PR #41186 完整报告

vllm-project/vllm

[CPU] Add FP8 W8A16 linear support

合并时间：2026-05-06 15:05

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/41186

执行摘要

- 一句话：CPU 端 FP8 W8A16 块量化线性支持
- 推荐动作：建议阅读者重点关注：
  - CPUFp8BlockScaledMMKernel 的实现模板，可参考添加其他 CPU 量化核。
  - process_weights_after_loading 中权重打包和参数替换的模式。
  - 注册位置争议，需与框架维护者确认 _POSSIBLE_FP8_BLOCK_KERNELS 与 _POSSIBLE_WFP8A16_KERNELS 的语义差异。
  - 测试套件的参考实现组织方式。

功能与动机

为 Intel CPU 提供 FP8 W8A16 块量化线性层加速，参考 sglang 实现，利用 AMX BRGEMM 指令集提升推理性能。PR body 明确说明“Add FP8 W8A16 block-quantized linear support for Intel CPUs”。

实现拆解

1. 新增 CPUFp8BlockScaledMMKernel 类（vllm/model_executor/kernels/linear/scaled_mm/cpu.py）：继承 Fp8BlockScaledMMLinearKernel，设置 apply_input_quant = False（激活保持 BF16 不量化），重写 is_supported（检查 CPU 平台、AMX 指令存在和算子可用性）、can_implement（校验权重块形状、输出 dtype）、process_weights_after_loading（VNNI 预打包权重并用 replace_parameter 替换为 Parameter）、apply_weights（调用 C++ 算子执行块量化 GEMM）。
2. C++ 算子绑定（csrc/cpu/torch_bindings.cpp）：声明和注册 fp8_scaled_mm_cpu 函数，适配 AMX 的块量化 GEMM，并与现有 #if 宏一起编译。
3. Python 入口和 fake 注册（vllm/_custom_ops.py）：新增 fp8_scaled_mm_cpu 函数转发至 C++ 算子，同时注册 fake 实现支持 meta device 推断，暴露 _supports_cpu_fp8_w8a16 标志供上层检测。
4. 集成到模型执行器（vllm/model_executor/kernels/linear/__init__.py 与 scaled_mm/__init__.py）：导出 CPUFp8BlockScaledMMKernel 并注册到 _POSSIBLE_FP8_BLOCK_KERNELS 字典（注：此注册位置可能与预期不符，详见讨论）。
5. 测试与 CI 配置：新增 tests/kernels/quantization/test_cpu_fp8_scaled_mm.py 包含参考量化 / 去量化实现和参数化测试；修改 .buildkite/hardware_tests/cpu.yaml 将超时增加至

30m 并加入新测试；在 tests/quantization/test_cpu_wna16.py 添加 Qwen3-0.6B-FP8烟囱测试。

关键文件：

- vllm/model_executor/kernels/linear/scaled_mm/cpu.py（模块 量化核；类别 source；类型 core-logic；符号 CPUFp8BlockScaledMMKernel, is_supported, can_implement, process_weights_after_loading）：核心实现文件，新增 CPU FP8 块量化线性核的全部逻辑：硬件检测、配置校验、权重预处理和前向计算。
- tests/kernels/quantization/test_cpu_fp8_scaled_mm.py（模块 测试；类别 test；类型 test-coverage；符号 cdiv, quantize_weight_block_fp8, dequant_weight_block_fp8, ref_fp8_block_scaled_mm）：完整测试套件，提供参考量化 / 去量化实现和参数化正确性测试，覆盖多种形状和偏置，确保 kernel 正确性。
- vllm/_custom_ops.py（模块 算子注册；类别 source；类型 core-logic；符号 fp8_scaled_mm_cpu_fake, fp8_scaled_mm_cpu, _supports_cpu_fp8_w8a16）：注册 FP8 W8A16 CPU 算子的 fake op 和 Python 包装函数，提供运行时特性检测标志，是算子调用的入口。

关键符号：CPUFp8BlockScaledMMKernel.is_supported, CPUFp8BlockScaledMMKernel.can_implement, CPUFp8BlockScaledMMKernel.process_weights_after_loading, CPUFp8BlockScaledMMKernel.apply_weights, fp8_scaled_mm_cpu, fp8_scaled_mm_cpu_fake, quantize_weight_block_fp8, dequant_weight_block_fp8, ref_fp8_block_scaled_mm

