# PR #47205 完整报告

vllm-project/vllm

[Kernel][XPU] Tensor-descriptor operand loads for Triton W8A8 scaled_mm

合并时间：2026-08-10 08:56

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/47205

## 执行摘要

- 一句话：Triton W8A8 scaled_mm 引入 Tensor 描述符加载并启用 XPU 路径
- 推荐动作：值得精读。重点看 TD 与掩码加载的设计权衡、转置权重（checkpoint 存储为 w_q.t()）如何通过 B_T 区分并在 tile 内转置、以及 tri-state 环境变量（VLLM_TRITON_USE_TD）在各平台上的默认策略。这个模式对后续其他 kernel 的 TD 迁移有参考价值。

## 功能与动机

PR body 指出，压缩张量 W8A8 INT8 linear 路径此前未在 XPU 上启用；且 kernel 使用掩码加载时，每个元素都要做 64 位地址计算，绕过了 Xe XMX 的 2D 块加载路径。TD 可以在 K 循环外一次性描述张量布局，把寻址开销卸载给硬件。性能数据（吞吐 +139.7%、TPOT -57.6%）直接验证了动机。

## 实现拆解

1. 核心 kernel 支持 TD 加载：在 vllm/model_executor/layers/quantization/compressed_tensors/triton_scaled_mm.py 的 scaled_mm_kernel 中新增 USE_TD 和 B_T 两个 tl.constexpr。开启时通过 tl.make_tensor_descriptor 分别描述 A（[M,K]）和 B 的布局，B 区分两种方向：通常 [K,N]，而转换后的 checkpoint 权重为 [N,K]（strides=(1,K)），通过 B_T 分支描述 N 主序缓冲并在 tile 加载后 tl.trans 转置。K 循环内用 a_desc.load / b_desc.load 直接块加载，替代原来的 tl.load + mask 计算，避免 int64 偏移逐元素寻址。
2. TD 启用的 gate 与资源初始化：triton_scaled_mm 新增 use_td: bool | None 参数，通过共享的 use_tensor_descriptor(use_td) 解析全局 VLLM_TRITON_USE_TD（XPU 默认开启，其他平台需显式开启）。启用条件包括内维连续（input.stride(1)==1，权重 stride(1)==1 或 b_t）、16 字节 tile 对齐、tile 尺寸为 2 的幂。首次在某个设备启用 TD 时调用 set_triton_allocator 并记录到 _TD_ALLOCATOR_DEVICES。
3. XPU 平台接入：vllm/model_executor/kernels/linear/scaled_mm/triton.py 的 TritonInt8ScaledMMLinearKernel.is_supported 允许 XPU；vllm/model_executor/kernels/linear/__init__.py 在 _POSSIBLE_INT8_KERNELS 中为 PlatformEnum.XPU 添加 Triton kernel。
4. XPU int8 量化 fallback：vllm/_custom_ops.py 的 scaled_int8_quant 增加 XPU 分支：非对称量化直接抛 NotImplementedError；静态 per-tensor 用量化公式模拟；动态 per-token 复用 xpu_ops.dynamic_per_token_int8_quant_ref。

5. 测试配套：tests/kernels/quantization/test_triton_scaled_mm.py 新增 test_scaled_mm_td_matches_plain，在 CUDA-alike 和 XPU 上对比 use_td=True/False 的输出，要求位级一致（rtol=0, atol=0），并覆盖 M=1/64/256、K=N∈{4096,2048}、两种 scale_a 布局和是否含 bias 的组合。

