PR #43409 完整报告

vllm-project/vllm

[CPU] Support CPU W4A16 INT4 MoE

合并时间：2026-06-12 15:12

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/43409

执行摘要

- 一句话：为 CPU 添加 W4A16 INT4 MoE 支持（AWQ/GPTQ/Compressed-Tensors）
- 推荐动作：建议团队精读该 PR，重点学习如何为新硬件后端添加 MoE 量化支持：包括后端注册、权重处理、内核封装和测试配套。Review 中的讨论也值得参考，尤其是关于 workspace 分配和零点传递的设计决策。

功能与动机

PR 描述明确目标：为 Intel CPU 支持 AWQ、GPTQ 和 Compressed-Tensors 的 W4A16 INT4 MoE。这填补了 CPU 平台在 MoE 模型量化推理上的空白，使用户能在 CPU 上部署如 Qwen3-30B-A3B 等 MoE 模型。

实现拆解

1. 注册 CPU 后端：在 int_wna16.py 的 WNA16MoEBackend 枚举中添加 CPU，并在 backend_to_kernel_cls 中映射到 CPUExpertsInt4；_get_priority_backends 优先返回 CPU，确保 CPU 平台默认使用该后端。
2. 实现 CPU INT4 MoE 专家类：在 cpu_moe.py 中新增 CPUExpertsInt4 类，继承 FusedMoEExpertsMonolithic，封装 fused_experts_cpu 内核调用，并声明 expects_unquantized_inputs=True 等属性。同时新增 prepare_int4_moe_layer_for_cpu 函数，处理零点（对称量化创建合成零点）并调用 convert_weight_packed_scale_zp 将 GPTQ/AWQ 格式权重转换为 blocked 格式。
3. 权重处理适配：在 int_wna16.py 的 convert_to_wna16_moe_kernel_format 中添加 CPU 分支，调用 _process_weights_cpu 函数，负责检测量化格式、解析零点、处理 bias 类型转换，并调用 prepare_int4_moe_layer_for_cpu。对 desc_act=True 的 GPTQ 模型显式抛出 NotImplementedError。
4. 量化框架调整：修改 auto_gptq.py、awq_marlin.py 和 compressed_tensors_moe_wna16_marlin.py：在 create_weights 中仅当 experts_cls 是 FusedMoEExpertsModular 子类时才分配 Marlin workspace；在 process_weights_after_loading 中保留并传递 qzeros；添加 apply_monolithic 方法。
5. 测试与部署：新增 test_cpu_quant_fused_moe.py 包含参考实现和内核一致性测试；在 test_cpu_wna16.py 中添加 E2E 测试；更新 cpu.yaml 以运行新测试。

关键文件：

- vllm/model_executor/layers/fused_moe/experts/cpu_moe.py（模块 CPU MoE 专家；类别 source；类型 core-logic；符号 prepare_int4_moe_layer_for_cpu, CPUExpertsInt4, init, expects_unquantized_inputs）：核心实现：新增 CPUExpertsInt4 类和权重预处理函数 prepare_int4_moe_layer_for_cpu
- vllm/model_executor/layers/fused_moe/oracle/int_wna16.py（模块 MoE 调度器；类别 source；类型 core-logic；符号 _process_weights_cpu）：后端调度：添加 CPU 后端选择和权重处理函数 _process_weights_cpu
- tests/kernels/moe/test_cpu_quant_fused_moe.py（模块 MoE 测试；类别 test；类型 test-coverage；符号 _pack_int4_gptq, _pack_int4_awq, _ref_int4_moe, _make_int4_moe_weights）：测试覆盖：新增 INT4 MoE 参考实现和内核一致性测试
- vllm/model_executor/layers/quantization/auto_gptq.py（模块 GPTQ 量化；类别 source；类型 data-contract；符号 apply_monolithic）：量化适配：调整 GPTQ MoE 路径以支持 CPU，包括 workspace 分配和零点传递
- vllm/model_executor/layers/quantization/awq_marlin.py（模块 AWQ 量化；类别 source；类型 data-contract；符号 apply_monolithic）：量化适配：类似 GPTQ 路径的调整
- vllm/model_executor/layers/quantization/compressed_tensors/compressed_tensors_moe/compressed_tensors_moe_wna16_marlin.py（模块 CompressedTensors；类别 source；类型 data-contract）：量化适配：调整 CompressedTensors MoE 路径
- tests/quantization/test_cpu_wna16.py（模块 CPU 量化测试；类别 test；类型 test-coverage）：E2E 测试：验证完整推理路径
- .buildkite/hardware_tests/cpu.yaml（模块 CI 配置；类别 infra；类型 test-coverage）：CI 配置：启用新测试

