

PR #55713 完整报告

vllm-project/vllm

[Spec Decode] Add NVFP4 DSpark gathered top-k projection

合并时间: 2026-09-10 01:33

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/55713>

执行摘要

- 一句话: 为 NVFP4 量化的 DSpark 草稿模型添加融合 gathered top-k 投影 Triton 内核, 使量化后的 Markov 头能直接索引加权, 避免全词表反量化。
- 推荐动作: 本 PR 是一个针对特定模型 (Nemotron 3.5 Lightning NVFP4 DSpark) 的关键功能补全, 实现了量化权重的高效 gathered projection。核心 Triton 内核设计精巧 (gather → decode → scale → dot → scatter 全融合), 测试覆盖充分 (数值一致性、平台回退、错误检测、CUDA Graph)。建议精读以下内容: (1) `_nvfp4_gathered_bias_kernel` 的 Triton 实现, 理解 packed NVFP4 的字节级解码和 block-wise 反量化策略; (2) `modelopt.py` 中权重保留逻辑与 `qwen3_dspark.py` 分派逻辑的配合方式; (3) CodeRabbit 讨论中发现的 `assert→ValueError`、`values clone`、`CPU is_cuda` 检查等三个正确性修复。

功能与动机

DSpark 的 Markov 头通过 `markov_w2` 投影矩阵对草稿 logits 施加顺序依赖偏置。当该矩阵被 ModelOpt W4A16 NVFP4 量化后, 权重以 packed uint8 + per-group scale 存储, 无法像 BF16 密集权重那样用 `weight[index]` 直接索引。原有路径要么无法运行 (缺少 NVFP4 gathered 支持), 要么需要先对全词表反量化再索引, 严重违背 top-k 的效率初衷。PR body 明确指出: "With a quantized W2 Markov head, those rows are stored as packed NVFP4 values plus per-group/global scales, so they cannot be indexed and used as ordinary dense weights. The selected rows must be gathered and dequantized before the correction matmul. This is required to run nvidia/NVIDIA-Nemotron-3.5-Lightning-30B-A3B-NVFP4-DSpark."

实现拆解

- Triton 内核实现 (`vllm/model_executor/layers/quantization/utils/nvfp4_emulation_utils.py`): 新增 `_nvfp4_gathered_bias_kernel`, 每个 program 处理一个 (request, candidate) 对。内核加载 packed NVFP4 权重行, 用 `_e2m1_inline` 逐字节解码为 BF16, 乘以 per-group FP8 scale 和 global scale 反量化, 与 Markov 嵌入向量点积后将修正值累加到 base value 上, 最终散射到 logits 的对应位置。配套新增 `_nvfp4_gathered_bias_impl` (参数校验 + kernel 调度)、`_nvfp4_gathered_bias_fake` (CUDA graph fake 实现)、以及通过 `direct_register_custom_op` 注册的 `nvfp4_gathered_bias` 公共 API。同时修复了 `dequantize_to_dtype` 在 CPU tensor 上误

入 Triton 路径的条件判断（添加 `tensor_fp4.is_cuda` 检查）。

2. 权重保留与加载适配 (`vllm/model_executor/layers/quantization/modelopt.py`) : 在 `ModelOptLinearMethod.process_weights_after_loading` 中新增逻辑, 当 `layer._retain_weight_for_gather` 为 `True` 且权重为 `W4A16 NVFP4` 时, 将 `packed weight`、`weight_scale`、`weight_global_scale` 以 `persistent=False` `buffer` 形式保留在 `layer` 上, 并记录 `group_size`。这确保量化后的原始 `packed` 格式不会被 `kernel.repack` 覆盖丢失。
3. 模型侧分派 (`vllm/model_executor/models/qwen3_dspark.py`) : `DSparkMarkovHead.__init__` 新增 `retain_weight_for_gather` 参数 (由 `config.dspark_draft_topk` 是否非 `None` 决定), 并在 `apply_bias_gathered` 中增加条件分支: 当权重为 `NVFP4` 且设备 `/dtype` 均满足 `BF16+CUDA` 条件时, 直接调用 `nvfp4_gathered_bias`; 否则回退到 `dequantize_to_dtype` 反量化后再执行 `baddbmm_ + scatter_`。回退路径也兼容了未量化的密集权重。
4. 测试覆盖 (`tests/v1/spec_decode/test_dspark_topk.py`) : 新增三个测试:
`test_gathered_markov_bias_dequantizes_selected_w4a16_rows` (端到端验证 `NVFP4 gathered` 路径的反量化正确性, 覆盖 `CUDA` 和 `CPU` 回退)、
`test_nvfp4_gathered_bias_rejects_non_16_group_size` (验证非 `16 group size` 的 `ValueError`)、
`test_nvfp4_gathered_markov_bias_kernel_matches_reference` (批量参数化验证 Triton 内核与参考实现的数值一致性, 含 `CUDA Graph` 兼容性)。原有测试的 `_markov_head` 辅助函数也补齐了 `_retain_weight_for_gather` 和 `is_w4a16_nvfp4` 属性初始化。

