PR #46340 完整报告

vllm-project/vllm

[Kernel] TD operand loads for batched MoE GEMM (moe_mmk) on XPU

合并时间：2026-07-26 08:50

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/46340

执行摘要

- 一句话：XPU batched MoE GEMM 新增 TD 加载路径，EP 场景吞吐翻倍
- 推荐动作：值得精读，尤其关注三处设计：TD 与 masked load 的性能差异机制及 gate 约束；三态开关的全局抽象（all-or-none 策略）；向量化 dispatch/combine 与 per-expert 循环的取舍。建议合并顺序上先合 #45781（统一开关）再合本 PR，并跟进 #46871 合入后的 EP 端到端验证；后续补充量化路径的 TD 测试。

功能与动机

PR body 指出：XPU 上带掩码的 tl.load 喂给 tl.dot 会绕过 Xe XMX 的二维块读路径，改用 tl.make_tensor_descriptor 加载两个操作数可恢复该路径，输出与 non-TD 路径 bit-identical。该 batched kernel 是 low-latency Expert-Parallel dispatch 实际使用的内核（batched activation format 为 [E_local, max_tokens, K]），其端到端效果必须在 EP 场景下衡量，而不是与单 GPU fused MoE 对比；PR 同时回应了此前 review 反馈——正确对比是 batched EP 路径上 TD on vs off。

实现拆解

1. 内核层 TD 路径：vllm/model_executor/layers/fused_moe/experts/fused_batched_moe.py 中的 moe_mmk 新增 a_base_ptr、b_base_ptr、M、N、stride_am、stride_bn 与 USE_TD 参数；USE_TD 为真时用 tl.make_tensor_descriptor 构造 A[M,K] 与 B[N,K] 描述符，K 循环内改用 a_desc.load / tl.trans(b_desc.load) 取块，替代带 mask 的 tl.load。expert_triton_kernel 与 batched_triton_kernel 负责透传基址、形状和 USE_TD。launch 侧 invoke_moe_batched_triton_kernel 计算 use_td：要求全局开关开启，且 A/B 末维 K 连续、K×element_size 16 字节对齐、BLOCK_M/N/K 均为 2 的幂；满足时调用 set_triton_allocator 并传入 USE_TD=True。原因：TD 加载能让 XPU 恢复 Xe XMX 二维块读路径，同时保持输出 bit-exact；这些约束来自 make_tensor_descriptor 要求 K stride 为 1 且块形状对齐。
2. 统一开关：新增 vllm/triton_utils/tensor_descriptor.py，提供 use_tensor_descriptor() 三态逻辑——VLLM_TRITON_USE_TD 未设置时自动（当前仅 XPU 开启），显式 1/0 强制开启 / 关闭；并在 vllm/triton_utils/__init__.py 导出。该开关独立于 MoE 后端选择，未来其他 Triton 内核可复用（对应 review 中“TD 使用应 all-or-none”的决策）。
3. 后端选择与激活格式：vllm/config/kernel.py 的 MoEBackend 字面量新增 batched_triton；vllm/model_executor/layers/fused_moe/oracle/unquantized.py 的

map_unquantized_backend 将其映射到 UnquantizedMoeBackend.BATCHED_TRITON，并在 select_unquantized_moe_backend 中把该后端强制切换到 BatchedExperts 激活格式；all2all_utils.py 的 maybe_make_prepare_finalize 在 current_platform.is_xpu() andmoe.moe_backend == "batched_triton" 时返回 BatchedPrepareAndFinalize（无 all-to-all 的本地重组）。用户只需 --moe-backend batched_triton，无需使用环境变量。

4. 向量化 dispatch/combine：prepare_finalize/batched.py 的 BatchedPrepareAndFinalize.prepare 在未量化分支用 [E_local, T] 命中掩码 + cumsum 一次算出各 expert 的 token 数与 slot 并写入 b_a1，替代逐 expert 循环；topk_weight_and_reduce.py 的 TopKWeightAndReduceNaiveBatched.apply 用相同掩码 + index_add_ 做加权 scatter-add。目的：减少 per-expert 循环带来的多次 kernel launch 与 CPU-GPU 同步，与 TD 内核配套；量化分支保留逐 expert 循环（因需要逐 token 量化）。
5. 平台兼容与测试配套：新增 _is_capturing_or_compiling() 统一 torch.compiler.is_compiling() 与 cudagraph 流捕获判断（XPU 的 torch 没有 is_current_stream_capturing），替换 apply 与 batched_moe_kernel_quantize_input 中的旧判断；BatchedTritonExperts._supports_current_device 放开 XPU。测试侧，tests/kernels/moe/test_batched_moe.py 新增 _td_supported / _run_td 辅助与 6 个用例：TD vs plain bit-exact（3 组形状）、零 expert token、设备启用、端到端参考实现对比、后端映射。

