PR #50613 完整报告

vllm-project/vllm

[Attention][MLA] Per-request scheduling for MLA chunked context

合并时间：2026-08-07 00:45

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/50613

执行摘要

- 一句话：MLA prefill 改为按请求调度 chunk，异构 batch 延迟大幅下降
- 推荐动作：值得精读。重点关注 plan_mla_context_chunks 的按请求打包与对齐拆分二分搜索（aligned_split_len）、init_mla_context_partial/accumulate_mla_context_chunk 的 partial 生命周期管理，以及如何通过数据结构变更消除额外的 mask kernel。对 MLA、attention 调度和 kernel 合并设计感兴趣的工程师会从中获得有价值的模式。

功能与动机

实现 #50497。旧实现中，MLA 预填充把 workspace 平均分给每个带 context 的请求，每个迭代处理所有请求的同一 context 窗口；当 batch 中请求上下文长度差异巨大时，大部分 workspace 被浪费，且每个迭代都要对整个 prefill batch 做 attention 与 merge，chunk 数多、kernel 启动开销大。PR body 明确指出 “MLA prefill chunks are fit into the available workspace rather than forced to be the same size. This can reduce the overall number of chunks and improve prefill latency.”

实现拆解

1. 重构 metadata 数据结构（vllm/model_executor/layers/attention/mla_attention.py）：用 ContextChunk 描述一个 workspace 大小内的连续请求切片，ChunkedContextMetadata 改为包含扁平 chunks 列表、context_lens 和 empty_token_slices，替代原来的按 batch 列的 cu_seq_lens/starts/max_seq_lens 等列表矩阵。
2. 新增打包调度算法：plan_mla_context_chunks 按请求顺序把 context 填进 workspace，超预算时用 aligned_split_len 在块边界对齐地拆分尾部请求；build_mla_chunked_context_metadata 据此生成每个 chunk 完整的 tensor 字段（cu_seq_lens、seq_lens、token_to_seq 等）。这保证“每个 context 行恰好被 gather 一次、chunk 不超 workspace、chunk 内不含空 context 请求”。
3. 调整 prefill 计算主循环：_compute_prefill_context 从按 chunk 索引遍历改为按 chunk 对象遍历，新增 init_mla_context_partial 和 accumulate_mla_context_chunk 负责 partial 的初始化与增量合并，从而删除了 mask_empty_context 这个额外的 Triton kernel 覆盖步骤。
4. 统一各prefill后端接口：FlashAttention（flash_attn.py）、AITER（aiter_flash_attn.py）、TRTLLM（trtllm_ragged.py）、FlashInfer（flashinfer.py）、tokenspeed（

tokenspeed_mla.py）的 run_prefill_context_chunk 签名从 chunk_idx 改为接收 ContextChunk，直接用 chunk 内嵌的 cu_seq_lens、max_seq_len 等，消除了对全局 chunked_context 列表的索引和断言。

5. 稀疏 MLA 与测试配套：sparse_mla_attention.py 同步遍历 chunks，并将 workspace 下限从 max_num_seqs * block_size 降为 block_size；新增 tests/v1/attention/test_mla_context_chunks.py 覆盖打包不变量、continuation 限定、尾部拆分、DCP 本地 chunk 等；test_mla_backends.py 新增 chunked_context_prefill batch 规格，对所有后端与 SDPA 参考输出对比。