关键文件：

- vllm/model_executor/layers/quantization/compressed_tensors/triton_scaled_mm.py（模块 量化内核；类别 source；类型 data-contract；符号 scaled_mm_kernel, triton_scaled_mm）：核心内核实现：为 scaled_mm_kernel 增加 USE_TD/B_T 分支，用 tl.make_tensor_descriptor 替代掩码加载，并在 triton_scaled_mm 中实现 TD 启用 gate 与分配器初始化。
- vllm/_custom_ops.py（模块 自定义算子；类别 source；类型 dependency-wiring；符号 scaled_int8_quant）：为缺少原生 _C 量化算子的 XPU 设备补充 scaled_int8_quant 的 fallback 路径，使 W8A8 推理链路在 XPU 上可完整运行。
- tests/kernels/quantization/test_triton_scaled_mm.py（模块 量化测试；类别 test；类型 test-coverage；符号 test_scaled_mm_td_matches_plain）：新增 TD 与普通路径的位级一致性测试，确保两种加载方式在多种形状、scale 布局和 bias 组合下输出完全一致。
- vllm/model_executor/kernels/linear/scaled_mm/triton.py（模块 线性内核；类别 source；类型 data-contract；符号 TritonInt8ScaledMMLinearKernel.is_supported）：TritonInt8ScaledMMLinearKernel.is_supported 扩展至 XPU，否则 kernel 选择阶段会拒绝在 XPU 上启用该 Triton 实现。
- vllm/model_executor/kernels/linear/__init__.py（模块 内核注册；类别 source；类型 data-contract；符号 _POSSIBLE_INT8_KERNELS）：在 _POSSIBLE_INT8_KERNELS 中为 XPU 平台注册 TritonInt8ScaledMMLinearKernel，使 XPU 推理时能通过 kernel 选择机制选中该实现。

关键符号：scaled_mm_kernel, triton_scaled_mm, scaled_int8_quant, TritonInt8ScaledMMLinearKernel.is_supported

## 关键源码片段

### vllm/model_executor/layers/quantization/compressed_tensors/triton_scaled_mm.py

核心内核实现：为 scaled_mm_kernel 增加 USE_TD/B_T 分支，用 tl.make_tensor_descriptor 替代掩码加载，并在 triton_scaled_mm 中实现 TD 启用 gate 与分配器初始化。

```
# 文件：vllm/model_executor/layers/quantization/compressed_tensors/triton_scaled_mm.py
# scaled_mm_kernel 中新增 USE_TD / B_T 两个 constexpr 分支。
# USE_TD 开启时，用 tl.make_tensor_descriptor 描述 A、B 的张量布局，
# 在 K 循环内直接 block load，避免每元素 int64 偏移计算。

if USE_TD:
    # 描述 A：shape [M, K]，stride [stride_am, 1]，
    # block_shape [BLOCK_SIZE_M, BLOCK_SIZE_K]
```

```
    a_desc = tl.make_tensor_descriptor(
        a_ptr,
        shape=[M, K],
        strides=[stride_am, 1],
        block_shape=[BLOCK_SIZE_M, BLOCK_SIZE_K],
    )
    if B_T:
        # checkpoint 权重以转置 [N, K] 视图存放（w_q.t()），
        # 这里描述底层 N 主序缓冲，并在每个 tile 加载后转置。
        b_desc = tl.make_tensor_descriptor(
            b_ptr,
            shape=[N, K],
            strides=[stride_bn, 1],
            block_shape=[BLOCK_SIZE_N, BLOCK_SIZE_K],
        )
    else:
        b_desc = tl.make_tensor_descriptor(
            b_ptr,
            shape=[K, N],
            strides=[stride_bk, 1],
            block_shape=[BLOCK_SIZE_K, BLOCK_SIZE_N],
        )

for k in range(0, tl.cdiv(K, BLOCK_SIZE_K)):
    if USE_TD:
        a = a_desc.load([pid_m * BLOCK_SIZE_M, k * BLOCK_SIZE_K])
        if B_T:
            b = tl.trans(b_desc.load([pid_n * BLOCK_SIZE_N, k * BLOCK_SIZE_K]))
        else:
            b = b_desc.load([k * BLOCK_SIZE_K, pid_n * BLOCK_SIZE_N])
    else:
        # 普通掩码加载路径（原有逻辑），mask 由 offsets_k 生成
        masks_k = offsets_k < K
        masks_a = masks_am[:, None] & masks_k[None, :]
        a = tl.load(a_ptrs, mask=masks_a)
        masks_b = masks_k[:, None] & masks_bn[None, :]
        b = tl.load(b_ptrs, mask=masks_b)
    accumulator = tl.dot(a, b, accumulator, out_dtype=accumulator_dtype)

# 外层 triton_scaled_mm 的 TD 启用判断。
# 条件：use_tensor_descriptor(use_td) 允许（tri-state 环境变量），
# 且内维连续、16 字节对齐、tile 为 2 的幂。
b_t = weight.stride(1) != 1 and weight.stride(0) == 1
b_inner = K if b_t else N
use_td = (
    use_tensor_descriptor(use_td)
    and input.stride(1) == 1
    and (weight.stride(1) == 1 or b_t)
    and (K * input.element_size()) % 16 == 0
```

```
        and (b_inner * weight.element_size()) % 16 == 0
        and (block_size_m & (block_size_m - 1)) == 0
        and (block_size_n & (block_size_n - 1)) == 0
        and (block_size_k & (block_size_k - 1)) == 0
    )
    # TD 需要专用分配器，每个设备只初始化一次。
    if use_td and input.device not in _TD_ALLOCATOR_DEVICES:
        set_triton_allocator(input.device)
        _TD_ALLOCATOR_DEVICES.add(input.device)
```