关键文件：

- vllm/model_executor/layers/fused_moe/experts/fused_batched_moe.py（模块 MoE 内核；类别 source；类型 core-logic；符号 moe_mmk, expert_triton_kernel, batched_triton_kernel, invoke_moe_batched_triton_kernel）：核心内核变更：moe_mmk 新增 TD 操作数加载分支，batched_triton_kernel / expert_triton_kernel / invoke_moe_batched_triton_kernel 串联 USE_TD 门控，并引入 XPU 安全的 _is_capturing_or_compiling()。
- tests/kernels/moe/test_batched_moe.py（模块 MoE 测试；类别 test；类型 test-coverage；符号 _td_supported, _run_td, test_batched_mm_td_matches_plain, test_batched_mm_td_zero_expert_tokens）：测试配套：新增 TD vs plain bit-exact 对比、零 expert token 处理、设备启用、端到端参考实现与后端映射测试，是 TD 正确性的主要保障。
- vllm/triton_utils/tensor_descriptor.py（模块 TD 开关；类别 source；类型 core-logic；符号 use_tensor_descriptor）：新增统一三态 TD 开关 use_tensor_descriptor()，XPU 自动开启、CUDA sm90+ 可 opt-in，是全局配置抽象的核心。
- vllm/model_executor/layers/fused_moe/prepare_finalize/batched.py（模块 分派合并；类别 source；类型 performance；符号 BatchedPrepareAndFinalize.prepare）：BatchedPrepareAndFinalize.prepare 未量化路径向量化 dispatch，一次性完成 expert 命中与 slot 分配，配套 TD 内核降低 launch 开销。
- vllm/model_executor/layers/fused_moe/topk_weight_and_reduce.py（模块 权重归约；类别 source；类型 performance；符号 TopKWeightAndReduceNaiveBatched.apply）：TopKWeightAndReduceNaiveBatched.apply 向量化 weighted scatter-add，与 prepare

的向量化对称，降低 combine 开销。

- vllm/model_executor/layers/fused_moe/all2all_utils.py（模块 后端路由；类别 source；类型 core-logic；符号 maybe_make_prepare_finalize）：maybe_make_prepare_finalize 在 XPU + batched_triton 时返回 BatchedPrepareAndFinalize，打通 opt-in 后端选择路径。
- vllm/model_executor/layers/fused_moe/oracle/unquantized.py（模块 后端映射；类别 source；类型 core-logic；符号 map_unquantized_backend, select_unquantized_moe_backend）：map_unquantized_backend 与 select_unquantized_moe_backend 新增 batched_triton 映射与 BatchedExperts 激活格式选择。
- vllm/config/kernel.py（模块 内核配置；类别 config；类型 configuration；符号 MoEBackend）：MoEBackend 字面量新增 batched_triton 选项及文档，是用户入口。
- vllm/triton_utils/__init__.py（模块 工具导出；类别 source；类型 dependency-wiring）：导出 use_tensor_descriptor，使内核模块与外部可统一引用 TD 开关。

关键符号：use_tensor_descriptor, _is_capturing_or_compiling, moe_mmk, expert_triton_kernel, batched_triton_kernel, invoke_moe_batched_triton_kernel, BatchedTritonExperts._supports_current_device, BatchedPrepareAndFinalize.prepare, TopKWeightAndReduceNaiveBatched.apply, maybe_make_prepare_finalize, map_unquantized_backend, select_unquantized_moe_backend