关键源码片段

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

核心实现文件，新增 CPU FP8 块量化线性核的全部逻辑：硬件检测、配置校验、权重预处理和前向计算。

```
def process_weights_after_loading(self, layer: torch.nn.Module) -> None:
    # 跳过 GPU 导向的基类处理（FP8 填充 / fnuz 归一化）
    params = self._get_layer_params(layer)

    # 使用 VNNI 格式对权重进行预打包，适配 AMX BRGEMM 要求
    packed_weight = torch.ops._C.convert_weight_packed(params.weight)
    # 替换权重为 Parameter 避免丢失状态（如 .parameters() 遍历）
    replace_parameter(
        layer,
        params.WEIGHT,
        torch.nn.Parameter(packed_weight, requires_grad=False),
    )

    # 同样将 scale 包装为 Parameter，以兼容 weight-loader
    scale_attr = (
        params.WEIGHT_SCALE_INV
```

```
        if params.weight_scale_inv is not None
        else params.WEIGHT_SCALE
    )
    weight_scale = (
        params.weight_scale_inv
        if params.weight_scale_inv is not None
        else params.weight_scale
    )
    assert weight_scale is not None
    replace_parameter(
        layer,
        scale_attr,
        torch.nn.Parameter(weight_scale.data, requires_grad=False),
    )
```

tests/kernels/quantization/test_cpu_fp8_scaled_mm.py

完整测试套件，提供参考量化 / 去量化实现和参数化正确性测试，覆盖多种形状和偏置，确保 kernel 正确性。

```
def quantize_weight_block_fp8(
    weight: torch.Tensor,
    block_size: list[int],
) -> tuple[torch.Tensor, torch.Tensor]:
    """将权重 [N, K] 量化为 FP8 块格式，返回 FP8 权重和 block scales。

    Returns:
        fp8_weight: [N, K] float8_e4m3fn
        scales: [n_tiles, k_tiles] float32
    """
    N, K = weight.shape
    block_n, block_k = block_size
    fp8_max = torch.finfo(torch.float8_e4m3fn).max

    n_tiles = cdiv(N, block_n)
    k_tiles = cdiv(K, block_k)

    # 将权重补齐至块边界，确保每个块大小一致
    pad_N = (block_n - (N % block_n)) % block_n
    pad_K = (block_k - (K % block_k)) % block_k
    if pad_N > 0 or pad_K > 0:
        weight = torch.nn.functional.pad(weight, (0, pad_K, 0, pad_N))

    # 重排为 [n_tiles, block_n, k_tiles, block_k] 格式
    w_blocks = weight.view(n_tiles, block_n, k_tiles, block_k)
    w_blocks = w_blocks.permute(0, 2, 1, 3).contiguous()

    # 计算每个 block 绝对最大值，按 FP8 最大值缩放
    abs_max = w_blocks.abs().amax(dim=(-2, -1), keepdim=True)
    scales = abs_max / fp8_max
```

```
    scales = torch.where(scales == 0, torch.ones_like(scales), scales)

    # 块内缩放并量化至 FP8
    q_fp8 = (w_blocks / scales).clamp(-fp8_max, fp8_max).to(torch.float8_e4m3fn)

    # 恢复原 shape，剔除 padding
    fp8_weight = (
        q_fp8.permute(0, 2, 1, 3)
        .contiguous()
        .view(N + pad_N, K + pad_K)[:N, :K]
        .contiguous()
    )
    scales = scales.view(n_tiles, k_tiles)
    return fp8_weight, scales
```

