PR #41326 完整报告

vllm-project/vllm

Faster per-token fp8 group quant packed kernel for blackwell

合并时间：2026-05-01 09:09

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/41326

执行摘要

- 一句话：Blackwell FP8 分组量化寄存器内核，加速 60%-2x
- 推荐动作：该 PR 值得精读，特别是 CUDA 寄存器优化和 shuffle 规约技术。评审中的三个高优先级问题展示了正确性与文档的平衡。后续同类优化可借鉴其 alignas 和 int64_t 的前置检查。

功能与动机

为 DeepGEMM MoE 的输入量化路径提供更快的 fp8 分组量化打包内核，在 Blackwell GPU 上实现 60%-2x 加速（根据批处理大小）。

实现拆解

1. 在 per_token_group_quant.cu 中引入寄存器驻留内核 per_token_group_quant_8bit_packed_register_kernel，每个线程持有 32 字节源数据（uint4 x 2）在寄存器中完成 absmax 规约、缩放计算和量化，消除共享内存和 __syncthreads。
2. 移除旧版共享内存内核 per_token_group_quant_8bit_packed_kernel 及其 host 辅助函数，清理 per_token_group_quant_8bit_packed_smem_impl。
3. 将每组的线程数从 16 降至 8，每线程加载量翻倍至 32 字节，shuffle 规约层级减少，提升寄存器利用率。
4. 在内核中添加 padding 零填充逻辑：当 output_q 存在 TMA 对齐的额外 mn 行时，内核在量化后原位清零这些行，确保下游无垃圾数据。
5. 根据 review 反馈修复：加入 alignas(16) 保证 uint4 加载对齐；将 num_groups_padded 和 num_scale_elems 从 int 改为 int64_t 避免大输入溢出；更新头文件注释，明确 group_size!=128 时报错而非 fallback。
6. 测试文件新增三个测试用例（all_zero、mantissa_rounds_up、zero_fills_padded_output_q），移除非 2 次幂 group size 参数（96）以匹配内核限制。

关键文件：

- csrc/libtorch_stable/quantization/w8a8/fp8/per_token_group_quant.cu（模块 量化内核；类别 source；类型 core-logic；符号 per_token_group_quant_8bit_packed_register_kernel）：核心变更文件：实现寄存器驻留 packed 量化内核，移除旧共享内存内核，调整 host 侧调度逻辑。

- csrc/libtorch_stable/quantization/w8a8/per_token_group_quant_8bit.h（模块 量化接口；类别 source；类型 core-logic；符号 per_token_group_quant_8bit_packed）：头文件添加新函数声明 per_token_group_quant_8bit_packed，并更新注释说明限制。
- tests/kernels/quantization/test_per_token_group_quant.py（模块 量化测试；类别 test；类型 test-coverage；符号 test_per_token_group_quant_fp8_packed_all_zero, test_per_token_group_quant_fp8_packed_mantissa_rounds_up, test_per_token_group_quant_fp8_packed_zero_fills_padded_output_q）：测试文件新增三个边界测试用例，移除不支持的 group size 参数化。

关键符号：per_token_group_quant_8bit_packed_register_kernel, per_token_group_quant_8bit_packed, test_per_token_group_quant_fp8_packed_all_zero, test_per_token_group_quant_fp8_packed_mantissa_rounds_up, test_per_token_group_quant_fp8_packed_zero_fills_padded_output_q

