PR #51739 完整报告

vllm-project/vllm

[Kernel] Optimize long-context MLA cache gathers

合并时间：2026-08-11 13:23

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/51739

执行摘要

- 一句话：按逻辑页调度 MLA cache gather，raw copy 提速最高 331 倍
- 推荐动作：值得精读。这是典型的“调度粒度决定内存事务效率”的 kernel 优化案例：把 per-token 查找变成 per-page 任务，配合 __shared__任务广播、float4 向量化和 FP8→BF16 合并且写出，效果立竿见影；新增的 benchmark_cp_gather.py 也很适合作为内核性能复现工具。但要注意两点：一是自动化 review 指出的两处越界读与 int32 溢出问题在合入时没有可见修复，建议先确认是否已被后续 PR 修复或本地补上；二是本 PR 没有跑模型级 eval，合入主线前建议在 DeepSeek-V4/Kimi-K3 上做一轮长 prefill 的正确性与延迟回归。

功能与动机

PR body 的 Why 部分明确指出：“The raw gather launched a small batch/split grid of 1024-thread CTAs and walked long requests serially. The conversion variants repeated request/block lookup per token and left memory transactions underfilled. These costs dominate chunked prefill when requests have very different cached-context lengths.” 即原有的小 batch/split 网格只有 1024 线程，长请求被串行遍历；转换变体每个 token 都重复做请求 / 块查找，内存事务没有打满，导致长上下文且长度极不均匀的 chunked prefill 中 gather 开销成为瓶颈。PR 还声明这是性能专项、保持 draft，并要求合入前人工逐行审查并复现验证。

实现拆解

实现按以下 5 步展开：

1. 内核调度模型重构（csrc/libtorch_stable/cache_kernels.cu）：新增 GatherPageTask 结构与 map_gather_page_task 模板函数，把一维任务号线性扫描到 (req_id, logical_block, 页内有效 token 范围 , 输出起始 token)。每个 CTA 处理一个逻辑页，block table 每页只查一次；首尾部分页通过显式的 page_token_begin/page_token_end 走同一条路径，不需要逐 token fallback。三个内核 gather_and_maybe_dequant_cache_page、cp_gather_and_upconvert_fp8_kv_cache_page、cp_gather_cache_page 统一改为 page 任务模型，CTA 内用 __shared__广播任务和物理块号，再按 float4 向量粒度加载 / 存储，并针对长而不均匀的 prefill 调整 CTA geometry。raw copy 保留 vectorized/scalar 两种 stride 处理以覆盖非常规张量步长。

2. 数据契约收紧（vllm/model_executor/layers/attention/mla_attention.py）：_compute_prefill_context 与 _context_parallel_compute_prefill_context 中 cp_gather_cache 的 dst 从整个 workspace 改为 workspace[:toks]，与 FP8 upconvert 路径的调用方式对齐，让内核输出规模和 num_tokens 严格一致，也避免把预留给 allgather 的 workspace 尾部区域直接暴露给 gather 内核。
3. 正确性测试配套：tests/kernels/attention/test_cache.py 新增 test_cp_gather_cache_mla_large_uneven_sequences、test_gather_and_maybe_dequant_cache_mla_large_uneven_sequences 和 test_cp_gather_cache_mla_with_seq_starts，覆盖 32K 级长度、17/71 等短请求混批、seq_starts 开关、auto/fp8、乱序物理块、部分页、非对齐 stride 与零长度请求；tests/kernels/test_cp_gather_fp8.py 新增 test_cp_gather_fp8_large_uneven_sequences_with_starts，验证 FP8 upconvert 的 seq_starts 切片与拼接参考实现一致。所有测试都用逐 token 的参考实现与内核输出 assert_close。
4. 可复用基准：新增 benchmarks/kernels/benchmark_cp_gather.py，内置 single-60K、single-300K、skew-2/4/8 场景，支持 --variant/--scenario/--dtype/--block-size/--entry-size 等参数，通过 triton.testing.do_bench 输出中位延迟与有效带宽，用于复现本 PR 宣称的 220x/331x 提速。
5. CI/ 构建配套：.buildkite/test_areas/kernels.yaml 将 Kernels Mamba Test 超时从 40 分钟提高到 60 分钟，原因是该 job 在 H200 35GB MIG 上 909 个用例约需 42 分钟，原预算会在无失败的情况下取消任务。这个改动与内核优化无直接关系，是同 PR 提交里附带的 CI 稳定性修复。

