PR #41813 完整报告

vllm-project/vllm

[CPU][Zen] Route W8A8 and W4A16 linear inference through zentorch on AMD Zen CPUs

合并时间：2026-05-31 03:17

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/41813

执行摘要

- 一句话：AMD Zen CPU 上 zentorch 加速 W8A8/W4A16 线性层
- 推荐动作：值得精读，尤其是 kernel 选择器 fallback 设计、平台检测函数抽象以及量化权重兼容性检查。建议在后续 PR 中考虑引入 PlatformEnum.ZEN 并增加端到端集成测试。

功能与动机

AMD Zen CPU 用户运行量化模型时，通用 oneDNN 核无法利用 zentorch 定制算子的性能优势。此 PR 从 RFC #41629 出发，为 vLLM 在 Zen CPU 上的部署提供通道级优化，同时通过 fallback 保证兼容性。

实现拆解

1. 创建独立 zentorch 后端文件：在 scaled_mm/zentorch.py 和 mixed_precision/zentorch.py 中分别新增 ZentorchInt8ScaledMMLinearKernel 和 ZentorchWNA16LinearKernel，继承自对应的 CPU 基类，覆写 is_supported、can_implement、process_weights_after_loading 和 apply_weights。
2. 实现平台和 op 可用性检查：通过 current_platform.is_zen_cpu() 和 has_zentorch_op 函数（定义于 zentorch_utils.py）确保仅在 AMD Zen CPU 且 zentorch 库已安装时激活。
3. 实现权重重排：ZentorchWNA16LinearKernel 在 _zentorch_woq_eligible 中检验压缩张量格式并排除带 g_idx 的层，然后通过 process_weights_after_loading 调用 torch.ops.zentorch.zentorch_woq_repack_weight 将 GPTQ 权重重排为 zentorch 布局；同时清理未使用的 zero_point 和 g_idx 属性。
4. 集成到 kernel 选择器：在 linear/__init__.py 中导入新 kernel，置于 CPU 平台列表队首，确保优先选择；在 _LINEAR_BACKEND_KERNEL_MAP 中添加条目；在 register_linear_kernel 的公共 API 中添加名称。
5. 更新依赖和清理：在 setup.py 中将 zentorch 依赖固定为 zentorch==5.2.1，移除过时的 weekly 注释。

关键文件：

- vllm/model_executor/kernels/linear/mixed_precision/zentorch.py（模块 W4A16 核；类别 source；类型 core-logic；符号 _import_unpack_from_int32, ZentorchWNA16LinearKernel, can_implement, _zentorch_woq_eligible）：新增的 zentorch W4A16 量化线性核后端，包含资格检查、权重重排和推理调用，是此 PR 的核心

实现之一。

- vllm/model_executor/kernels/linear/scaled_mm/zentorch.py（模块 W8A8 核；类别 source；类型 core-logic；符号 ZentorchInt8ScaledMMLinearKernel, is_supported, can_implement, process_weights_after_loading）：新增的 zentorch W8A8 动态 symmetric 量化线性核后端，包含 is_supported/can_implement 检查、权重格式转换和推理调用。
- vllm/model_executor/kernels/linear/zentorch_utils.py（模块 工具函数；类别 source；类型 utility；符号 has_zentorch_op）：提供 has_zentorch_op 函数，统一平台检测和 op 可用性检查，被所有 zentorch 后端调用。
- vllm/model_executor/kernels/linear/__init__.py（模块 核注册；类别 source；类型 entrypoint）：主入口导入并注册新 kernel，调整优先级列表，使 zentorch kernel 优先于 oneDNN kernel。
- vllm/model_executor/kernels/linear/mixed_precision/__init__.py（模块 核注册；类别 source；类型 entrypoint）：导出 ZentorchWNA16LinearKernel 用于子包公开 API。
- vllm/model_executor/kernels/linear/scaled_mm/__init__.py（模块 核注册；类别 source；类型 entrypoint）：导出 ZentorchInt8ScaledMMLinearKernel 用于子包公开 API。
- setup.py（模块 构建配置；类别 config；类型 configuration）：将 Zen CPU 优化依赖从 weekly 版本切换到官方稳定版本 zentorch==5.2.1，并移除过时注释。