关键源码片段

csrc/libtorch_stable/quantization/w8a8/fp8/per_token_group_quant.cu

核心变更文件：实现寄存器驻留 packed 量化内核，移除旧共享内存内核，调整 host 侧调度逻辑。

```
// Register-resident fast path for group_size==128.
// 每个线程持有 16 个源元素（32 B，两个 uint4）在寄存器中完成：
// absmax 规约 → 缩放计算 → 量化流水线。不使用共享内存和 __syncthreads。
// UE8M0 缩放因子通过位运算提取（与 exp2f(ceilf(log2f)) 位精确）。
// Blackwell nvcc 会将连续 uint4 加载融合为单条 256-bit LDG.E.256 指令。
template <typename T, typename DST_DTYPE, int GROUP_SIZE>
__global__ void per_token_group_quant_8bit_packed_register_kernel(
    const T* __restrict__ input,
    void* __restrict__ output_q,
    unsigned int* __restrict__ output_s_packed,
    const int64_t num_groups_padded,
    const int groups_per_block,
    const int padded_groups_per_row,
    const int groups_per_row,
    const int mn,
    const int output_q_mn_extent,
    const int tma_aligned_mn,
    const int64_t num_scale_elems,
    const float eps,
    const float min_8bit,
    const float max_8bit) {
  static_assert(GROUP_SIZE == 128, "fast path supports GROUP_SIZE==128");
  constexpr int THREADS_PER_GROUP = 8;
  constexpr int VEC_SIZE = 32 / sizeof(T); // 对于 bf16/fp16 为 16
  /* 每个组的 8 个线程必须位于同一个 warp 八位组内，使得 shuffle mask
     0xffu << (threadIdx.x & 24u) 恰好选中同组线程。要求
     32 % THREADS_PER_GROUP == 0 且启动线程数为 group_per_block * 8。 */
  static_assert(32 % THREADS_PER_GROUP == 0);
```

```
  // ... 后续实现包括 absmax 规约、缩放计算、量化和存储
}
```

csrc/libtorch_stable/quantization/w8a8/per_token_group_quant_8bit.h

头文件添加新函数声明 per_token_group_quant_8bit_packed，并更新注释说明限制。

```
#pragma once

#include <torch/csrc/stable/tensor.h>

// 8-bit 逐 token 分组量化辅助函数（INT8 和 FP8 共用）
void per_token_group_quant_8bit(const torch::stable::Tensor& input,
                                torch::stable::Tensor& output_q,
                                torch::stable::Tensor& output_s,
                                int64_t group_size, double eps, double min_8bit,
                                double max_8bit, bool scale_ue8m0 = false);

// Public op: 面向 DeepGEMM Blackwell 路径的寄存器驻留 packed 量化。
// 仅支持 group_size == 128 且 bf16/fp16 输入；其他配置直接抛出 STD_TORCH_CHECK。
// 旧版共享内存回退已被移除，因为无生产调用方使用其他形状。
void per_token_group_quant_8bit_packed(const torch::stable::Tensor& input,
                                       torch::stable::Tensor& output_q,
                                       torch::stable::Tensor& output_s_packed,
                                       int64_t group_size, double eps,
                                       double min_8bit, double max_8bit);
```

tests/kernels/quantization/test_per_token_group_quant.py

测试文件新增三个边界测试用例，移除不支持的 group size 参数化。

```
@pytest.mark.skipif(
    not current_platform.is_cuda(), reason="DeepGEMM not available on this platform"
)
def test_per_token_group_quant_fp8_packed_all_zero():
    """全零输入必须通过内核 UE8M0 路径中的 eps 下界产生确定的 UE8M0 缩放字节。
    在优化前锁定全零行为。

    CUDA 内核计算：
        y_s = eps / fp8_max
        y_s = exp2(ceil(log2(fmax(y_s, 1e-10))))
    对于全零输入，eps/fp8_max < 1e-10，内部 fmax 钳回到 1e-10，得到
    exp2(ceil(log2(1e-10))) = exp2(-33) => UE8M0 字节 0x5E (94)。
    """
    device = "cuda"
    num_tokens, hidden_dim, group_size = 4, 7168, 128
    x = torch.zeros((num_tokens, hidden_dim), device=device, dtype=torch.bfloat16)

    out_q, out_s_packed = fp8_utils.per_token_group_quant_fp8_packed_for_deepgemm(
        x, group_size=group_size, use_ue8m0=True,
    )
```

```
    # 量化输出必须全零
    assert torch.equal(
        out_q.view(torch.uint8),
        torch.zeros_like(out_q, dtype=torch.uint8),
    ), "All-zero input should produce all-zero FP8 output"

    # 全零输入时内核产生的 UE8M0 字节
    expected_exp_byte = 0x5E

    mn = num_tokens
    groups_per_row = hidden_dim // group_size
    k_num_packed = (groups_per_row + 3) // 4
    tma_aligned_mn = ((mn + 3) // 4) * 4
    num_scale_elems = mn + (k_num_packed - 1) * tma_aligned_mn

    actual = torch.as_strided(out_s_packed, (num_scale_elems,), (1,)).cpu()
    expected = torch.zeros(num_scale_elems, dtype=torch.int32, device="cpu")
    for row in range(mn):
        for g in range(groups_per_row):
            pack_col = g // 4
            pos = g % 4
            idx = pack_col * tma_aligned_mn + row
            expected[idx] |= expected_exp_byte << (pos * 8)

    assert torch.equal(actual, expected), "All-zero scale bytes mismatch"
```

