# PR #52217 完整报告

vllm-project/vllm

[Attention] Vectorize sparse MLA mask loads

合并时间：2026-08-19 11:32

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/52217

## 执行摘要

- 一句话：Sparse MLA mask 向量化 128-bit 加载，B200 mask 开销降约 35%
- 推荐动作：值得精读，尤其是 dense_mask_mod 这段 cute 写法：assume + flat_divide + autovec_copy + recast_tensor 的组合是 CUTLASS 编程模型里做对齐向量加载的范式，可复用到其他 mask 或 packed 数据内核；“由 shape 函数统一保证对齐 + 测试固化不变量”的做法也值得借鉴。另一个关注点是 _build_topk_mask 的返回语义变化，合入后留意其他消费方。

## 功能与动机

PR body 的基准数据显示，sparse MLA 的 32-bit mask 加载在 FA4 内核中相对 unmasked baseline 带来 +53.7%~86.4% 的额外开销，是明显的性能瓶颈。作者将 mask_mod 改为一次加载 4 个 packed word（128-bit 向量），同时保留行内 padding 保证对齐，把开销压到 +17.8%~37.2%；由于只是换加载方式、不改变 mask 语义，作者声明无需模型评估，kernel 与 sparse-MLA 正确性测试即可覆盖。

## 实现拆解

1. mask_mod 向量化（vllm/model_executor/layers/attention/sparse_mla_mask.py）：重写 dense_mask_mod。旧实现每次 kv 位置做一次 32-bit word 加载加位移；新实现先用 cute.assume 断言 batch/query 方向 stride 可被 4 整除，再用 flat_divide 把 mask 行切成 4-word 块，autovec_copy 一次性把 128-bit 拷入寄存器，最后 recast_tensor 转成 4 个 Uint32 返回。__vec_size__ 由 32 改为 128，让上层按向量宽度展开。offset_dense_mask_mod 保持 32-bit 不变。
2. 数据契约：行宽对齐（vllm/model_executor/layers/attention/sparse_mla_attention.py）：_topk_mask_shape 在算完 num_words 后新增 triton.cdiv(num_words, 4) * 4，把每行 word 数补到 4 的倍数；_build_topk_mask 的返回值从 out[:B, :Q, :num_words] 改为 out[:B, :Q]，不在 view 层把 padding 切片掉，保证每行起始地址相对基址保持 128-bit 对齐；docstring 同步改为 “preserving padded row storage”。
3. 对齐不变量的测试（tests/v1/attention/test_sparse_mla_mask.py）：新增 test_topk_mask_rows_are_aligned_for_vectorized_loads（纯 CPU 校验 shape：129 个 key 对齐后仍为 8 words，含 reserve 场景也是 8）与 test_build_topk_mask_preserves_aligned_row_storage（CUDA 上构造 129 key 的 mask，断言结果为 8 个 word、bit 0 与 bit 128 位置正确、padding 保留）。

4. 性能验证（PR 内基准）：B200、BF16、8 heads、QK dim 192、V dim 128、num_splits=1、全 1 mask，三组 Q x KV 规模下 mask 相对开销由 32-bit 的 +56.2%/+86.4%/+53.7% 降到 128-bit 的 +21.1%/+37.2%/+17.8%；新旧 callback 输出逐位一致。

关键文件：

- vllm/model_executor/layers/attention/sparse_mla_mask.py（模块 注意力内核；类别 source；类型 core-logic；符号 dense_mask_mod）：核心 kernel 改动所在：dense_mask_mod 从 32-bit 逐字加载改为 128-bit 对齐向量加载，__vec_size__ 由 32 改为 128，是全部性能收益的来源。
- vllm/model_executor/layers/attention/sparse_mla_attention.py（模块 注意力层；类别 source；类型 data-contract；符号 _topk_mask_shape, _build_topk_mask）：数据契约变更所在：_topk_mask_shape 将 num_words 对齐到 4 的倍数，_build_topk_mask 返回 view 时保留 padding，是向量化加载成立的前提。
- tests/v1/attention/test_sparse_mla_mask.py（模块 单元测试；类别 test；类型 test-coverage；符号 test_topk_mask_rows_are_aligned_for_vectorized_loads, test_build_topk_mask_preserves_aligned_row_storage）：新增两个测试把“行宽 4 对齐”固化为不变量，覆盖非整 chunk 长度（129 key）场景，防止未来误伤对齐契约。

