PR #37835 完整报告

vllm-project/vllm

[ROCm] Add UE8M0 scale packing for Triton silu_mul_quant

合并时间：2026-08-19 23:20

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/37835

执行摘要

- 一句话：ROCm Triton 量化路径新增 UE8M0 scale 打包支持
- 推荐动作：值得精读：一是“float32 中间计算 + 位运算打包 UE8M0”的两阶段思路，对理解 UE8M0 编码（float32 指数位提取、4 字节对齐、strided copy）很有价值；二是 Triton/C++ 双路径分叉的设备守卫模式，可作为平台差异化 kernel 实现的参考。对 ROCm/XPU 平台维护者尤其建议细看。评审中关于 view 保持维度与 copy_ 非连续语义的交锋也值得留意，作者对 tensor 内存布局的理解是正确的。

功能与动机

PR body 明确说明：为 ROCm 上的 Triton _silu_mul_fp8_quant_deep_gemm fallback kernel 增加 UE8M0（int32 packed）scale 输出支持，目标是消除 ROCm 与 CUDA 在 FP8 量化 MoE 路径上的能力差距。原文指出 "This is effectively dead code on ROCm today (UE8M0 requires DeepGEMM which is CUDA-only), but ensures feature parity and removes a test skip"——即该路径在 ROCm 上目前实际是死代码，但补齐后可保证特征对齐并移除测试跳过。此前 Triton fallback 断言只支持 float32 scale，测试中 UE8M0 也被显式限制在 CUDA 平台。

实现拆解

实现按 4 步拆解：

1. 路径分叉与临时缓冲分配：vllm/model_executor/layers/fused_moe/experts/batched_deep_gemm_moe.py 中的 persistent_masked_m_silu_mul_quant 先根据 quant_scale_fmt 判断 is_packed_ue8m0（仅 DeepGemmQuantScaleFMT.UE8M0 为真）。当需要打包时，用 FLOAT32_CEIL_UE8M0 的 shape/stride（scales_shape_stride_dtype）额外分配一个 torch.float32 临时 scale 张量 y_s_f32；否则把 y_s_f32 直接指向 y_s，保证非 UE8M0 路径行为完全不变。原先“Triton fallback 仅支持 float32 scale”的 assert 被改为只对非 packed 场景生效。
2. Triton kernel 调用改造：Triton kernel _silu_mul_fp8_quant_deep_gemm 统一接收 y_s_f32 作为输出，strides 参数从 f32_strides 获取；ceil_ue8m0 标志原本就同时覆盖 FLOAT32_CEIL_UE8M0 与 UE8M0，保证 kernel 内对 scale 做向上取整后再输出 float32 值。

3. UE8M0 后处理打包：Kernel 返回后，若 is_packed_ue8m0，将 float32 位模式按 torch.int32 右移 23 位提取指数（bits 30:23），转成 uint8；若 G 不是 4 的倍数则用 torch.nn.functional.pad 补齐；4 个 uint8 再 view 成 int32，得到 (E, T, G//4) 的 packed scale。目标 y_s 是特殊 stride 的空张量，因此按 tokens_per_expert 逐 expert 用 copy_ 写入有效 token 行（零 token 行保持 y_s.zero_() 的零值）。
4. 测试配套：tests/kernels/moe/test_silu_mul_fp8_quant_deep_gemm.py 将 DeepGemmQuantScaleFMT.UE8M0 直接加入所有平台的 scale_fmts，删除 current_platform.is_cuda() 的跳过逻辑，使 ROCm/XPU 也运行 UE8M0 用例并与 ref_with_scale_fmt 浮点参考对照；DeepGEMM 可用且为 sm100 时仍额外做 deepgemm scales transform 校验。

关键文件：

- vllm/model_executor/layers/fused_moe/experts/batched_deep_gemm_moe.py（模块 专家层；类别 source；类型 data-contract；符号 persistent_masked_m_silu_mul_quant, _silu_mul_fp8_quant_deep_gemm）：PR 的核心实现所在：为 ROCm/XPU 的 Triton fallback 增加 UE8M0（int32 packed）scale 的临时 float32 缓冲、kernel 输出重定向与指数位打包逻辑，并调整 dtype 断言为仅对非 packed 场景生效。
- tests/kernels/moe/test_silu_mul_fp8_quant_deep_gemm.py（模块 内核测试；类别 test；类型 test-coverage；符号 test_silu_mul_fp8_quant_deep_gemm）：测试配套改动：把 UE8M0 从 CUDA-only 条件中移除，让 ROCm/XPU 平台也运行 UE8M0 格式用例，验证打包逻辑与浮点参考实现的一致性。