关键文件:

- `vllm/model_executor/layers/quantization/utils/nvfp4_emulation_utils.py` (模块 量化工具; 类别 `source`; 类型 `core-logic`; 符号 `_nvfp4_gathered_bias_kernel`, `_nvfp4_gathered_bias_impl`, `_nvfp4_gathered_bias_fake`, `nvfp4_gathered_bias`) : 核心变更: 新增 `NVFP4 gathered bias` Triton 内核、`impl/fake` 封装和自定义算子注册
- `vllm/model_executor/models/qwen3_dspark.py` (模块 `DSpark` 模型; 类别 `source`; 类型 `core-logic`; 符号 `DSparkMarkovHead.init`, `DSparkMarkovHead.apply_bias_gathered`, `Qwen3DSparkModel.init`) : 模型侧集成: `DSparkMarkovHead` 的 `NVFP4 gathered` 分派逻辑与权重保留配置
- `vllm/model_executor/layers/quantization/modelopt.py` (模块 量化加载; 类别 `source`; 类型 `data-contract`; 符号 `ModelOptLinearMethod.process_weights_after_loading`) : 权重保留: 在 `ModelOpt` 加载流程中保留 `packed NVFP4` 权重副本供 `gathered` 投影使用
- `tests/v1/spec_decode/test_dspark_topk.py` (模块 `DSpark` 测试; 类别 `test`; 类型 `test-coverage`; 符号 `test_gathered_markov_bias_dequantizes_selected_w4a16_rows`, `test_nvfp4_gathered_bias_rejects_non_16_group_size`, `test_nvfp4_gathered_markov_bias_kernel_matches_reference`) : 测试覆盖: `NVFP4 gathered` 路径的端到端验证、错误检测和数值一致性

关键符号: `_nvfp4_gathered_bias_kernel`, `_nvfp4_gathered_bias_impl`, `_nvfp4_gathered_bias_fake`, `nvfp4_gathered_bias`,

`DSparkMarkovHead.apply_bias_gathered`, `ModelOptLinearMethod.process_weights_after`

_loading

关键源码片段

vllm/model_executor/layers/quantization/utils/nvfp4_emulation_utils.py

核心变更：新增 NVFP4 gathered bias Triton 内核、impl/fake 封装和自定义算子注册

```
@triton.jit
def _nvfp4_gathered_bias_kernel(
    markov_ptr, packed_weight_ptr, weight_scale_ptr, global_scale_ptr,
    values_ptr, index_ptr, logits_ptr,
    stride_markov_b, stride_markov_k, stride_weight_v, stride_weight_k,
    stride_scale_v, stride_scale_k, stride_values_b, stride_values_j,
    stride_index_b, stride_index_j, stride_logits_b, stride_logits_v,
    alpha, TOPK: tl.constexpr, K: tl.constexpr, NUM_BLOCKS: tl.constexpr,
):
    """对每个 (request, candidate) 程序，执行以下步骤：
    1. 从 index 中加载当前候选词的 vocab_id
    2. 按 16 元素 block 为单位，从 packed NVFP4 权重中加载 8 字节（16 个 nibble）
    3. 用 _e2m1_inline 逐字节解码为 float32
    4. 乘以 per-group FP8 scale 和 global scale 完成反量化
    5. 与 Markov 嵌入向量逐 block 点积并求和
    6. base_value + alpha * dot 得到修正后的 logit
    7. 将修正值散射到 logits 的 vocab_id 位置
    """
    candidate_id = tl.program_id(0)
    request_id = candidate_id // TOPK
    candidate = candidate_id - request_id * TOPK

    # 加载当前候选的词表 id
    vocab_id = tl.load(
        index_ptr + request_id * stride_index_b + candidate * stride_index_j
    ).to(tl.int64)

    block_ids = tl.arange(0, NUM_BLOCKS)
    byte_ids = tl.arange(0, 8)
    block_mask = block_ids * 16 < K

    # 加载 16 元素 block 对应的 8 字节 packed NVFP4 权重
    packed_offsets = block_ids[:, None] * 8 + byte_ids[None, :]
    packed = tl.load(
        packed_weight_ptr + vocab_id * stride_weight_v
        + packed_offsets * stride_weight_k,
        mask=block_mask[:, None], other=0,
    )
    # 解码高 4 位和低 4 位 nibble
    low = _e2m1_inline(packed & 0x0F)
    high = _e2m1_inline((packed >> 4) & 0x0F)
    decoded = tl.interleave(low, high) # [NUM_BLOCKS, 16]
```

```

# 加载并反量化 per-group scale
raw_weight_scale = tl.load(
    weight_scale_ptr + vocab_id * stride_scale_v + block_ids * stride_scale_k,
    mask=block_mask, other=0,
)
weight_scale = tl.cast(raw_weight_scale, tl.float8e4nv, bitcast=True).to(tl.float32)
global_scale = tl.load(global_scale_ptr).to(tl.float32)

# 加载 Markov 嵌入向量并执行 block-wise 点积
element_ids = tl.arange(0, 16)
k_offsets = block_ids[:, None] * 16 + element_ids[None, :]
k_mask = k_offsets < K
markov = tl.load(
    markov_ptr + request_id * stride_markov_b + k_offsets * stride_markov_k,
    mask=k_mask, other=0.0,
)

dequant_weight = (decoded * weight_scale[:, None] * global_scale).to(tl.bfloat16)
products = dequant_weight.to(tl.float32) * markov.to(tl.float32)
block_sums = tl.sum(products, axis=1)
dot = tl.sum(block_sums, axis=0)

# 修正并散射到 logits
base_value = tl.load(
    values_ptr + request_id * stride_values_b + candidate * stride_values_j
).to(tl.float32)
corrected = base_value + alpha * dot
tl.store(
    logits_ptr + request_id * stride_logits_b + vocab_id * stride_logits_v,
    corrected,
)