评论区精华

gemini-code-assist[bot] 提出三个高优先级问题：

- regs 数组需要 alignas(16) 保证 uint4 加载对齐（正确性）。
- num_groups_padded 和 num_scale_elems 应使用 int64_t 防止大输入溢出（潜在 bug）。
- 头文件注释声称 group_size!=128 会 fallback 到 smem 内核，但实际已报错，文档与实现不一致（回归风险）。 claude[bot] 指出上述三个问题关键且未被解决，并补充了注释中拼写换行的纯风格 nits。作者在后续提交中全部修复（commit d2cb322）。
- 寄存器数组对齐要求 alignas(16) (correctness): 作者在 commit d2cb322 中添加了 alignas(16) 修复。
- 参数类型 int 应改为 int64_t 防止溢出 (correctness): 作者在后续提交中将 kernel 签名改为 int64_t。
- 头文件注释与实现不符：fallback 误导 (documentation): 作者更新注释，明确限制并移除 fallback 描述。
- 注释换行拼写风格问题 (style): 作者在后续提交中修复。

风险与影响

- 风险：

1. group_size 硬限制：新内核仅支持 group_size==128，非 128 配置直接报错。若上游调用方（如 input_quant_fp8）可能使用其他 group size，则会功能回归。但 PR 说明无生产路径使用非 128 分组。
2. 对齐未定义行为：若 alignas(16) 未正确应用，uint4 加载可能触发硬件异常或性能降级——已在 review 后修复。
3. 输入溢出：原本 int 类型在超大上下文时可能溢出，已改为 int64_t。
4. 平台限制：新内核依赖 CUDA 12.8+ 指令（256-bit 加载），且测试仅在 CUDA 平台上运行；AMD 或其他后端不可用。
5. 测试覆盖：新测试仅覆盖 CUDA 平台，但 DeepGEMM not available on this platform 的 skip 条件合理。 - 影响：用户影响：Blackwell GPU 上使用 DeepGEMM MoE 的服务将获得显著加速（60%-2x），无需任何配置变更。非 Blackwell 或 group_size!=128 的用户不受影响（原有路径保持不变）。系统影响：内核二进制体积减少（移除 smem 变体），启动时间可能略有改善。团队影响：维护性降低，因为代码行数减少且依赖条件集中；但未来若需支持其他 group size 需重新设计。 - 风险标记：去除共享内存回退可能影响非 128 组大小，对齐问题已修复，大输入溢出已修复

关联脉络

- PR #40033 [NVFP4][Hopper/AMD Instinct] Add Triton kernels for NVFP4 dequantization and QDQ emulation: 同为量化 kernel 优化，涉及 fp8 精度处理，与本 PR 的量化路径相关，但本 PR 针对 Blackwell 的寄存器驻留优化。