PR #42988 完整报告

vllm-project/vllm

[Perf] `zeros` -> `empty` to remove additional fill

合并时间：2026-05-22 04:00

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/42988

执行摘要

- 一句话：将 torch.zeros 替换为 torch.empty，消除额外清零开销
- 推荐动作：此 PR 是一次小范围、低风险、有明显理论收益的微观优化。源码变更模式（zeros→empty）是一种普遍适用的手法，值得在 vllm 其他类似场景（如其他注意力机制、量化路径中的输出缓冲区）中推广。建议精读 _custom_ops.py 与 fp8_utils.py 的完整代码片段，理解"先分配后覆写"的模式识别方法。

功能与动机

量化内核和 attention 路径中部分输出张量仅作为写入缓冲区使用，后续立即被内核或赋值操作完全覆写。使用 torch.zeros（而非 torch.empty）会额外执行 GPU 填零操作，产生不必要的内存带宽消耗。开发者通过将 zeros 切换为 empty 来消除这部分开销。

实现拆解

变更分为三组：

1. FP4 量化（vllm/_custom_ops.py）：create_fp4_scale_tensor 的两个分支（swizzled 布局与非 swizzled 布局）均将 torch.zeros 替换为 torch.empty。由于 create_fp4_output_tensors 的调用者和 _scaled_fp4_quant_fake 立即将返回值传入 C++ 内核或 fake 张量，内核随后会写入所有 scale 值，因此无需预清零。
2. FP8 量化（vllm/model_executor/layers/quantization/utils/fp8_utils.py）：silu_mul_quant_fp8_packed_triton 中的 output_scale_packed 分配从 zeros 改为 empty。该张量随后由 Triton 内核 _silu_mul_quant_fp8_packed_kernel 完全写入，清零冗余。
3. TurboQuant 混合批次 attention（vllm/v1/attention/backends/turboquant_attn.py）：混合批次路径中的 attn_out 分配从 zeros 改为 empty。后续代码通过 _decode_attention 和 _prefill_attention 逐步填充 attn_out 的各个部分，每次写入互不重叠，因此无需清零。
4. GDN attention ROCm 路径（vllm/model_executor/layers/mamba/gdn_linear_attn.py）：删除一行 z_out.zero_()，因为紧接着的 z_out[:] = z 会立即覆写整个张量，使预清零变为无用操作。

关键文件：

- vllm/model_executor/layers/quantization/utils/fp8_utils.py（模块 量化工具；类别 source；类型 data-contract）：FP8 packed scale 分配从 torch.zeros 切换到 torch.empty，消除 Triton 内核写入前的清零开销。

- vllm/_custom_ops.py（模块 自定义算子；类别 source；类型 core-logic）：FP4 scale 张量分配（swizzled 与普通布局）从 torch.zeros 切换到 torch.empty，是最早的修改点之一。
- vllm/v1/attention/backends/turboquant_attn.py（模块 注意力后端；类别 source；类型 core-logic）：混合批次路径中 attn_out 分配从 torch.zeros 切换到 torch.empty，消除整张量清零，因为后续 slice 写入将会完全覆写。
- vllm/model_executor/layers/mamba/gdn_linear_attn.py（模块 Mamba 层；类别 source；类型 data-contract）：删除 ROCm 路径中冗余的 z_out.zero_()，因为它立即被 z_out[:] = z 覆写。

关键符号：create_fp4_scale_tensor, silu_mul_quant_fp8_packed_triton, forward (TurboQuantAttention backend), _forward_core_rocm (GDN attention)

关键源码片段

vllm/model_executor/layers/quantization/utils/fp8_utils.py

FP8 packed scale 分配从 torch.zeros 切换到 torch.empty，消除 Triton 内核写入前的清零开销。

```
# silu_mul_quant_fp8_packed_triton 的 scale 分配 (vllm/model_executor/layers/quantization/utils/fp8_utils.py)

    if output_q is None:
        output_q = torch.empty((M, N_2), dtype=fp8_dtype, device=input.device)

    # 关键变更：torch.zeros → torch.empty
    # 原因：Triton 内核 _silu_mul_quant_fp8_packed_kernel 会完全写入 output_scale_packed
    # 预先填零是不必要的，empty 跳过清零，减少内存带宽消耗
    output_scale_packed = torch.empty(
        (num_packed_groups, tma_aligned_M),
        dtype=torch.int32,
        device=input.device,
    ).T[:M, :]

    # ... 后续 grid launch 中内核直接写入该 buffer
```