### vllm/_custom_ops.py

为缺少原生 _C 量化算子的 XPU 设备补充 scaled_int8_quant 的 fallback 路径，使 W8A8 推理链路在 XPU 上可完整运行。

```
    # 文件：vllm/_custom_ops.py
    # scaled_int8_quant 在 XPU 上走 torch.compile 参考实现，
    # 因为 XPU 没有对应的 _C 原生 int8 量化算子。
    if current_platform.is_xpu():
        # XPU 仅支持对称量化，asymmetric 直接报错，避免静默错误结果。
        if not symmetric:
            raise NotImplementedError(
                "asymmetric int8 activation quantization is unsupported on XPU"
            )
        # 静态 per-tensor 量化：用 float32 除法 + round + clamp 模拟。
        if scale is not None:
            q = (input.to(torch.float32) / scale).round().clamp(-128, 127)
            return q.to(torch.int8), scale, None

        # 动态 per-token 量化：复用 xpu_ops 的参考实现。
        from vllm._xpu_ops import xpu_ops
        q, scales, _ = xpu_ops.dynamic_per_token_int8_quant_ref(
            input.contiguous(), True, 8
        )
        return q, scales.reshape(-1, 1).to(torch.float32), None
```

## 评论区精华

该 PR 没有实质性的 review 评论：claude[bot] 因 fork PR 自动停用 review；维护者 jikunshang 直接 APPROVED。唯一值得注意的评论是 jikunshang 在 CI 中说明 xpu sleep model is fail on main. merge this.，即 XPU sleep model 测试在 main 上已失败，与本次改动无关，因此不阻塞合入。由于缺少代码级讨论，设计决策主要依赖 PR body 中给出的 microbenchmark 和准确率数据支撑。

- XPU 相关 CI 失败是否阻塞合入 (other): 确认 XPU sleep model 失败属于 main 已存在问题，不阻塞本 PR。

## 风险与影响

- 风险：TD 门控条件（内维连续、16 字节对齐、2 的幂 tile）若未覆盖某些合法布局会回退到普通路径，逻辑上是安全的；但若 b_t 检测不准确可能产生错误结果，测试仅覆盖有限形状。XPU 上新启用的 scaled_int8_quant fallback 依赖 torch.compile 参考实现，不支持 asymmetric 量化，遇到不对称权重量化会直接抛 NotImplementedError；且动态量化走 xpu_ops 参考路径，性能未充分验证。此外，XPU kernel 注册顺序将 TritonInt8ScaledMMLinearKernel 列为唯一候选，可能导致此前可用的其他 XPU int8 路径被替代，需确认无回归。
- 影响：对 XPU 用户：首次获得 compressed-tensors W8A8 INT8 模型支持，并默认启用 TD，显著提升吞吐、降低 TTFT/TPOT；对 CUDA/ROCm 用户：功能默认关闭，行为不变；对团队：新增一条平台专用量化内核路径，需要为 XPU 维护 TD 特性和 fallback 参考实现。改动集中在 kernel 层、量化层和线性内核注册，影响面有限但跨模块。
- 风险标记：新平台路径启用，TD 门控条件严格，XPU 量化 fallback 依赖 torch.compile 参考实现，测试覆盖形状有限

## 关联脉络

- PR #50949 [CPU] Optimize routed FP8/MXFP4 MoE GEMM dispatch: 同为量化 GEMM 内核性能优化，在 CPU 平台通过 BRGEMM 选择提升吞吐，与本 PR 在 XPU 上的优化形成跨平台对照。
- PR #51457 [Test] Add ROCm AITER FP8 MLA prefill accuracy test: 同为量化内核新增测试覆盖，验证新 kernel 路径的数值正确性，与本 PR 的位级一致性测试目标一致。
- PR #51458 [Perf] Avoid some more unnecessary GPU<->CPU syncs: 同属 kernel 层性能优化系列，体现 vLLM 在多种硬件平台上持续削减不必要开销的整体趋势。