关键符号：prepare_int4_moe_layer_for_cpu, CPUExpertsInt4.init, CPUExpertsInt4.apply, CPUExpertsInt4.expects_unquantized_inputs, CPUExpertsInt4.activation_format, _process_weights_cpu, _pack_int4_gptq, _pack_int4_awq, _ref_int4_moe, _make_int4_moe_weights, test_int4_w4a16_cpu_fused_moe, apply_monolithic

关键源码片段

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

核心实现：新增 CPUExpertsInt4 类和权重预处理函数 prepare_int4_moe_layer_for_cpu

```
# SPDX-License-Identifier: Apache-2.0
# SPDX-FileCopyrightText: Copyright contributors to the vLLM project

import torch
from vllm._custom_ops import CPUQuantAlgo, CPUQuantMethod, convert_weight_packed_scale_
zp, fused_experts_cpu
from vllm.model_executor.layers.fused_moe import mk
# ... 其他导入略

def prepare_int4_moe_layer_for_cpu(
    w13_packed: torch.Tensor, # [E, K//8, 2*I] int32 (packed int4)
```

```
    w2_packed: torch.Tensor, # [E, I//8, K] int32
    w13_scale: torch.Tensor, # [E, num_groups, 2*I] float16/bf16
    w2_scale: torch.Tensor, # [E, num_groups, K] float16/bf16
    quant_algo: CPUQuantAlgo = CPUQuantAlgo.GPTQ,
    w13_zeros: torch.Tensor | None = None,
    w2_zeros: torch.Tensor | None = None,
) -> tuple[torch.Tensor, torch.Tensor, torch.Tensor, torch.Tensor, torch.Tensor, torch.Tensor]:
    """将 GPTQ/AWQ packed 权重转换为 CPU 内核所需的 blocked 格式。"""
    # 若零点为 None（对称量化），则创建合成零点 0x77777777
    # GPTQ 解包内核会对存储零点 +1，存储 7 得到真实零点 8
    if w13_zeros is None:
        num_groups = w13_scale.size(1)
        N = w13_scale.size(2) # 2*I
        w13_zeros = torch.full((w13_packed.size(0), num_groups, N // 8),
                               0x77777777, dtype=torch.int32)
    if w2_zeros is None:
        num_groups = w2_scale.size(1)
        N = w2_scale.size(2) # K
        w2_zeros = torch.full((w2_packed.size(0), num_groups, N // 8),
                               0x77777777, dtype=torch.int32)

    # 调用自定义操作进行格式转换
    blocked_w13, blocked_z13, blocked_s13 = convert_weight_packed_scale_zp(
        w13_packed, w13_zeros, w13_scale, quant_algo)
    blocked_w2, blocked_z2, blocked_s2 = convert_weight_packed_scale_zp(
        w2_packed, w2_zeros, w2_scale, quant_algo)
    return (blocked_w13, blocked_w2, blocked_s13, blocked_s2, blocked_z13, blocked_z2)


class CPUExpertsInt4(mk.FusedMoEExpertsMonolithic):
    """CPU INT4 W4A16 分组量化 MoE Monolithic 专家。"""
    def __init__(self, moe_config, quant_config):
        super().__init__(moe_config, quant_config)
        # 直接使用继承自基类的 w1_scale、w1_zp 等属性，避免冗余

    @property
    def expects_unquantized_inputs(self) -> bool:
        return True # 内核内部处理量化

    @property
    def activation_format(self):
        return mk.FusedMoEActivationFormat.Standard

    def apply(self, hidden_states, ..., topk_weights, topk_ids, ...):
        # ... 前置逻辑（获取 scale/zp）...
        return fused_experts_cpu(
            hidden_states, self.w1, self.w2, topk_weights, topk_ids,
            False, CPUQuantMethod.INT4_W4A8,
            self.w1_scale, self.w2_scale,
```

```
            self.w1_zp, self.w2_zp, # 传递零点
            self.quant_config.block_shape[1], # group_size
            w1_bias, w2_bias, alpha, limit, is_vnni=True)
```

