PR #40835 完整报告

vllm-project/vllm

[Feature] Triton INT4 per-token-head KV cache quantization

合并时间：2026-06-24 18:21

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/40835

执行摘要

- 一句话：为 Triton 后端新增 INT4 per-token-head KV 缓存量化
- 推荐动作：值得精读，特别是 int4_per_token_head.py 中 nibble 打包、RHT 变换和零点隐写的细节，展示了高效 sub-byte 量化的工程实现。对于关注显存优化的团队，此 PR 提供了实用的新选项。建议关注后续对 ROCm 等平台的适配。

功能与动机

为降低 KV 缓存显存占用和带宽消耗，在现有 INT8/FP8 per-token-head 量化基础上增加更激进的 INT4 量化。INT4 通过每字节打包两个 4-bit 值，相比 INT8 可再减少约一半的缓存大小。同时，为确保量化精度，引入随机 Hadamard 变换（RHT）和非对称量化（带零点）。本 PR 基于 #40633 的 per-token-head 框架，但改为仅专注于 INT4 实现。

实现拆解

1. 量化模式定义：在 KVQuantMode 枚举中新增 INT4_PER_TOKEN_HEAD，更新 get_kv_quant_mode、is_per_token_head 等辅助函数，确保后端能正确识别 INT4 模式。
2. 专用 INT4 内核：新建 vllm/v1/attention/ops/int4_per_token_head.py，包含 nibble 级别打包 / 解包 Triton JIT 函数、写入内核 _reshape_cache_int4_kernel（执行 RHT 变换、计算每 token 每 head 的 scale 和零点并打包）、读取内核 _attn_packed / unified_attention_int4（解包、反量化、注意力计算）。
3. 集成到 Triton 后端：在 TritonAttentionBackend 中注册 int4_per_token_head 支持，调整 get_kv_cache_shape 使打包布局（head_size // 2 + scale_pad）。前向路径根据 _kv_quant_mode 分发：INT4 走专用内核，INT8/FP8 走统一内核。
4. 写入路径适配：在 triton_reshape_and_cache_flash.py 的 triton_reshape_and_cache_flash_per_token_head_quant 中添加 INT4 分支，直接调用 reshape_and_cache_int4。
5. 统一注意力路由：在 triton_unified_attention.py 的 unified_attention 中检测 KVQuantMode.INT4_PER_TOKEN_HEAD，转发到 unified_attention_int4。
6. 配置与类型映射：更新 vllm/utils/torch_utils.py 的 STR_DTYPE_TO_TORCH_DTYPE，添加 "int4_per_token_head": torch.uint8；更新 vllm/config/cache.py 的允许配置列表。
7. 测试：扩展 tests/quantization/test_per_token_kv_cache.py，新增 INT4 的 QuantConfig 和 test_int4_per_token_head、test_kv_quant_mode_int4 等方法，验证打

包、解包、量化循环精度及统一注意力端到端正确性。

关键文件：

- vllm/v1/attention/ops/int4_per_token_head.py（模块 INT4 量化；类别 source；类型 core-logic；符号 pack_int4_nibbles, unpack_int4_nibbles, _reshape_cache_int4_kernel, _run_reshape_kernel）：核心实现文件，包含所有 INT4 专用内核（nibble 打包、reshape 写入、拆分点积注意力）和公共入口函数。
- tests/quantization/test_per_token_kv_cache.py（模块 测试；类别 test；类型 test-coverage；符号 test_int4_per_token_head, test_kv_quant_mode_int4, _pack_int4）：扩展测试覆盖 INT4 模式，包括精度验证、模式识别和与统一注意力的集成测试。
- vllm/v1/attention/backends/triton_attn.py（模块 注意力后端；类别 source；类型 core-logic；符号 _pth_key_value_caches）：后端入口，注册 INT4 模式、调整 KV 缓存形状计算、新增 per-token-head 缓存视图方法。
- vllm/v1/kv_cache_interface.py（模块 缓存接口；类别 source；类型 core-logic；符号 KVQuantMode.INT4_PER_TOKEN_HEAD, get_kv_quant_mode）：定义 INT4_PER_TOKEN_HEAD 量化模式枚举及相关辅助函数，影响所有量化模式的识别和内存预算。
- vllm/v1/attention/ops/triton_unified_attention.py（模块 统一注意力；类别 source；类型 infrastructure）：在统一注意力入口中添加 INT4 模式检测和转发到专用内核的逻辑。
- vllm/v1/attention/ops/triton_reshape_and_cache_flash.py（模块 缓存重排；类别 source；类型 infrastructure）：在 per-token-head 写入函数中增加 INT4 分支，调用 reshape_and_cache_int4。

关键符号：KVQuantMode.INT4_PER_TOKEN_HEAD, get_kv_quant_mode, is_per_token_head, pack_int4_nibbles, unpack_int4_nibbles, reshape_and_cache_int4, unified_attention_int4, _pth_key_value_caches, _launch_packed_attn, _attn_packed, triton_reshape_and_cache_flash_per_token_head_quant