```

vllm/model_executor/models/qwen3_dspark.py

模型侧集成：DSparkMarkovHead 的 NVFP4 gathered 分派逻辑与权重保留配置

```

def apply_bias_gathered(
    self,
    markov_embed: torch.Tensor,
    logits: torch.Tensor,
    values: torch.Tensor,
    index: torch.Tensor,
    scale: float = 1.0,
) -> torch.Tensor:
    """仅对 top-k 选中的 logits 施加 Markov 偏置。

    当 markov_w2 为 NVFP4 量化权重且满足 CUDA + BF16 条件时,
    调用融合 Triton 内核; 否则回退到先反量化再 baddbmm_ + scatter_。
    """
    if self.markov_w2._retain_weight_for_gather and self.markov_w2.is_w4a16_nvfp4:

```

```

# 快速路径: CUDA BF16 fused Triton kernel
if (
    current_platform.is_cuda()
    and markov_embed.is_cuda
    and markov_embed.dtype == torch.bfloat16
    and values.dtype == torch.bfloat16
    and logits.dtype == torch.bfloat16
):
    nvfp4_gathered_bias(
        markov_embed,
        self.markov_w2._nvfp4_weight_for_gather,
        self.markov_w2._nvfp4_weight_scale_for_gather,
        self.markov_w2._nvfp4_weight_global_scale_for_gather,
        values, index, logits, scale,
    )
    return logits

# 慢速路径: 先反量化选定行, 再执行密集矩阵运算
flat_index = index.reshape(-1)
packed_weight = self.markov_w2._nvfp4_weight_for_gather.index_select(0, flat_index)
weight_scale = self.markov_w2._nvfp4_weight_scale_for_gather.index_select(0, flat_index)
weight = dequantize_to_dtype(
    packed_weight, weight_scale,
    self.markov_w2._nvfp4_weight_global_scale_for_gather,
    dtype=markov_embed.dtype,
    block_size=self.markov_w2._nvfp4_group_size_for_gather,
    swizzle=False,
).view(*index.shape, -1)
else:
    # 未量化权重: 直接索引
    weight = self.markov_w2.weight[index]
    if weight.dtype != markov_embed.dtype:
        raise RuntimeError(
            "Gathered Markov projection requires unquantized weights or "
            "retained ModelOpt W4A16 NVFP4 weights."
        )
    corrected = values.unsqueeze(-1).clone() # clone 避免修改原始 values
    corrected.baddbmm_(weight, markov_embed.unsqueeze(-1), beta=1.0, alpha=scale)
    return logits.scatter_(1, index, corrected.squeeze(-1))

```

评论区精华

1. CodeRabbit: 将 assert 改为 ValueError (已采纳) : CodeRabbit 指出 `_nvfp4_gathered_bias_impl` 中的 assert 在 `-O` 模式下会被移除, 可能导致 GPU 越界访问。建议将 group size 不匹配时的断言改为 ValueError。作者采纳后, `_nvfp4_gathered_bias_impl` 中对 `k % 16 != 0` 和 `weight_scale` 维度不匹配的检查均改为 `raise ValueError`。

