

PR #47474 完整报告

vllm-project/vllm

[Perf] Cache `token\_to\_req\_indices` for dsV4, 5x~6x kernel performance improvement

合并时间: 2026-07-06 23:17

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/47474>

执行摘要

- 一句话: 缓存 `token_to_req_indices` 提升 DeepSeek V4 注意力 kernel 性能 5-6 倍
- 推荐动作: 建议精读, 这是一个典型的数据驱动缓存优化: 利用重复使用的中间结果减少不必要的 H2D 拷贝, 设计简洁且收益明确, 适合作为相似性能优化的模板。

功能与动机

PR 描述中指出, 原实现中每个 step 需要两次额外的 CPU 到 GPU 拷贝 (三个组件各一次), 微基准测试显示缓存后可获得 5-6 倍的 kernel 性能改进, 显著降低注意力元数据构建开销。

实现拆解

1. 在 `CommonAttentionMetadata` 中添加缓存机制 (`vllm/v1/attention/backend.py`): 新增 `_token_to_req_indices_cache` 实例属性和 `token_to_req_indices` 方法。方法首次调用时基于 `query_start_loc_cpu` 通过 numpy 计算 `token_to_req_indices`, 拷贝到 GPU 并缓存; 后续调用直接返回缓存的切片, 避免重复计算和 H2D 传输。
2. 消费方统一使用新方法: 在 `DeepseekV4FlashMLABackend.build` (`vllm/models/deepseek_v4/sparse_mla.py`)、`CompressorMetadataBuilder.build` (`vllm/models/deepseek_v4/compressor.py`)、`DeepseekSparseSWABackend.build` (`vllm/v1/attention/backends/mla/sparse_swa.py`) 中, 将原本独立且重复的 `np.repeat` + `torch.repeat_interleave` + `pin_memory copy` 逻辑替换为一行 `common_attn_metadata.token_to_req_indices(self.buffer)` 调用。
3. 删除冗余导入和局部变量: `sparse_mla.py` 删除了 `import numpy as np`、`from vllm.utils.torch_utils import np_to_pinned_tensor`, 以及 `build` 方法中的 `num_tokens`、`starts`、`seg_lengths` 等局部变量。
4. 无测试配套变更: 本次改动未直接修改测试文件; 缓存逻辑的验证依赖于现有集成测试。

关键文件:

- `vllm/v1/attention/backend.py` (模块 注意力层; 类别 source; 类型 core-logic; 符号 `token_to_req_indices`): 核心变更: 在 `CommonAttentionMetadata` 中新增 `_token_to_req_indices_cache` 属性和 `token_to_req_indices` 方法, 实现缓存机制的创建和管理。
- `vllm/models/deepseek_v4/sparse_mla.py` (模块 DeepSeek 模型; 类别 source; 类型 data-contract): DeepSeek V4 Sparse MLA 组件的构建入口, 将原有的手动计算替换为

token_to_req_indices 调用，消除重复逻辑。

- vllm/models/deepseek_v4/compressor.py (模块 压缩器; 类别 source; 类型 data-contract) : Compressor 组件同样消除了重复的 token_to_req_indices 构建逻辑, 使用缓存方法。
- vllm/v1/attention/backends/mla/sparse_swa.py (模块 SWA 后端; 类别 source; 类型 core-logic) : SWA 后端同样采用缓存方法, 移除原有的重复计算和 num_reqs 变量。