关键符号：dense_mask_mod, offset_dense_mask_mod, _topk_mask_shape, _build_topk_mask, test_topk_mask_rows_are_aligned_for_vectorized_loads, test_build_topk_mask_preserves_aligned_row_storage

## 关键源码片段

### vllm/model_executor/layers/attention/sparse_mla_mask.py

核心 kernel 改动所在：dense_mask_mod 从 32-bit 逐字加载改为 128-bit 对齐向量加载，__vec_size__ 由 32 改为 128，是全部性能收益的来源。

```
@cute.jit
def dense_mask_mod(
    batch: cute.TensorSSA,
    head: cute.TensorSSA,
    q_idx: cute.TensorSSA,
    kv_idx: cute.TensorSSA,
    seqlen_info,
    aux_tensors: list,
) -> cute.TensorSSA:
    dense_mask = aux_tensors[0]
    batch_idx = utils.ssa_to_scalar(batch)
    q_idx = utils.ssa_to_scalar(q_idx)
    kv_idx = utils.ssa_to_scalar(kv_idx)

    # 取 batch/query 两个方向的 stride，并用 assume(divby=4) 向编译器宣告
    # 它们都是 4 的倍数：配合下方按 4 个 word 分组，保证 128-bit 对齐加载合法
    batch_stride, query_stride, _ = dense_mask.stride
    aligned_mask = cute.make_tensor(
```

```
            dense_mask.iterator,
            cute.make_layout(
                dense_mask.shape,
                stride=(
                    cute.assume(batch_stride, divby=4),
                    cute.assume(query_stride, divby=4),
                    1,
                ),
            ),
        )

        # 取当前 (batch, query) 对应的整行 mask，再按 (4,) 做 flat_divide，
        # 把行切成若干 128-bit 块；kv_idx >> 5 是 32-bit word 索引，
        # 再 >> 2 得到该 kv 所属的 4-word chunk 索引
        mask_row = aligned_mask[batch_idx, q_idx, None]
        mask_chunks = cute.flat_divide(mask_row, (4,))
        mask_chunk = mask_chunks[None, (kv_idx >> 5) >> 2]

        # autovec_copy 一次把 128-bit 拷入寄存器，recast_tensor 将其解释为
        # 4 个 Uint32 后 load 返回，由上层按向量宽度消费，减少逐字加载与索引指令
        loaded = cute.make_rmem_tensor_like(mask_chunk)
        cute.autovec_copy(mask_chunk, loaded)
        return cute.recast_tensor(loaded, cutlass.Uint32).load()


    dense_mask_mod.__vec_size__ = 128
```

## vllm/model_executor/layers/attention/sparse_mla_attention.py

数据契约变更所在：_topk_mask_shape 将 num_words 对齐到 4 的倍数，_build_topk_mask 返回 view 时保留 padding，是向量化加载成立的前提。

```
    def _topk_mask_shape(
        batch_size: int,
        max_query_len: int,
        max_key_len: int,
        reserve_key_starts_word: bool = False,
    ) -> tuple[int, int, int]:
        """Shape of a bit-packed top-k mask, shared by every site that builds one."""
        tile_m = 128 if max_query_len <= 128 else 256
        padded_q_len = triton.cdiv(max_query_len, tile_m) * tile_m
        num_words = triton.cdiv(max_key_len, 32) + int(reserve_key_starts_word)
        # 关键对齐不变量：把每行 word 数补到 4 的倍数，使行长为 128-bit 的
        # 整数倍，sparse_mla_mask.dense_mask_mod 才能安全做向量化加载
        num_words = triton.cdiv(num_words, 4) * 4
        return batch_size, padded_q_len, num_words


    def _build_topk_mask(
        topk_indices_per_req: list[torch.Tensor],
```

```
    q_lens: list[int],
    max_q_len: int,
    max_seq_len: int,
    out: torch.Tensor,
) -> torch.Tensor:
    """Build a bit-packed top-k mask while preserving padded row storage."""
    batch_size = len(q_lens)
    num_words = (max_seq_len + 31) // 32
    total_rows = batch_size * max_q_len
    if total_rows == 0:
        # 空请求时同样保留完整二维 view，不裁剪 padding
        return out[:batch_size, :max_q_len]

    # 中间调用 _scatter_topk_single_req_kernel / _scatter_topk_kernel
    # 用 atomic_or 把 top-k 索引对应的 bit 写入各 mask 行，这里省略

    # 返回时保留 padding：不按 num_words 裁剪最后一维，
    # 这样返回 view 的每行起始地址仍然 128-bit 对齐
    return out[:batch_size, :max_q_len]
```