关键文件：

- vllm/model_executor/layers/attention/mla_attention.py（模块 注意力层；类别 source；类型 data-contract；符号 ContextChunk, ChunkedContextMetadata, plan_mla_context_chunks, aligned_split_len）：核心实现文件，重写 MLA 分块上下文调度：新增 ContextChunk 数据结构、plan_mla_context_chunks 打包算法、init_mla_context_partial/accumulate_mla_context_chunk，并连带调整 DCP 与稀疏路径。
- tests/v1/attention/test_mla_context_chunks.py（模块 调度测试；类别 test；类型 test-coverage；符号 build_chunked_context, test_chunks_gather_every_context_row_exactly_once, test_continuation_is_confined_to_a_chunks_first_request, test_tail_splitting_minimizes_chunks）：新增测试文件，核心验证 chunk 打包不变量（无重叠 / 空洞、不超 workspace、continuation 仅限首请求）以及 DCP 本地 chunk 数据的一致性。
- vllm/model_executor/layers/attention/sparse_mla_attention.py（模块 稀疏注意力；类别 source；类型 data-contract；符号 _compute_context_mha, _masked_mha_workspace_fits, determine_chunked_prefill_workspace_size）：稀疏 MLA 后端适配新的 chunk 数据结构，改用 init_mla_context_partial/accumulate_mla_context_chunk 组装 partial，并调整 workspace 下限与 top-k mask 拟合检查。
- tests/v1/attention/test_mla_backends.py（模块 后端测试；类别 test；类型 test-coverage；符号 test_chunked_context_backend_correctness, _run_backend_correctness）：增加 chunked_context_prefill batch 规格和对应的后端正确性测试，确保所有 MLA 后端在分块场景与 SDPA 参考一致。
- vllm/v1/attention/ops/triton_merge_attn_states.py（模块 合并算子；类别 source；类型 infrastructure；符号 mask_empty_context, mask_empty_context_kernel）：删除 mask_empty_context 及其 Triton kernel，因为新调度保证 chunk 不覆盖空 context，简化 merge 逻辑并移除对应测试。

关键符号：plan_mla_context_chunks, build_mla_chunked_context_metadata, init_mla_context_partial, accumulate_mla_context_chunk, aligned_split_len, run_prefill_context_chunk, _compute_context_mha, mask_empty_context

关键源码片段

vllm/model_executor/layers/attention/mla_attention.py

核心实现文件，重写 MLA 分块上下文调度：新增 ContextChunk 数据结构、plan_mla_context_chunks 打包算法、init_mla_context_partial/accumulate_mla_context_chunk，并连带调整 DCP 与稀疏路径。

```
# _ContextChunkPlan 描述一个 chunk 在请求空间上的布局，
# starts / seq_lens 按请求顺序记录每个请求在 chunk 内的 context 偏移与行数。
@dataclass
class _ContextChunkPlan:
    request_start: int
    request_end: int
    starts: list[int]
    seq_lens: list[int]
    is_continuation: bool


def plan_mla_context_chunks(
    context_lens: list[int],
    row_budget: int,
    max_context_chunk: int,
    split_alignment: int,
    padded_rows: Callable[[int], int],
) -> list[_ContextChunkPlan]:
    """把各请求的 context 打包到 workspace 大小的 chunk 中。"""

    # aligned_split_len 用二分查找在可用行数内选择最大的对齐拆分长度。
    def aligned_split_len(start: int, remaining: int, available: int) -> int:
        max_split = min(max_context_chunk, round_down(remaining - 1, split_alignment))
        low, high = 0, max_split // split_alignment
        while low < high:
            mid = (low + high + 1) // 2
            length = mid * split_alignment
            rows = padded_rows(start + length) - padded_rows(start)
            if rows <= available:
                low = mid
            else:
                high = mid - 1
        return low * split_alignment

    plans: list[_ContextChunkPlan] = []
    num_requests = len(context_lens)
    request = 0
    start = 0
    while request < num_requests:
        context_len = context_lens[request]
        if context_len == 0:
            # 无 context 的请求直接跳过，chunk 只覆盖带 context 的请求。
            assert start == 0
            request += 1
            continue
```

```
            request_start = request
            starts: list[int] = []
            seq_lens: list[int] = []
            rows = 0
            while request < num_requests and context_lens[request] > 0:
                remaining = context_lens[request] - start
                request_rows = padded_rows(start + remaining) - padded_rows(start)
                if rows + request_rows > row_budget:
                    # 当前请求放不下时，尝试在块边界对齐拆分尾部。
                    split_len = aligned_split_len(start, remaining, row_budget - rows)
                    if split_len > 0:
                        starts.append(start)
                        seq_lens.append(split_len)
                        rows += padded_rows(start + split_len) - padded_rows(start)
                        start += split_len
                        break
                rows += request_rows
                starts.append(start)
                seq_lens.append(remaining)
                request += 1
                start = 0
            assert seq_lens
            plans.append(
                _ContextChunkPlan(
                    request_start=request_start,
                    request_end=request_start + len(seq_lens),
                    starts=starts,
                    seq_lens=seq_lens,
                    is_continuation=starts[0] > 0,
                )
            )
        return plans
```