2. CodeRabbit: fallback 路径 values 被原地修改 (已采纳) : CodeRabbit 发现 `corrected = values.unsqueeze(-1)` 与 values 共享内存, `baddbmm_` 会原地修改 caller 可见的 values 张量。NVFP4 kernel 路径只读 values 不修改, 但 fallback 路径会修改。作者修复为 `corrected = values.unsqueeze(-1).clone()`。
 3. CodeRabbit: CPU tensor 在 `is_cuda_alike` 平台上误入 Triton 路径 (已采纳) : `dequantize_to_dtype` 中原先只检查 `current_platform.is_cuda_alike()`, 在 CPU tensor + CUDA 主机的场景下会错误进入 Triton 路径。修复为同时检查 `tensor_fp4.is_cuda`。对应 commit `2e1add6acadb6f285d3f3551b6d56d0b06f3ccac` 专门修复此问题。
 4. benchislett: 要求补充 kernel 文档注释: reviewer 要求在 `_nvfp4_gathered_bias_kernel` 头部添加 docstring, 说明内核的职责。作者已在最终版本中添加了详细的 docstring, 描述了 BF16 和 NVFP4 两条路径的差异以及 fused 路径的必要性。
- NVFP4 kernel 参数校验应使用 `ValueError` 而非 `assert (correctness)`: 已采纳, `k % 16` 和 `weight_scale` 维度检查均改为 `raise ValueError`
 - fallback 路径 values 张量被原地修改 (`correctness`): 已采纳, 修复为 `corrected = values.unsqueeze(-1).clone()`
 - CPU tensor 在 `is_cuda_alike` 平台上误入 Triton `dequantize` 路径 (`correctness`): 已采纳, 修复为同时检查 `tensor_fp4.is_cuda`, 对应独立 commit `2e1add6ac`
 - Triton kernel 缺少文档注释 (`documentation`): 已采纳, 添加了详细 docstring 描述 `gather` → `dequantize` → `dot` → `scatter` 流程

风险与影响

- 风险:
 1. Triton 内核数值精度: NVFP4 反量化后使用 BF16 累加, 测试容忍度设为 `rtol=1e-2`, `atol=6.25e-2`。对于不同 scale 值分布的权重, 精度偏差可能超出预期。内核仅在 `k % 16 == 0` 且 `group_size=16` 时可用, 其他配置需走 fallback。
 2. 权重保留的内存开销: `_retain_weight_for_gather=True` 会额外保留 `packed weight`、`weight_scale`、`weight_global_scale` 三份 buffer (`persistent=False`), 对于大词表模型 (如 128k vocab), 这会增加约 `128k * rank/2` 字节的额外显存占用。
 3. CUDA Graph 兼容性: 内核通过 `direct_register_custom_op` 注册并提供 fake impl, 理论上支持 CUDA Graph 捕获。测试中仅验证了 `batch_size=1` 的场景, 多 batch CUDA Graph 未显式测试。
 4. 平台兼容性: 快速 Triton 路径仅限 CUDA + BF16 dtype, ROCm、FP16、CPU 均走 fallback。Fallback 路径的 `dequantize_to_dtype` 在 CPU 上使用非 Triton 实现, 性能差异可能显著。
 5. ModelOpt 耦合: `process_weights_after_loading` 中的 buffer 注册逻辑直接依赖 ModelOpt 的 `kNvfp4Static` 常量和 `group_size` 属性, ModelOpt 升级可能需要同步调整。
- 影响:
 1. 用户影响: 使 NVIDIA-Nemotron-3.5-Lightning-30B-A3B-NVFP4-DSpark 模型的 draft top-k 功能可用 (此前无法运行或 fallback 到不兼容的密集权重假设)。性能测试显示 BF16 路径下 TPS 提升约 3-4%。

2. 系统影响：仅影响 DSpark 投机解码路径中的 Markov 头，不影响主模型前向、其他投机解码模式或非 NVFP4 量化路径。变更集中在 model_executor 的 quantization 和 model 模块。
3. 团队影响：为后续 NVFP4 量化模型在投机解码中的应用建立了可复用的 gathered projection 模式。Triton 内核实现可作为类似 packed 量化权重 + 稀疏索引场景的参考。
 - 风险标记：核心路径变更，量化权重格式依赖，Triton 内核数值精度

关联脉络

- PR #56011 [CI][Test][Spec Decode] Fix CI failure of Qwen3 Omni DSpark loader mock: 同属 DSpark 投机解码功能线，修复 Qwen3 Omni DSpark 加载器模拟测试的 CI 失败
- PR #54371 [Qwen4Exp] Support UVA PLE-offload and Engram tensor parallelism: 同属模型量化与投机解码相关特性，涉及 qwen 系列模型的量化支持