关键文件：

- csrc/libtorch_stable/cache_kernels.cu（模块 缓存内核；类别 source；类型 core-logic；符号 GatherPageTask, map_gather_page_task, gather_and_maybe_dequant_cache_page, cp_gather_and_upconvert_fp8_kv_cache_page）：核心改动文件：三个 MLA cache gather 内核全部改为按逻辑页任务调度，新增 GatherPageTask/map_gather_page_task，并做向量化与合并且写出；性能提升和 review 指出的风险都集中在此。
- vllm/model_executor/layers/attention/mla_attention.py（模块 注意力层；类别 source；类型 data-contract；符号 _compute_prefill_context, _context_parallel_compute_prefill_context）：调用契约同步收紧：普通与上下文并行两条路径中 cp_gather_cache 的 dst 从 workspace 改为 workspace[:toks]，与 FP8 upconvert 路径对齐，保证内核输出规模与 num_tokens 一致。
- tests/kernels/attention/test_cache.py（模块 缓存测试；类别 test；类型 test-coverage；符号 test_gather_and_maybe_dequant_cache_mla_large_uneven_sequences, test_cp_gather_cache_mla_with_seq_starts, test_cp_gather_cache_mla_large_uneven_sequences）：新增本 PR 最重要的正确性覆盖：32K 级不均匀长请求、seq_starts 开关、auto/fp8、乱序物理块与部分页，全部用逐 token 参考实现对照 page 内核输出。
- tests/kernels/test_cp_gather_fp8.py（模块 FP8 转换；类别 test；类型 test-coverage；符号 test_cp_gather_fp8_large_uneven_sequences_with_starts）：补充 FP8 upconvert

路径的 seq_starts 切片与不均匀长请求测试，覆盖 page 任务模型在 FP8→BF16 转换场景下的正确性。

- benchmarks/kernels/benchmark_cp_gather.py（模块 基准测试；类别 other；类型 benchmark；符号 make_page_table, make_cache_gather, run, make_fp8_upconvert）：新增可复用的 cp_gather 系列基准脚本，覆盖 60K/300K 与 skew 场景，PR 中的 220x/331x 数据即由此产出，是后续性能回归的重要工具。
- .buildkite/test_areas/kernels.yaml（模块 CI 配置；类别 config；类型 configuration）：附带 CI 稳定性修复：Kernels Mamba Test 超时从 40 分钟提高到 60 分钟，避免 H200 MIG 上 909 个用例约 42 分钟被预算取消。

关键符号：map_gather_page_task, gather_and_maybe_dequant_cache_page, cp_gather_and_upconvert_fp8_kv_cache_page, cp_gather_cache_page, make_page_table, make_cache_gather, make_fp8_upconvert, make_maybe_dequant_gather, run_scenario, test_cp_gather_cache_mla_large_uneven_sequences, test_cp_gather_cache_mla_with_seq_starts, test_gather_and_maybe_dequant_cache_mla_large_uneven_sequences, test_cp_gather_fp8_large_uneven_sequences_with_starts, _compute_prefill_context, _context_parallel_compute_prefill_context