关键符号：persistent_masked_m_silu_mul_quant, _silu_mul_fp8_quant_deep_gemm, test_silu_mul_fp8_quant_deep_gemm

关键源码片段

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

PR 的核心实现所在：为 ROCm/XPU 的 Triton fallback 增加 UE8M0（int32 packed）scale 的临时 float32 缓冲、kernel 输出重定向与指数位打包逻辑，并调整 dtype 断言为仅对非 packed 场景生效。

以下片段为 persistent_masked_m_silu_mul_quant 中基于 head 版本整理的 Triton fallback 分支与 UE8M0 打包逻辑（函数前段已按 quant_scale_fmt 分配好 y_q 与 strided 的 y_s）：

```
# UE8M0 与 FLOAT32_CEIL_UE8M0 都需要 ceil 行为，CUDA C++ 与 Triton 共用该标志
ceil_ue8m0 = quant_scale_fmt in [
    DeepGemmQuantScaleFMT.FLOAT32_CEIL_UE8M0,
    DeepGemmQuantScaleFMT.UE8M0,
]

# CUDA sm80+ 走 C++ kernel；ROCm 与 XPU 走 Triton fallback（#ifndef USE_ROCM 守卫）
if current_platform.is_cuda() and current_platform.has_device_capability(80):
    torch.ops._C.persistent_masked_m_silu_mul_quant(
        y, tokens_per_expert, y_q, y_s, ceil_ue8m0
```

```
    )
else:
    # UE8M0 是 int32 packed scale，Triton kernel 只能输出 float32，
    # 因此先分配一份 float32 临时 scale，计算完成后再打包
    is_packed_ue8m0 = quant_scale_fmt == DeepGemmQuantScaleFMT.UE8M0
    if is_packed_ue8m0:
        f32_shape, f32_strides, _ = scales_shape_stride_dtype(
            E, T, G, DeepGemmQuantScaleFMT.FLOAT32_CEIL_UE8M0
        )
        y_s_f32 = torch.empty_strided(
            f32_shape, f32_strides, dtype=torch.float32, device=y.device
        )
    else:
        y_s_f32 = y_s

    stride_cnt_e = tokens_per_expert.stride()[0]
    grid = (E * G,)
    stride_i_e, stride_i_t, stride_i_h = y.stride()
    stride_yq_e, stride_yq_t, stride_yq_h = y_q.stride()
    fp8_min, fp8_max = get_fp8_min_max()
    eps = 1e-10

    # 只有非 packed 场景才要求 y_s 直接是 float32；UE8M0 场景允许 y_s 为 int32
    if not is_packed_ue8m0:
        assert y_s.dtype == torch.float32, (
            "_silu_mul_fp8_quant_deep_gemm Triton fallback does not "
            f"support {y_s.dtype} scales. Only torch.float32 supported."
        )
    f32_strides = y_s_f32.stride()

    _silu_mul_fp8_quant_deep_gemm[grid](
        y, y_q, y_s_f32, tokens_per_expert, H, group_size,
        stride_i_e, stride_i_t, stride_i_h,
        stride_yq_e, stride_yq_t, stride_yq_h,
        f32_strides[0], f32_strides[1], f32_strides[2],
        stride_cnt_e, eps, fp8_min, fp8_max, ceil_ue8m0,
        BLOCK=group_size, NUM_STAGES=4, num_warps=1,
    )

    if is_packed_ue8m0:
        # 打包：提取 float32 指数位（bits 30:23）作为 UE8M0 的 uint8 值
        E_dim, T_dim, G_dim = y_s_f32.shape
        y_s_cont = y_s_f32.contiguous()
        i32_pad = round_up(G_dim, 4) - G_dim
        y_s_u8 = (y_s_cont.view(torch.int32) >> 23).to(torch.uint8)
        if i32_pad > 0:
            y_s_u8 = torch.nn.functional.pad(y_s_u8, (0, i32_pad)) # 补齐到 4 字节对齐
        # 4 个 uint8 位模式拼成一个 int32；目标 y_s 形状为 (E, T, G//4)
        packed = y_s_u8.view(torch.int32)
```

```
        # y_s 是带特殊 stride 的空张量（stride 为 (T*G//4, 1, T)），逐 expert 用 copy_ 写入
        for e_idx in range(E_dim):
            nt = tokens_per_expert[e_idx].item()
            if nt > 0:
                y_s[e_idx, :nt].copy_(packed[e_idx, :nt])

    return y_q, y_s
```

