在 awq_marlin 基础上。
- 风险标记：旧配置兼容性风险，W4A8 核缺少充分测试，CPU 平台回归风险

关联脉络

- 暂无明显关联 PR