## tests/v1/attention/test_sparse_mla_mask.py

新增两个测试把“行宽 4 对齐”固化为不变量，覆盖非整 chunk 长度（129 key）场景，防止未来误伤对齐契约。

```
def test_topk_mask_rows_are_aligned_for_vectorized_loads() -> None:
    # 129 个 key 需要 5 个 32-bit word，对齐后行宽应为 8 个 word
    assert _topk_mask_shape(2, 129, 129) == (2, 256, 8)
    # reserve_key_starts_word 场景同样需要对齐到 8
    assert _topk_mask_shape(2, 129, 129, reserve_key_starts_word=True) == (
        2,
        256,
        8,
    )


@pytest.mark.skipif(not torch.cuda.is_available(), reason="requires CUDA")
def test_build_topk_mask_preserves_aligned_row_storage() -> None:
    shape = _topk_mask_shape(1, 1, 129)
    out = torch.zeros(shape, dtype=torch.int32, device="cuda")
    topk = torch.tensor([[0, 128]], dtype=torch.int32, device="cuda")

    mask = _build_topk_mask([topk], [1], 1, 129, out)

    # 行宽保持 8：bit 0 与 bit 128 分别落在 word 0 与 word 4，
    # 其余是保留的 padding，值为 0
    assert mask.shape == (1, 1, 8)
    assert mask[0, 0].tolist() == [1, 0, 0, 0, 1, 0, 0, 0]
```

# 评论区精华

review 中有两条值得记录的问答：

- simon-veitner-redhat 问 store 是否也能向量化："is the store vectorised and if not can we vectorize the store aswell?"，作者回复这里没有真实内存写、会被 lower 成寄存器赋值，改用 return cute.recast_tensor(loaded, cutlass.Uint32).load() 性能相同但更可读，于是顺手用了这个写法。
- simon-veitner-redhat 追问 Blackwell 上是否试过 256-bit 加载："did we try to leverage 256 bit load for blackwell?"，作者回复在 hdim 256 kernel 与通用 kernel 上都没有性能提升，因此停留在 128-bit。
- store 是否也能向量化 (performance): 不存在真实 store，改用 recast_tensor 提升可读性，性能不变。
- Blackwell 是否尝试 256-bit 加载 (performance): 256-bit 尝试后无收益，保持 128-bit 向量宽度。

# 风险与影响

- 风险：
  - 返回形状变化：_build_topk_mask 返回的 view 不再裁剪到 num_words，依赖 shape[-1] 与 key 长度直接换算的调用方需要核对；本 PR 未改普通 FA4 路径，影响面限定在 SM100 masked-MHA 路径。
  - 对齐假设：cute.assume(..., divby=4) 依赖所有 mask 分配都经过 _topk_mask_shape；若有调用方绕过该函数直接分配非对齐 buffer，会触发未定义行为。当前所有分配点均返回对齐 shape，测试也覆盖了 129 这类非整 chunk 长度。
  - 平台限定：_is_masked_mha_available 限定 SM100，其他 GPU 不受影响也不会受益；offset_dense_mask_mod 仍是 32-bit，与 dense_mask_mod 行为不一致但不影响正确性。
  - 内存：padding 使每行最多多 3 个 word，workspace 适配检查基于 _topk_mask_shape 自动覆盖，总内存占用有极小增量。
  - 影响：对用户：SM100（Blackwell）上 sparse MLA（如 DeepSeek top-k 稀疏注意力）的 mask 相关时间显著下降，大 Q x KV 场景尤其明显（相对开销最多降低约 50%），模型输出不变，无需重新评测。对团队：mask 行布局的数据契约更新，后续 kernel 若要再向量化需要保持同样的对齐约定；新增测试把"行宽 4 对齐"固化为不变量。影响范围窄，仅在 CUDA SM100 的 masked attention 内核路径。
  - 风险标记：数据契约变更：mask 行保留 padding，对齐假设依赖 divby=4，仅 SM100 平台生效，offset_dense_mask_mod 未向量化

# 关联脉络

- PR #52775 [Kernel] SM120: stop routing misaligned-M blockwise FP8 GEMMs to the small-M swapAB config: 同为 SM1xx 平台内核级性能优化，反映 NVIDIA Blackwell 内核调优方向上的并行工作；虽非同一功能线，但在基准数据与内核路由策略上可互相参考。