关键符号：has_zentorch_op, ZentorchInt8ScaledMMLinearKernel.is_supported, ZentorchInt8ScaledMMLinearKernel.can_implement, ZentorchInt8ScaledMMLinearKernel.process_weights_after_loading, ZentorchInt8ScaledMMLinearKernel.apply_weights, ZentorchWNA16LinearKernel.can_implement, ZentorchWNA16LinearKernel._zentorch_woq_eligible, ZentorchWNA16LinearKernel.process_weights_after_loading, ZentorchWNA16LinearKernel.apply_weights

关键源码片段

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

新增的 zentorch W4A16 量化线性核后端，包含资格检查、权重重排和推理调用，是此 PR 的核心实现之一。

```
class ZentorchWNA16LinearKernel(CPUWNA16LinearKernel):
    """W4A16 GPTQ kernel backed by ``torch.ops.zentorch.zentorch_woq_linear``."""

    @classmethod
    def can_implement(cls, c: MPLinearLayerConfig) -> tuple[bool, str | None]:
        ok, reason = super().can_implement(c)
        if not ok:
            return ok, reason
        # 检查平台：仅 AMD Zen CPU
        if not current_platform.is_zen_cpu():
            return False, "ZentorchWNA16 requires an AMD Zen CPU."
```

```
        # 检查 zentorch op 注册
        if not has_zentorch_op(["zentorch_woq_repack_weight", "zentorch_woq_linear"]):
            return (False, "torch.ops.zentorch.{...} are not registered.")
        # 不支持激活重排（g_idx）
        if c.has_g_idx:
            return False, "ZentorchWNA16 does not support activation re-ordering."
        return True, None

    def _zentorch_woq_eligible(self, layer: torch.nn.Module) -> bool:
        # 排除带 g_idx 的层
        if (self.w_gidx_name is not None
            and getattr(layer, self.w_gidx_name, None) is not None) \
           or getattr(self.config, "has_g_idx", False):
            return False
        # 必须存在权重和尺度
        weight_packed = getattr(layer, self.w_q_name, None)
        weight_scale = getattr(layer, self.w_s_name, None)
        if weight_packed is None or weight_scale is None:
            return False
        bits = self.config.weight_type.mantissa
        pack_factor = torch.iinfo(weight_packed.dtype).bits // bits
        # 仅支持 4-bit 打包（每个 int32 存 8 个值）
        if pack_factor != 8:
            return False
        # 仅支持 compressed-tensors 格式（input_dim == packed_dim == 1）
        in_dim = getattr(weight_packed, "input_dim", None)
        pk_dim = getattr(weight_packed, "packed_dim", None)
        if not (in_dim == pk_dim == 1):
            return False
        if weight_packed.dim() != 2 or weight_scale.dim() != 2:
            return False
        in_features = weight_packed.shape[1] * 8
        num_groups = weight_scale.shape[1]
        return num_groups > 0 and in_features % num_groups == 0
```

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

新增的 zentorch W8A8 动态 symmetric 量化线性核后端，包含 is_supported/can_implement 检查、权重格式转换和推理调用。

```
class ZentorchInt8ScaledMMLinearKernel(Int8ScaledMMLinearKernel):
    @classmethod
    def is_supported(cls, compute_capability=None) -> tuple[bool, str | None]:
        if not current_platform.is_cpu():
            return False, "requires CPU."
        if not current_platform.is_zen_cpu():
            return False, "requires AMD Zen CPU."
        if not has_zentorch_op(["zentorch_dynamic_qlinear"]):
            return False, "torch.ops.zentorch.zentorch_dynamic_qlinear is not registered."
        return True, None
```

```
    @classmethod
    def can_implement(cls, c: Int8ScaledMMLinearLayerConfig) -> tuple[bool, str | None]:
        if c.is_static_input_scheme:
            return False, "requires dynamic activation quantization."
        if not c.input_symmetric:
            return False, "requires symmetric activation quantization."
        if not c.is_channelwise:
            return False, "requires per-channel weight quantization."
        return True, None

    def process_weights_after_loading(self, layer: torch.nn.Module) -> None:
        w_q_name, w_s_name, _, _, _ = self.layer_param_names
        weight = getattr(layer, w_q_name)
        n = weight.shape[0]
        # 权重保持 [N, K] 布局且 contiguous
        replace_parameter(layer, w_q_name,
            torch.nn.Parameter(weight.data.contiguous(), requires_grad=False))
        # 权重尺度转换为 bf16 并调整为 (N,) 形状
        weight_scale = getattr(layer, w_s_name)
        ws = weight_scale.data
        if ws.dim() == 2 and ws.shape[-1] == 1:
            ws = ws.squeeze(-1)
        ws = ws.to(torch.bfloat16).contiguous()
        assert ws.shape == (n,), f"expected ({n},), got {tuple(ws.shape)}"
        replace_parameter(layer, w_s_name,
            torch.nn.Parameter(ws, requires_grad=False))
        logger.info_once("[zen_cpu] Using zentorch_dynamic_qlinear for W8A8 (dynamic-
        symmetric)")

    def apply_weights(self, layer, x, bias=None) -> torch.Tensor:
        w_q_name, w_s_name, _, _, _ = self.layer_param_names
        return torch.ops.zentorch.zentorch_dynamic_qlinear(
            x, getattr(layer, w_q_name), getattr(layer, w_s_name),
            bias, zentorch_op_name="zentorch::zentorch_dynamic_qlinear")
```