关键源码片段

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

核心内核变更：moe_mmk 新增 TD 操作数加载分支，batched_triton_kernel / expert_triton_kernel / invoke_moe_batched_triton_kernel 串联 USE_TD 门控，并引入 XPU 安全的 _is_capturing_or_compiling()。

```
# moe_mmk：batched MoE expert GEMM 的核心矩阵乘内核。
# USE_TD 开启时改用 Tensor Descriptor 加载 A/B，XPU 上可恢复 Xe XMX
# 2D 块读路径，输出与 masked tl.load 路径 bit-exact。
@triton.jit
def moe_mmk(
    a_ptrs, b_ptrs, K, expert_id, a_scale_ptr, b_scale_ptr,
    stride_ak, stride_bk, stride_ase, stride_asm, stride_ask,
    stride_bse, stride_bsk, stride_bsn, offs_m, offs_n, offs_bn, mask_m,
    group_n, group_k, BLOCK_M, BLOCK_N, BLOCK_K, compute_type,
    use_w8a8, use_w8a16, per_act_token_quant,
    # TD 路径：a_base_ptr / b_base_ptr 是 expert 与 CTA 偏移后的 A[M,K] / B[N,K] 基址
    a_base_ptr=None,
    b_base_ptr=None,
    M=0,
    N=0,
    stride_am: tl.int64 = 0,
    stride_bn: tl.int64 = 0,
    USE_TD: tl.constexpr = False,
):
```

```
    offs_k = tl.arange(0, BLOCK_K)

    if USE_TD:
        # make_tensor_descriptor 要求最后一个维度（K）的 stride 是编译期常量 1，
        # launch 侧的 use_td 只在 A/B K 连续时置位。
        a_desc = tl.make_tensor_descriptor(
            a_base_ptr,
            shape=[M, K],
            strides=[stride_am, 1],
            block_shape=[BLOCK_M, BLOCK_K],
        )
        b_desc = tl.make_tensor_descriptor(
            b_base_ptr,
            shape=[N, K],
            strides=[stride_bn, 1],
            block_shape=[BLOCK_N, BLOCK_K],
        )

    accumulator = tl.zeros((BLOCK_M, BLOCK_N), dtype=tl.float32)
    for k in range(0, tl.cdiv(K, BLOCK_K)):
        if USE_TD:
            # B 布局为 [N, K]，取出的 tile 是 [BLOCK_N, BLOCK_K]，需转置后喂给 dot
            a = a_desc.load([0, k * BLOCK_K])
            b = tl.trans(b_desc.load([0, k * BLOCK_K]))
        else:
            # 原始路径：带 mask 的 tl.load，XPU 上会绕过 XMX 2D 块读
            a = tl.load(
                a_ptrs,
                mask=mask_m[:, None] & (offs_k[None, :] < K - k * BLOCK_K),
                other=0.0,
            )
            b = tl.load(b_ptrs, mask=offs_k[:, None] < K - k * BLOCK_K, other=0.0)
        # 量化分支（w8a8 / w8a16）省略：TD 只改变操作数加载方式，
        # 后续累加与 scale 逻辑对两条路径完全一致。
        accumulator += tl.dot(a, b)
        a_ptrs += BLOCK_K * stride_ak
        b_ptrs += BLOCK_K * stride_bk

# launch 侧决定是否启用 TD：全局开关 + 内存布局约束。
# A/B 必须 K 连续（stride == 1），K 上 16 字节对齐，BLOCK 均为 2 的幂。
use_td = (
    use_tensor_descriptor()
    and A.stride(2) == 1
    and B.stride(2) == 1
    and (K * A.element_size()) % 16 == 0
    and (BLOCK_M & (BLOCK_M - 1)) == 0
    and (BLOCK_N & (BLOCK_N - 1)) == 0
    and (BLOCK_K & (BLOCK_K - 1)) == 0
)
```

```
if use_td:
    # TD 需要在绑定的 Triton 分配器下工作
    set_triton_allocator(A.device)
batched_triton_kernel[grid](..., USE_TD=use_td)
```

vllm/triton_utils/tensor_descriptor.py

新增统一三态 TD 开关 use_tensor_descriptor()，XPU 自动开启、CUDA sm90+ 可 opt-in，是全局配置抽象的核心。

```
def use_tensor_descriptor(override: bool | None = None) -> bool:
    """三态 VLLM_TRITON_USE_TD：未设置 = 自动（XPU 开启），1/0 = 强制开关。"""
    from vllm import envs
    from vllm.platforms import current_platform

    if override is None:
        override = envs.VLLM_TRITON_USE_TD
    if override is not None:
        return override
    # 默认策略：XPU 上 TD 是恢复 XMX 读路径的关键，自动开启；
    # CUDA 上保持关闭，由用户显式 opt-in（sm90+ 才有收益）。
    return current_platform.is_xpu()
```