tests/v1/attention/test_mla_context_chunks.py

新增测试文件，核心验证 chunk 打包不变量（无重叠 / 空洞、不超 workspace、continuation 仅限首请求）以及 DCP 本地 chunk 数据的一致性。

```
    def test_tail_splitting_minimizes_chunks():
        """尾部请求可填满一个 chunk 并在下一个 chunk 头部继续。

        若不拆分尾部，这些 context 需要 3 个 chunk；在块边界拆分可降到 2 个，
        即 workspace 行数约束下的最低值。
        """
        metadata = build_chunked_context([768, 512, 512], [3, 5, 7], 1024)
        assert metadata is not None

        assert len(metadata.chunks) == 2
        first, second = metadata.chunks
        # 第一个 chunk 覆盖请求 0 的全部 768 行和请求 1 的前 256 行。
```

```
    assert first.starts.tolist() == [0, 0]
    assert first.seq_lens.tolist() == [768, 256]
    assert not first.is_continuation

    # 第二个 chunk 从请求 1 的偏移 256 继续，并覆盖请求 2 全部 512 行。
    assert second.starts.tolist() == [256, 0]
    assert second.seq_lens.tolist() == [256, 512]
    assert second.is_continuation
```

评论区精华

Review 中 LucasWilkinson 提出了若干优化建议：将 request_start/request_end 改为 slice 对象（已采纳），质疑 covers_all_context_tokens 早退路径在存在无 context 请求时 attn_output 的形状与初始化是否正确，以及部分测试断言（如 _preserves_request_order）是否过度具体。MatthewBonanni 回应"Yeah I also guarded against having no-context requests"，确认早退路径已做防护。最终 LucasWilkinson 批准："LGTM thank you!!!"

- init_mla_context_partial 的形状与空 context 请求处理 (correctness): 确认提前返回路径已处理无 context 请求，形状正确。
- covers_all_context_tokens 是否可简化为 len(chunks)==1 (design): 保留辅助方法或等价简化，早退路径安全。
- ContextChunk 字段使用 slice 表示范围 (design): 作者采纳，最终使用 request_slice 和 token_slice 字段。
- 测试是否过度具体化 (testing): 部分过于具体的断言被精简，保留核心不变量测试。

风险与影响

- 风险：核心调度路径改动集中在 mla_attention.py，影响所有 MLA 模型（DeepSeek、Kimi 等）的预填充正确性，回归风险高。删除 mask_empty_context 后，空 context 请求的 partial 依赖 init_mla_context_partial 正确初始化；若某些组合未覆盖，merge 可能读到未初始化内存产生 NaN。DCP 路径的本地 chunk 规划（padded_local_*、local_starts）复杂度高，跨 rank 一致性问题可能导致 all-gather 结果错位。workspace 下限变化会影响显存占用策略；短请求均匀场景可能因为 per-chunk 字段构建开销略有回退。
- 影响：对用户：长上下文、异构 batch 的 prefill 延迟显著降低（最坏场景约 4 倍）。对系统：chunk 数量减少降低 kernel 启动与 attention/merge 开销，删除一个 Triton kernel 覆盖 pass 简化了 merge 链路。对团队：MLA prefill 后端接口统一为 ContextChunk，未来新增后端（如 CPU MLA）需适配；同时为 chunked context 调度建立了可验证的不变量测试基线。
- 风险标记：核心路径变更，跨后端接口重构，未初始化内存风险，DCP 路径复杂度高，性能收益依赖 batch 异构性

关联脉络

- PR #49453 [CPU] Add MLA backend so DeepSeek-V2/V3 can run on CPU: 本 PR 统一了 MLA prefill 后端的 chunk 接口，CPU MLA 后端后续需适配新的 ContextChunk 参数。

- PR #51113 [Bugfix] Keep mamba align prefill chunks block-aligned past last_cache_position: 同样涉及 chunk 对齐与拆分，与本 PR 的 aligned_split_len 关注同一问题域。
- PR #51249 [Bugfix][Model] Add missing fused_qkv_a_proj to Kimi-Linear packed_modules_mapping: PR 涉及 Kimi K3 模型文件（vllm/models/kimi_k3/nvidia/mla.py），且标签含 kimi/k3，同属 Kimi MLA 优化线。