vllm/model_executor/kernels/linear/zentorch_utils.py

提供 has_zentorch_op 函数，统一平台检测和 op 可用性检查，被所有 zentorch 后端调用。

```
from __future__ import annotations
import torch
from vllm.platforms import current_platform

__all__ = ["has_zentorch_op"]

def has_zentorch_op(op_names: list[str]) -> bool:
    """返回当运行在 Zen CPU 且所有命名 op 已注册时为 ``True``。"""
    if not op_names:
        raise ValueError("has_zentorch_op requires at least one op name")
    # 先检查平台：仅 AMD Zen CPU 才继续
```

```
    if not current_platform.is_zen_cpu():
        return False
    # 检查 torch.ops.zentorch 命名空间
    ns = getattr(torch.ops, "zentorch", None)
    if ns is None:
        return False
    # 验证所有请求的 op 都已注册
    return all(hasattr(ns, op_name) for op_name in op_names)
```

评论区精华

W4A16 偏移量争议：Gemini Code Assist 认为 unpack 后 +8 导致数据损坏，贡献者解释 unpack_from_int32 输出 int4 范围 [-8,7]，而 zentorch 期望 uint4 [0,15]，偏移为必要转换，最终保留。 TorchAO 类型检查问题：Gemini 指出 isinstance(layer.weight, Int8Tensor) 因 Parameter 包装始终 False，贡献者测试确认实际路径正确；TorchAO 集成随后移出此 PR。 独立后端重构：mgoin 建议创建独立 zentorch.py 避免污染 cpu.py，团队采纳并实施。 测试条件统一：AndreasKaratzas 建议合并两个 skipif 装饰器，贡献者合并；SQNR 阈值从 30 dB 提升至 40 dB，参考 torchao 标准。 PlatformEnum 讨论：Andreas 建议引入 PlatformEnum.ZEN，团队同意作为后续 PR。

- W4A16 权重偏移量 +8 的正确性 (correctness): 保留偏移，贡献者验证通过。
- TorchAO 集成中 isinstance 检查失效 (correctness): TorchAO 集成移至后续 PR。
- 将 zentorch 逻辑拆分为独立后端 (design): 已创建独立后端文件。
- 测试跳过条件和 SQNR 阈值 (testing): 已更新测试。
- 未来引入 PlatformEnum.ZEN (design): 暂时保留在 CPU 下，后续 PR 引入 ZEN。

风险与影响

- 风险： 1) 权重布局假设风险：ZentorchWNA16LinearKernel 假设 packed weight 为 compressed-tensors GPTQ 格式（input_dim == packed_dim == 1），其他格式（如 Marlin）会被 _zentorch_woq_eligible 排除并回退，无数据损坏风险。 2) zentorch 依赖风险：未安装时自动回退，但用户可能期望加速而实际未使用；缺少日志提示。 3) PerTensor 激活量化升级：当模型使用 PerTensor 时，代码静默升级为 PerRow，输出品质可能变化，但不会出错。 4) 平台枚举耦合：PlatformEnum.CPU 同时包含 oneDNN 和 zentorch，未来引入 PlatformEnum.ZEN 可解耦。 5) 测试覆盖缺口：缺少端到端模型推理测试，仅单元测试覆盖 dispatch。
- 影响：
  - 用户影响：AMD Zen CPU 用户在运行 GPTQ（W4A16）或 LLM-Compressor（W8A8）量化模型时自动获得加速；其他平台无影响。
  - 系统影响：新增可选依赖 zentorch，非 Zen 平台无额外开销。
  - 团队影响：需维护独立后端，跟踪 zentorch 版本；kernel 选择器复杂度略有增加。
  - 风险标记：平台假设耦合，依赖条件性降级，布局假设有限，无端到端测试

关联脉络

- 暂无明显关联 PR