vllm/_custom_ops.py

FP4 scale 张量分配（swizzled 与普通布局）从 torch.zeros 切换到 torch.empty，是最早的修改点之一。

```
# vllm/_custom_ops.py - FP4 scale 张量分配

def create_fp4_scale_tensor(
    m: int,
    n: int,
    device: torch.device,
    is_sf_swizzled_layout: bool,
) -> torch.Tensor:
    block_size = 16
```

```
    if is_sf_swizzled_layout:
        rounded_m = round_up(m, 128)
        scale_n = n // block_size
        rounded_n = round_up(scale_n, 4)
        # 原为 torch.zeros，现改为 torch.empty
        # 原因：C++ scaled_fp4_quant 内核会完全写入 scale 值
        return torch.empty(
            (rounded_m, rounded_n // 4), device=device, dtype=torch.int32
        )
    else:
        # 同上，非 swizzled 布局也无需预清零
        return torch.empty((m, n // block_size), device=device, dtype=torch.uint8)
```

vllm/v1/attention/backends/turboquant_attn.py

混合批次路径中 attn_out 分配从 torch.zeros 切换到 torch.empty，消除整张量清零，因为后续 slice 写入将会完全覆写。

```
# vllm/v1/attention/backends/turboquant_attn.py - 混合批次路径

        else:
            # Mixed batch: decodes first (guaranteed by reorder_batch).
            # 原为 torch.zeros，现改为 torch.empty
            # 原因：attn_out[:num_decode_tokens] 经 _decode_attention 写入
            # 剩余部分由 _prefill_attention 写入，完全覆盖，无需清零
            attn_out = torch.empty(
                N, self.num_heads, self.head_size, device=device, dtype=q.dtype
            )

            # --- Decode portion (first num_decodes requests) ---
            # ... _decode_attention 写入 attn_out[:num_decode_tokens]
            # --- Prefill portion (remaining requests) ---
            # ... _prefill_attention 写入 attn_out[num_decode_tokens:]
```

评论区精华

核心讨论点：mgoin 对 gdn_linear_attn.py 中的 ROCm 路径变更加以质疑，询问“confidence here? ”。yewentao256 即时回应论证 z_out.zero_() 后的立即覆写 z_out[:] = z（同一函数内第 1039-1041 行）使清零冗余，作者信心已表达。mgoin 最终在阅毕 evals 结果后批准 PR。

正确性验证：yewentao256 贴出 Qwen3-4B + turboquant_4bit_nc 在 GSM8K 上的 eval 结果（exact_match 0.792），并引用已有单元测试用例 tests/kernels/quantization/test_nvfp4_quant.py、tests/kernels/quantization/test_per_token_group_quant.py，使 mgoin 确信变更是安全的。

- GDN ROCm 路径中 z_out.zero_() 的安全性 (correctness): 作者论证清晰，变更安全，maintainer 批准。

- 需要端到端模型评估以确认 TurboQuant 变更的正确性 (testing): 已提供端到端 eval 结果，证明变更不影响正确性。

风险与影响

- 风险：风险较低，原因如下：
  - 所有变更的共性是“张量在分配后会被完全覆写”，这被代码逻辑明确保证（内核写入、切片赋值等）。
  - GDN attention 的 z_out.zero_() 删除经过行内证据审查，确认立即被覆写。
  - FP4/FP8 量化以及 TurboQuant 的混合批次路径中，后续的 C++/Triton 内核会完整写入目标缓冲区，不存在读取未初始化数据的风险。
  - 唯一潜在薄弱点是未来代码维护中若在该张量的“分配”与“完全写入”之间插入了读取路径，才可能出现问题。此类风险极低，且在 review 中被覆盖。
  - 影响：影响范围：涉及 FP4 量化、FP8 量化的 scale 张量分配、TurboQuant 的混合批次 attention 输出、GDN 的 ROCm attention 路径。每个点各去掉一次 GPU 内存清零操作。

影响程度：较微，属于微观优化。典型推理场景下四次清零的累积耗时可能在毫秒级别，但每一点改进本身都很小。对于 vllm 这样延迟敏感的服务，每一项开销剔除均有正向价值。

测试覆盖：变更由现有单元测试覆盖（nvfp4_quant、per_token_group_quant），以及 Qwen3-4B 的 GSM8K eval 作为端到端验证。无新增测试文件。

兼容性：完全向后兼容，没有 API、配置或行为变化。

- 风险标记：缺少针对性新增测试，核心路径变更

关联脉络

- PR #43261 [Bug] Fix ci issue assert output_size is not None AssertionError: 同样涉及 FP8 量化路径（fp8_utils.py）的分配逻辑，均与量化内核输出缓冲区相关。
- PR #43195 Update KDA chunk prefill decay to use exp2 semantics: 同为性能优化 PR，涉及细化注意力 / 量化路径的算子性能，属于同类微优化风格。