关键源码片段

csrc/libtorch_stable/cache_kernels.cu

核心改动文件：三个 MLA cache gather 内核全部改为按逻辑页任务调度，新增 GatherPageTask/map_gather_page_task，并做向量化与合并且写出；性能提升和 review 指出的风险都集中在此。

```
// 每个 CTA 负责一个"逻辑页"粒度的 gather 任务：
// 只需要查一次 block_table，就能连续拷贝该页内所有 token 对应的一整段 entry。
struct GatherPageTask {
  int32_t req_id; // 请求序号
  int32_t logical_block; // 请求 block_table 行内的逻辑页下标
  int32_t page_token_begin; // 本页实际拷贝的起始槽位（相对页首，兼容部分页）
  int32_t page_token_end; // 本页实际拷贝的结束槽位（开区间）
  int32_t output_token_begin; // 对应输出 workspace 的起始 token
};

// 将全局一维任务号映射到 (req_id, logical_block)。
// has_terminal_start 控制 cu_seq_lens 的末端语义，用于容纳不同的调用约定。
template <bool has_terminal_start>
__device__ __forceinline__ bool map_gather_page_task(
    int32_t task, const int32_t* __restrict__ output_starts, int32_t num_reqs,
    int32_t total_tokens, int32_t block_size,
    const int32_t* __restrict__ seq_starts, GatherPageTask& page) {
  int32_t relative_page = task;
  for (int32_t req_id = 0; req_id < num_reqs; ++req_id) {
    // 每个请求的输出 token 区间为 [output_begin, output_end)
    const int32_t output_begin = min(output_starts[req_id], total_tokens);
```

```
    int32_t output_end;
    if constexpr (has_terminal_start) {
      output_end = min(output_starts[req_id + 1], total_tokens);
    } else {
      output_end =
          min(req_id + 1 < num_reqs ? output_starts[req_id + 1] : total_tokens,
              total_tokens);
    }
    const int32_t seq_len = max(output_end - output_begin, 0);
    // seq_starts 非空时，源端从缓存内偏移 seq_starts 处开始取，
    // 因此首个逻辑页可能是部分页。
    const int32_t source_begin = seq_starts == nullptr ? 0 : seq_starts[req_id];
    const int32_t first_block = source_begin / block_size;
    const int32_t num_pages =
        cuda_utils::ceil_div(source_begin + seq_len, block_size) - first_block;
    if (relative_page < num_pages) {
      // 命中本请求的第 relative_page 页，计算有效拷贝范围（兼容首尾部分页）
      page.req_id = req_id;
      page.logical_block = first_block + relative_page;
      const int32_t page_begin = page.logical_block * block_size;
      const int32_t copy_begin = max(source_begin, page_begin);
      const int32_t copy_end =
          min(source_begin + seq_len, page_begin + block_size);
      page.page_token_begin = copy_begin - page_begin;
      page.page_token_end = copy_end - page_begin;
      page.output_token_begin = output_begin + copy_begin - source_begin;
      return true;
    }
    relative_page -= num_pages;
  }
  return false;
}
```

tests/kernels/attention/test_cache.py

新增本 PR 最重要的正确性覆盖：32K 级不均匀长请求、seq_starts 开关、auto/fp8、乱序物理块与部分页，全部用逐 token 参考实现对照 page 内核输出。

```
@pytest.mark.parametrize('kv_cache_dtype', ['auto', 'fp8'])
@pytest.mark.parametrize('use_seq_starts', [False, True])
@pytest.mark.parametrize('device', CUDA_DEVICES)
@torch.inference_mode()
def test_cp_gather_cache_mla_large_uneven_sequences(
    kv_cache_dtype, use_seq_starts, device,
):
    # 复现生产中最棘手的形态：batch 很小但请求长度差异极大（17 / 32768 / 71 token），
    # block table 用 randperm 打散物理块，保证"乱序物理页 + 尾页部分拷贝"路径被覆盖。
    block_size, entry_size, num_blocks = 64, 576, 1024
    src_cache = _create_mla_cache(
        num_blocks, block_size, entry_size, torch.bfloat16, kv_cache_dtype, device
```

```
)
_fill_mla_cache(src_cache, kv_cache_dtype=kv_cache_dtype)

starts = torch.tensor([3, 17, 5], dtype=torch.int32, device=device)
seq_starts = starts if use_seq_starts else None
seq_lens = torch.tensor([17, 32_768, 71], dtype=torch.int32, device=device)
batch_size = seq_lens.shape[0]
cu_seq_lens = torch.zeros(batch_size + 1, dtype=torch.int32, device=device)
cu_seq_lens[1:] = seq_lens.cumsum(dim=0)
block_table = torch.stack(
    [torch.randperm(num_blocks, device=device) for _ in range(batch_size)]
).to(torch.int32)

# 参考实现逐 token 构造预期输出，直接按物理块表寻址，
# 与内核"每页一次查表"的批量路径形成对照。
expected_batches = []
for req_id in range(batch_size):
    start = starts[req_id].item() if use_seq_starts else 0
    source_tokens = torch.arange(
        start, start + seq_lens[req_id].item(), device=device
    )
    physical_blocks = block_table[req_id, source_tokens // block_size].long()
    expected_batches.append(src_cache[physical_blocks, source_tokens % block_size])
expected = torch.cat(expected_batches)

dst = torch.empty_like(expected)
ops.cp_gather_cache(
    src_cache, dst, block_table, cu_seq_lens, batch_size, seq_starts,
)
torch.testing.assert_close(dst, expected)
```