关键源码片段

tests/quantization/test_per_token_kv_cache.py

扩展测试覆盖 INT4 模式，包括精度验证、模式识别和与统一注意力的集成测试。

```
# == 新增 INT4 测试配置 ==
INT4_CONFIG = QuantConfig(
    cache_dtype=torch.uint8, # INT4 使用 uint8 作为存储类型（两个 4-bit 值打包）
    kv_cache_dtype_str="int4_per_token_head",
    quant_max=7.0, # 对称范围 [-8, 7]
    quant_min=-8.0,
    kv_quant_mode=KVQuantMode.INT4_PER_TOKEN_HEAD,
    # rounds_before_store 对 INT4 无效，其量化路径自带取整
    rounds_before_store=False,
)
# 测试参数组新增 INT4 模式
QUANT_CONFIGS = [INT4_CONFIG, INT8_CONFIG, FP8_CONFIG]
```

```
# == 验证 INT4 模式识别 ==
def test_int4_per_token_head(self):
    assert is_quantized_kv_cache("int4_per_token_head")

def test_kv_quant_mode_int4(self):
    from vllm.v1.kv_cache_interface import get_kv_quant_mode
    assert (
        get_kv_quant_mode("int4_per_token_head") == KVQuantMode.INT4_PER_TOKEN_HEAD
    )
```

vllm/v1/attention/backends/triton_attn.py

后端入口，注册 INT4 模式、调整 KV 缓存形状计算、新增 per-token-head 缓存视图方法。

```
# 在 get_kv_cache_shape 中，对于 per-token-head 模式计算数据维度：
if kv_cache_uses_per_token_head_scales(cache_dtype_str):
    # INT4 每个字节存储两个 4-bit 值，因此数据维度为 head_size // 2
    if get_kv_quant_mode(cache_dtype_str) == KVQuantMode.INT4_PER_TOKEN_HEAD:
        data_head_size = head_size // 2
    else:
        data_head_size = head_size
    # scale_pad 用于在尾部存储 float32 scale（每个 (token, head) 一个）
    scale_pad = get_dtype_size(torch.float32) // get_dtype_size(cache_dtype)
    return (num_blocks, 2, block_size, num_kv_heads, data_head_size + scale_pad)
return (num_blocks, 2, block_size, num_kv_heads, head_size)
```

评论区精华

- 内核复杂度与分发逻辑缩减：评审者 tdoublep 认为引入大量新内核和复杂分发逻辑破坏 Triton 后端可读性，要求尽量使用现有 unified_attention 内核。作者 JartX 移除了工厂抽象（QuantKVFactory），将 INT8/FP8 路由回统一内核，仅保留 INT4 专用内核，最终获得批准。
- 零点隐写术：JartX 将 4-bit 零点编码在 scale 的低 4-bit 中，避免额外存储，此设计被评审者接受。
- 测试设备硬编码：tdoublep 指出测试中多余的 device="cuda" 参数与 set_default_device 重复且硬编码 CUDA，JartX 移除了所有此类参数。
- 向后兼容重导出：mgoin 质疑向后兼容重导出的必要性，JartX 将其移除，直接更新调用方。
- INT2 模式移除：为简化 PR，JartX 移除了实验性的 INT2 模式，仅保留 INT4。
  - 内核复杂度与分发逻辑缩减 (design): JartX 移除了工厂抽象，将 INT8/FP8 路由回统一内核，仅保留 INT4 专用内核，得到 tdoublep 认可。
  - 零点隐写设计 (design): 评审者接受此设计，未提出替代方案。
  - 测试设备硬编码 (style): JartX 移除所有 device="cuda"，恢复默认设备上下文。
  - 向后兼容重导出 (design): JartX 移除了 triton_reshape_and_cache_flash.py 中的重导出，直接调用新位置。

风险与影响

- 风险：
  - 精度风险：INT4 量化位宽较低，可能对模型质量产生影响，需在真实任务中验证端到端指标。
  - 新内核稳定性：新增的 Triton JIT 内核（nibble 打包、解包、注意力）增加了编译失败或运行时错误的可能性，尤其是在非 CUDA 平台上。
  - 平台兼容性：当前仅验证 CUDA 平台，对 AMD ROCm 的适配尚未测试，某些内核依赖特定计算能力。
  - 内存布局变化：INT4 打包布局（head_size // 2 + scale_pad）改变了 KV 缓存形状，可能影响与 KV 连接器（如 Mooncake）或混合部署的兼容性。
  - 因果注意力限制：INT4 模式仅支持因果注意力，非因果场景未覆盖。
- 影响：
  - 用户侧：新增 --kv-cache-dtype int4_per_token_head 选项，可减少约 50% KV 缓存显存占用，有助于加载更大模型或支持更长上下文。
  - 系统侧：内核编译时间增加，但运行时性能预期优于纯软件模拟方案。
  - 团队侧：维护复杂度有所增加，但经过简化后的代码结构相对清晰，便于后续扩展。
  - 风险标记：新内核稳定性，INT4 精度影响，平台兼容性（ROCm 未验证），打包布局兼容性，仅支持因果注意力

关联脉络

- PR #40633 Per-token-head KV cache quantization (base): 本 PR 基于 #40633 的 per-token-head 量化框架，但改为仅支持 INT4 模式，并大幅简化了实现。