vllm/_custom_ops.py

注册 FP8 W8A16 CPU 算子的 fake op 和 Python 包装函数，提供运行时特性检测标志，是算子调用的入口。

```
# 检查底层 C++ 算子是否存在
if hasattr(torch.ops._C, "fp8_scaled_mm_cpu"):

    @register_fake("_C::fp8_scaled_mm_cpu")
    def fp8_scaled_mm_cpu_fake(
        mat1: torch.Tensor,
        mat2: torch.Tensor,
        scales2: torch.Tensor,
        block_size: list[int],
        bias: torch.Tensor | None,
        out_dtype: torch.dtype,
        is_vnni: bool,
    ) -> torch.Tensor:
        """Fake 实现用于 meta device / shape 推断，不执行实际计算."""
        M = mat1.size(0)
        N = mat2.size(0)
        return torch.empty((M, N), dtype=out_dtype, device=mat1.device)


# 运行时标志供上层判断是否支持
_supports_cpu_fp8_w8a16 = bool(hasattr(torch.ops._C, "fp8_scaled_mm_cpu"))


def fp8_scaled_mm_cpu(
    mat1: torch.Tensor,
    mat2: torch.Tensor,
    scales2: torch.Tensor,
    block_size: list[int],
    bias: torch.Tensor | None,
    out_dtype: torch.dtype,
```

```
    is_vnni: bool,
) -> torch.Tensor:
    """将调用转发至 C++ 算子 fp8_scaled_mm_cpu."""
    return torch.ops._C.fp8_scaled_mm_cpu(
        mat1, mat2, scales2, block_size, bias, out_dtype, is_vnni
    )
```

评论区精华

Review 中核心讨论集中在以下四点：

- 注册位置争议：gemini-code-assist 指出 CPUFp8BlockScaledMMKernel 被注册在 _POSSIBLE_FP8_BLOCK_KERNELS，但该字典用于激活和权重均块量化的场景（如 DeepSeek-V3），而 W8A16 核应属于 _POSSIBLE_WFP8A16_KERNELS。最终合并代码未更改，可能存在设计考量或遗漏。
- can_implement 校验对象错误：机器人最初建议检查 weight_quant_key 而非 activation_quant_key，已修复。
- 参数类型丢失：process_weights_after_loading 中 packed_weight 和 weight_scale 应包装为 torch.nn.Parameter，否则影响 model.parameters() 和 state_dict，已采纳修正。
- CI 配置：bigPYJ1151 建议增加超时和添加烟囱测试，均已实施。
  - CPUFp8BlockScaledMMKernel 注册位置 (correctness): 最终代码维持原注册位置，未采纳建议，可能为设计考量或遗漏。
  - can_implement 校验对象错误 (correctness): 最终代码已修正为 weight_quant_key，问题已解决。
  - 权重和尺度未包装为 Parameter (correctness): 最终代码已使用 replace_parameter 和 nn.Parameter 包装，问题已解决。
  - CI 超时和烟囱测试建议 (testing): 超时已调整，烟囱测试已添加，问题已解决。

风险与影响

- 风险：
  1. 注册位置风险：CPUFp8BlockScaledMMKernel 被注册在 _POSSIBLE_FP8_BLOCK_KERNELS 而非 _POSSIBLE_WFP8A16_KERNELS，可能导致对应 W8A16 层无法自动选到此核，用户需手动指定或回退。
  2. 硬件依赖风险：is_supported 检查 torch.cpu._is_amx_tile_supported()，仅 Sapphire Rapids 及以上 CPU 支持，老旧平台会跳过此核，但不会崩溃。
  3. 算子编译兼容性：C++ 算子仅通过 #if defined(__AVX512BF16__) && defined(__AVX512F__) && defined(__AVX512VNNI__) 条件编译，若未定义则 fp8_scaled_mm_cpu 不会注册，但 Python 侧通过 hasattr 安全检测。
  4. 测试覆盖局限：测试参数覆盖了多种 shape 和偏置，但未测试极端大 batch（M>128）或混合精度场景。 - 影响：用户影响：Intel CPU 用户（AMX 支持）可直接利用 FP8 量化加速线性层，推理性能提升；不支持的 CPU 无影响。系统影响：新增约 330 行代码，线性层计算路径增加 CPU 专用分支，不影响 GPU 或其他平台。团队影响：维护成本较

低，与现有 CPUInt8ScaledMMLinearKernel 风格一致；需确保后续 FP8 量化模型在 CPU 上能正确选核。

- 风险标记：注册位置可能错误，依赖 AMX 硬件指令，算子编译条件宏限制

关联脉络

- PR #41318 [Feat] dnnl build for AVX2 W8A8 Int8: 同属 CPU 量化加速支持，共享 csrc/cpu/torch_bindings.cpp 和 cmake 构建，本 PR 扩展了 CPU 上的量化计算能力。
- PR #41387 [Fix] Add missing stubs from cpu fp8 attention changes: 解决了 CPU FP8 编译问题，为本 PR 的 FP8 核提供干净的编译环境。