tests/kernels/moe/test_silu_mul_fp8_quant_deep_gemm.py

测试配套改动：把 UE8M0 从 CUDA-only 条件中移除，让 ROCm/XPU 平台也运行 UE8M0 格式用例，验证打包逻辑与浮点参考实现的一致性。

测试中 scale 格式列表的改动（基于 head 版本整理）：

```
# 三种 scale 格式现在对所有平台统一覆盖
scale_fmts = [
    DeepGemmQuantScaleFMT.FLOAT32,
    DeepGemmQuantScaleFMT.FLOAT32_CEIL_UE8M0,
    # UE8M0 在 ROCm/XPU 的 Triton fallback 上已可用，无需再限制 CUDA
    DeepGemmQuantScaleFMT.UE8M0,
]

# 每种格式都跑一次 kernel，并与浮点参考实现对照
for scale_fmt in scale_fmts:
    y_q, y_s = persistent_masked_m_silu_mul_quant(
        y, tokens_per_expert, group_size=group_size, quant_scale_fmt=scale_fmt,
    )

    ref_y_q, ref_y_s = ref_with_scale_fmt(
        E, T, H, group_size, tokens_per_expert, gate, up, scale_fmt=scale_fmt,
    )
```

评论区精华

核心讨论围绕 UE8M0 打包代码的正确性展开：gemini-code-assist[bot] 提出 critical 级别问题，称 view(torch.int32) 会把 3D 张量变成 1D，导致 packed[e_idx, :nt] 触发 IndexError，且 y_s[e_idx, :nt] 是非连续 slice，copy_ 会失败；作者 AndreasKaratzas 反驳称 3D uint8 张量 view(torch.int32) 仍保持 3D（仅末维除以 4），copy_ 本身就支持非连续目标，不存在 bug。最终评审未再质疑，tjtanaa 给出 APPROVED 并指出这是 enablement PR，建议后续 PR 处理 custom logic。

- UE8M0 打包逻辑的 view/copy_ 正确性 (correctness): 作者澄清 PyTorch view 语义与 copy_ 行为后，评审未继续质疑；tjtanaa 最终 APPROVED，代码保持原实现合并。

风险与影响

- 风险：风险集中在 3 点：一是位运算打包逻辑的正确性依赖对 PyTorch view 语义和小端字节序的假设，一旦测试未覆盖到非 4 对齐的 G、零 token expert 等边界情况，可能出现静默错误，当前靠 ref_with_scale_fmt 对照测试兜底；二是新增了 float32 临时张量分配和

按 expert 的 Python 循环 copy_，在 token/expert 数量大时有额外启动与拷贝开销，不过该路径本就是 fallback 且当前为死代码；三是 XPU 与 ROCm 共用 Triton fallback，UE8M0 在 XPU 上此前从未被测试覆盖，行为变化需要关注 XPU CI 结果。CUDA sm80+ 路径不受影响（仍走 C++ kernel）。

- 影响：对 ROCm 用户，启用相应 FP8 量化 MoE 与 UE8M0 格式时 Triton fallback 不再因 unsupported dtype 报错，为后续 ROCm 上真实启用该格式铺路；对 XPU 用户，共用 fallback 的行为随之变化，测试覆盖同步扩大。当前 UE8M0 在 ROCm 上仍属 enablement 性质，实际推理收益需 DeepGEMM 或等效后端支持后才能体现。对团队而言，本 PR 减少了一个测试跳过点，延续了 vLLM 对非 NVIDIA 平台 kernel 覆盖补齐的策略，回归保护增强。
- 风险标记：位运算打包正确性依赖测试兜底，逐 expert Python 循环拷贝开销，XPU 共用 Triton 路径需关注，enablement 性质需后续跟进

关联脉络

- PR #37833 （依赖 PR，标题未在材料中提供）：PR body 明确声明 Depends on #37833，本 PR 的 UE8M0 scale 支持建立在该前置 PR 的基础之上。
- PR #52775 [Kernel] SM120: stop routing misaligned-M blockwise FP8 GEMMs to the small-M swapAB config: 同属 FP8 量化 GEMM kernel 路径的优化，但面向 NVIDIA SM120 的 C++/CUTLASS 路径；本 PR 面向 ROCm/XPU 的 Triton fallback 路径，两者在量化 kernel 覆盖策略上互补。