vllm/model_executor/layers/fused_moe/oracle/int_wna16.py

后端调度：添加 CPU 后端选择和权重处理函数 _process_weights_cpu

```
def _process_weights_cpu(
    quant_config, w13, w2, w13_scale, w2_scale,
    w13_g_idx=None, w2_g_idx=None,
    w13_qzeros=None, w2_qzeros=None,
    w13_bias=None, w2_bias=None,
):
    """CPU INT4 W4A16 权重后处理：检测格式并调用 prepare_int4_moe_layer_for_cpu。"""
    # 通过形状判断 packed 格式
    # AWQ: qweight 形状为 [E, K, 2*N//8]（沿输出 /N 维打包）
    # GPTQ: qweight 形状为 [E, K//8, 2*N]（沿输入 /K 维打包）
    if w13.dim() == 3 and w13.size(2) * 8 > w13.size(1):
        quant_format = "GPTQ"
        quant_algo = CPUQuantAlgo.GPTQ
    else:
        quant_format = "AWQ"
        quant_algo = CPUQuantAlgo.AWQ

    # 处理 bias：转换为 float32
    if w13_bias is not None:
        w13_bias = w13_bias.to(torch.float32)
    if w2_bias is not None:
        w2_bias = w2_bias.to(torch.float32)

    # 调用核心重打包函数
    (w13_qweight, w2_qweight, w13_scales, w2_scales,
     w13_qzeros, w2_qzeros) = prepare_int4_moe_layer_for_cpu(
        w13, w2, w13_scale, w2_scale, quant_algo,
        w13_qzeros, w2_qzeros)

    return (w13_qweight, w2_qweight, w13_scales, w2_scales,
            w13_g_idx, w2_g_idx, None, None, # g_idx 和 sort_indices
            w13_qzeros, w2_qzeros, # 返回零点供后续使用
            None, None, w13_bias, w2_bias)
```

tests/kernels/moe/test_cpu_quant_fused_moe.py

测试覆盖：新增 INT4 MoE 参考实现和内核一致性测试

```
def _ref_int4_moe(a, w1_int4, w2_int4, w1_zeros, w2_zeros, w1_s, w2_s,
                  topk_weight, topk_ids, group_size):
    """纯 PyTorch 参考实现：按 expert 循环，反量化并计算。"""
    B, topk = a.shape[0], topk_ids.size(1)
    out = torch.zeros(B, topk, a.shape[1], dtype=torch.float32)
```

```
    for b in range(B):
        for t in range(topk):
            eid = topk_ids[b, t].item()
            x = a[b:b+1].float()
            # 反量化 w1: [K, 2*N]
            K_dim = w1_int4.shape[1]
            w1_dq = torch.zeros(K_dim, w1_int4.shape[2], dtype=torch.float32)
            for g in range(w1_s.shape[1]):
                k_start = g * group_size
                k_end = min((g+1)*group_size, K_dim)
                zp = w1_zeros[eid, g, :].float() if w1_zeros is not None else 8.0
                w1_dq[k_start:k_end, :] = (
                    w1_int4[eid, k_start:k_end, :].float() - zp) * w1_s[eid, g, :].float()
            ic = torch.matmul(x, w1_dq) # [1, K] @ [K, 2*N] -> [1, 2*N]
            ic = silu_and_mul(ic) # 激活 + 乘法
            # 反量化 w2: [N, K]
            N_dim = w2_int4.shape[1]
            w2_dq = torch.zeros(N_dim, w2_int4.shape[2], dtype=torch.float32)
            for g in range(w2_s.shape[1]):
                n_start = g * group_size
                n_end = min((g+1)*group_size, N_dim)
                zp = w2_zeros[eid, g, :].float() if w2_zeros is not None else 8.0
                w2_dq[n_start:n_end, :] = (
                    w2_int4[eid, n_start:n_end, :].float() - zp) * w2_s[eid, g, :].float()
            oc = torch.matmul(ic, w2_dq) # [1, N] @ [N, K] -> [1, K]
            out[b, t] = oc.squeeze(0)
    return (out * topk_weight.unsqueeze(-1)).sum(dim=1).to(a.dtype)


def test_int4_w4a16_cpu_fused_moe(M, N, K, E, topk, group_size, quant_algo):
    a = torch.randn(M, K, dtype=torch.bfloat16)
    # 创建随机 INT4 权重并打包
    w1_int4, w2_int4, w1_packed, w2_packed, w1_zeros, w2_zeros, w1_zeros_packed, w2_zeros_
    packed, w1_s, w2_s = \
        _make_int4_moe_weights(E, N, K, group_size, quant_algo)
    topk_ids = torch.randint(0, E, (M, topk))
    topk_weights = torch.randn(M, topk).softmax(dim=-1)
    # 调用参考实现
    ref_out = _ref_int4_moe(a, w1_int4, w2_int4, w1_zeros, w2_zeros, w1_s, w2_s,
                            topk_weights, topk_ids, group_size)
    # 调用 CPU 内核（需先通过 prepare_int4_moe_layer_for_cpu）
    # ... 验证一致性
```