关键符号: token_to_req_indices

关键源码片段

vllm/v1/attention/backend.py

核心变更: 在 CommonAttentionMetadata 中新增 _token_to_req_indices_cache 属性和 token_to_req_indices 方法, 实现缓存机制的创建和管理。

```
# vllm/v1/attention/backend.py
```

```
def token_to_req_indices(self, buffer: torch.Tensor) -> torch.Tensor:
    """Build or reuse the per-token request index mapping."""
    num_tokens = self.num_actual_tokens
    # 如果缓存已存在, 直接返回切片 (零成本视图)
    if self._token_to_req_indices_cache is not None:
        assert self._token_to_req_indices_cache.device == buffer.device
        assert self._token_to_req_indices_cache.dtype == torch.int32
        assert self._token_to_req_indices_cache.shape[0] >= num_tokens
        return self._token_to_req_indices_cache[:num_tokens]

    # 首次计算: 基于 query_start_loc_cpu 生成每个 token 对应的 req_id
    starts = np.asarray(self.query_start_loc_cpu, dtype=np.int32)
    query_lens = np.diff(starts)
    token_to_req_indices = np.repeat(
        np.arange(query_lens.shape[0], dtype=np.int32), query_lens
    )
    num_mapped_tokens = token_to_req_indices.shape[0]
    assert buffer.shape[0] >= max(num_mapped_tokens, num_tokens)
    # 从 CPU pinned tensor 异步拷贝到 GPU
    buffer[:num_mapped_tokens].copy_(
        np_to_pinned_tensor(token_to_req_indices), non_blocking=True
    )
    # 如果实际 token 数不足缓存大小, 末尾补零 (用于 CUDA 图固定 shape)
    if num_mapped_tokens < num_tokens:
        buffer[num_mapped_tokens:num_tokens].zero_()
    # 缓存整个 buffer 的视图 (避免后续重新计算)
    self._token_to_req_indices_cache = buffer[:max(num_mapped_tokens, num_tokens)]
    return self._token_to_req_indices_cache[:num_tokens]
```

vllm/models/deepseek_v4/sparse_mla.py

DeepSeek V4 Sparse MLA 组件的构建入口，将原有的手动计算替换为 token_to_req_indices 调用，消除重复逻辑。

```
# vllm/models/deepseek_v4/sparse_mla.py

def build(
    self,
    common_prefix_len: int,
    common_attn_metadata: CommonAttentionMetadata,
    fast_build: bool = False,
) -> DeepseekV4FlashMLAMetadata:
    cm = common_attn_metadata
    # 旧代码：6 行手动计算 + copy，现替换为一句话调用
    # 内部自动处理缓存，首次计算后复用结果
    req_id_per_token = cm.token_to_req_indices(self.req_id_per_token_buffer)

    slot_mapping = cm.slot_mapping
    if self.compress_ratio > 1:
        slot_mapping = get_compressed_slot_mapping(
            cm.num_actual_tokens,
            cm.query_start_loc,
            cm.seq_lens,
            cm.block_table_tensor.clamp(min=0),
            int(self.kv_cache_spec.storage_block_size),
            self.compress_ratio,
            out=self.compressed_slot_mapping_buffer,
        )

    c128a_fields: dict[str, torch.Tensor | None] = {}
    if self.compress_ratio == 128:
        c128a_fields = self._build_c128a_metadata(cm, req_id_per_token)

    return DeepseekV4FlashMLAMetadata(
        ...
        req_id_per_token=req_id_per_token,
        ...
    )
```

评论区精华

无实质性讨论，获得 reviewer sfeng33 批准后直接合并。

- 暂无高价值评论线程

风险与影响

- 风险：

1. 缓存状态假设：token_to_req_indices 方法依赖 _token_to_req_indices_cache 的存在与否来判断是否重新计算。如果同一个 CommonAttentionMetadata 实例在两次调用之

间 `query_start_loc_cpu` 发生了变化，缓存会返回过期数据。但典型用法中，`metadata` 实例每次前向时重新创建，且在同一 `step` 内 `query_start_loc_cpu` 不变，因此风险较低。

2. `assert` 语句 crash: 方法中包含多个 `assert` (检查 `device`、`dtype`、`shape`)，如果传入的 `buffer` 不符合预期，会导致进程终止。生产环境中建议改为更优雅的错误处理。
3. 缺少测试覆盖: 新增的缓存逻辑没有对应的单元测试，回归风险由现有集成测试承担。
 - 影响: 影响范围限于 DeepSeek V4 模型 (`dsv4`) 的注意力路径。用户在使用 `dsv4` 时将获得显著的 `kernel` 性能提升 (微基准显示 5-6 倍)，且内存开销增加一个缓存的 `tensor` (大小不超过 `max_num_batched_tokens`)。对其他模型无影响。
 - 风险标记: 缓存状态假设，缺少测试覆盖，断言失败 crash

关联脉络

- 暂无明显关联 PR