vllm/model_executor/layers/fused_moe/prepare_finalize/batched.py

BatchedPrepareAndFinalize.prepare 未量化路径向量化 dispatch，一次性完成 expert 命中与 slot 分配，配套 TD 内核降低 launch 开销。

```
# BatchedPrepareAndFinalize.prepare 的未量化路径：向量化 dispatch。
# 一次构建 [E_local, T] hit mask，并用 cumsum 计算每个 token 在所属
# expert 批内的 slot，避免 per-expert Python 循环的多次 kernel launch。
if quant_config.quant_dtype is None:
    local_ids = torch.arange(first_expert, last_expert, device=a1.device).view(-1, 1, 1)
    # hits 为 [E_local, T]，记录 token 是否命中本地 expert
    hits = (topk_ids.unsqueeze(0) == local_ids).any(dim=2) # [E_local, T]
    tokens_per_expert[:num_local_experts] = hits.sum(dim=1).to(torch.int32)
    # slot 保持 token 在 expert 批内的原始顺序
    slots = hits.to(torch.int32).cumsum(dim=1) - 1 # [E_local, T]
    e_idx, t_idx = hits.nonzero(as_tuple=True)
    b_a1[e_idx, slots[e_idx, t_idx]] = a1[t_idx].to(b_type)
else:
    # 量化路径保留逐 expert 循环（需要逐 token 量化与 scale 写入）
    for expert_id in range(first_expert, last_expert):
        ...
```

评论区精华

核心争论围绕两个层面：一是性能证据的可信度与对比口径，二是配置 / 开关的抽象层级。yewentao256 先以单卡数据质疑“不值得引入复杂度”，作者澄清单卡对比误导后补充了 2× Arc B70 与 4× H200 的 EP 路径数据；jikunshang 推动把 TD 开关抽成全局 Triton 配置（TD 应 all-or-none）、用 --moe-backend 替代环境变量，作者分别在 a9e2b52 与 822a813

采纳；quinnlp 提出 output store 是否也 TD 化，作者用微基准说明在常用 block_m=64 下 store-TD 反而变慢且需要独立 gate，不值得。

- 单 GPU 基准误导与 EP 场景定位 (design): 补充 2× Arc B70 与 4× H200 的 EP 路径数据（+116% 吞吐、−58% TPOT、+72% req/s），疑虑解除。
- TD 开关应抽成全局 Triton 配置 (design): 作者在 a9e2b52 移到 vllm/triton_utils/tensor_descriptor.py。
- 后端选择方式：env var vs --moe-backend (design): 作者在 822a813 改为 --moe-backend batched_triton，并同步 oracle 映射与 activation format 选择。
- output store 是否也 TD 化 (performance): 维持 plain masked store，不引入 TD store。
- 性能与精度证据完整性 (testing): 作者附上 COMMANDS.md、多份 serve/lm_eval 日志，并补充 CUDA 侧 H200 报告。

风险与影响

- 风险：1）内核正确性：TD 仅在满足 K 连续、16 字节对齐、BLOCK 为 2 的幂时才启用，若未来新调用路径不满足约束会静默回退 plain，行为依赖 gate 完整性；当前单测仅覆盖 bf16 未量化，w8a8/w8a16 量化与 TD 的组合无测试覆盖。2）平台行为：_is_capturing_or_compiling() 替换了直接调用 torch.cuda.is_current_stream_capturing() 的旧代码，对 XPU 更安全，但 cudagraph / torch.compile 路径需回归验证。3）数值一致性：向量化 combine 改用 index_add_ 后浮点累加顺序变化，topk>1 时与旧循环结果存在微小差异（单测容差 3e-2）。4）依赖顺序：TD 开关与 EP 后端分别来自 #45781、#46871，若单独合入本 PR，默认配置下新路径不会被执行。5）性能误用：单 GPU 上 batched 路径明显慢于 fused，用户误开 --moe-backend batched_triton 会回退性能，需要文档明确适用场景。
- 影响：用户影响：XPU 用户获得新 --moe-backend batched_triton 选项与自动 TD；在 low-latency EP 部署（配合 #46871）中吞吐可翻倍、TPOT 减半；CUDA sm90+ 用户可经 VLLM_TRITON_USE_TD=1 获得 DeepEP 场景下每内核 1.3–6.1 倍加速。系统影响：新增全局 TD 开关成为未来 Triton 内核的公共配置；batched activation format 在非 EP 配置下也可用，MoE oracle/config 需要维护新后端分支。团队影响：为 XPU 多卡 EP 铺路，但该 PR 单独合入时收益不可见，需要与 #46871 配合评审。
- 风险标记：核心内核路径变更，量化路径缺测试覆盖，依赖未合入 PR（45781/46871），浮点累加顺序变化

关联脉络

- PR #46871 portable low-latency all-to-all EP backend: PR body 明确本 PR 'Stacks under the portable low-latency all-to-all EP backend (#46871)'，该后端才是真正调用 batched MoE 内核的部署场景。
- PR #45781 unified VLLM_TRITON_USE_TD toggle: 评论中说明本 PR 依赖 #45781 引入的统一 VLLM_TRITON_USE_TD 开关，应在其之后合并。