评论区精华

在 Review 中，主要讨论包括：

- GPTQ 的 desc_act 支持：bigPYJ1151 指出应检查 desc_act=True 的 GPTQ 模型并抛出错误，作者已添加。

- bias 类型转换：bigPYJ1151 要求 bias 转为 float32，作者在 _process_weights_cpu 中修复。
- 冗余属性 vs 继承属性：gemini-code-assist[bot] 建议 CPUExpertsInt4 直接使用基类属性，作者采纳。
- workspace 分配条件：bigPYJ1151 建议基于 experts_cls 是否为 FusedMoEExpertsModular 判断，作者修改。
  - GPTQ desc_act=True 的异常处理 (correctness): 作者在 _process_weights_cpu 中添加了显式异常抛出。
  - bias 类型转换为 float32 (correctness): 作者在 _process_weights_cpu 中进行了转换。
  - workspace 分配条件的设计权衡 (design): 作者修改了条件判断，使用 issubclass(self.experts_cls, FusedMoEExpertsModular)。

风险与影响

- 风险：经评估，存在以下风险：
  1. 破坏 NVIDIA GPU 量化：评论指出本 PR 修改了 auto_gptq.py 中 get_fused_moe_quant_config 的零点传递逻辑，导致 Autoround 量化的 NVIDIA GPU 推理失败。虽然可能已在后续分支修复，但合并时需确保兼容性。
  2. 对 GPTQ desc_act=True 的支持缺失：CPU 后端尚不支持 desc_act=True 的 GPTQ 权重，若用户尝试使用，会抛出异常。需确保文档明确标注此限制。
  3. bias 类型转换：若模型不包含 bias，转换路径需确保安全处理 None。当前代码已涵盖，但测试覆盖可能不足。
  4. 回归风险：对 compressed_tensors_moe_wna16_marlin.py 中 if not self.symmetric 替换为 if w13_qzeros is not None 可能改变其他后端的零点行为，需确认无副作用。 - 影响：对用户：CPU 用户可直接加载 AWQ、GPTQ、Compressed-Tensors 格式的 INT4 MoE 模型，无需额外转换。对系统：扩展了 MoE 后端的可插拔架构，但新增了条件分支，可能增加维护成本。对团队：需跟踪后续可能出现的兼容性问题（如 Autoround）。影响程度中等。 - 风险标记：核心路径变更，破坏 NVIDIA Autoround，缺少 GPTQ desc_act 支持，bias 转换风险

关联脉络

- PR #44400 [ROCm][Perf] Enable W4A16 FlyDSL MoE: 同样扩展 MoE 量化后端到 ROCm 平台，遵循类似的模式（int_wna16.py 注册后端、新增强化专家类）。
- PR #45136 [XPU] Support int4 group_size=32 W4A16 MoE: 为 XPU 添加 INT4 MoE 支持，与本 PR 的 CPU MoE 实现共享同一框架。
- PR #45391 [CPU] Refine CPU attention frontend: 同一平台（CPU）的 MoE 相关改进，优化了 CPU attention 前端，为 MoE 推理提供基础。