评论区精华

自动化 review（depthfirst-app[bot]）针对 csrc/libtorch_stable/cache_kernels.cu 提出 3 条评论：

1. MEDIUM 越界读：cp_gather_cache_page（约 line 1351）与 cp_gather_and_upconvert_fp8_kv_cache_page（约 line 1306）在访问 block_table 前没有检查 page.logical_block < block_table_stride，而姊妹内核 gather_and_maybe_dequant_cache_page 有该守卫；过大的 seq_starts 可把 logical_block 推出 block table 行宽，从而读取任意物理块索引并访问无关显存。bot 建议把边界判断并入 has_task，越界即跳过拷贝并提前返回。
2. MEDIUM 有符号整型溢出：连续快速路径中 page.output_token_begin（int32_t）× entry_size_bytes（int32_t）在 int32 下相乘后才拓宽，FP32 entry（2304 字节）约在 93 万 token 时溢出，而 PR 自己的 skew-8 场景合计 116 万 token，溢出后的字节偏移可能导致 dst 越界写。bot 建议先 static_cast 再乘。

3. 人类 reviewer jeejeelee 最终 APPROVED，author 与 bot 评论之间没有可见回复；合入的 2 个 commit 中也没有看到针对这两类问题的修复。

- block_table 越界读：cp_gather_cache_page / cp_gather_and_upconvert_fp8_kv_cache_page 缺少 block_table_stride 检查 (correctness): bot 建议把边界判断并入 has_task，越界即跳过拷贝并提前返回；PR 合入时未见对应修复 commit。
- 有符号整型溢出导致 dst 越界写 (correctness): bot 建议先 static_cast 再乘；合入时未见修复确认。

风险与影响

- 风险：
  1. block_table 越界读：cp_gather_cache_page 与 cp_gather_and_upconvert_fp8_kv_cache_page 缺少 page.logical_block < block_table_stride 守卫，异常 / 恶意 seq_starts 可产生越界块表索引。该区域是 PR body 自己提到的 #45537（block-table bounds 正确性修复）的历史问题区，说明此类缺陷曾真实发生过；虽然正常调度下 chunk.starts 由上层生成，但作为内核层防御仍应补齐。
1. int32 乘法溢出：连续快速路径用 int32_t 计算 page.output_token_begin * entry_size_bytes，FP32 entry 在约 93 万 token 时溢出，而本 PR 的基准已覆盖 116 万 token 场景，存在越界写风险。非连续路径使用 int64_t 的 dst_entry_stride 计算，因此问题只影响连续路径。
2. 等价性验证不足：PR 明确声明“Model evaluation was not run”，只依赖 exact/reference kernel 测试。测试覆盖了 auto/fp8、seq_starts、shuffled blocks、部分页、不均匀长度、非对齐 stride，但都是在单 GPU（GB300）上的 kernel 级校验；对调度 / 内存访问语义变化的模型级回归（尤其 DeepSeek-V3.2/V4 与 Kimi-K3 类 MLA 模型）仍是空白。
3. 数据契约收紧：mla_attention.py 把 cp_gather_cache 的 dst 改为 workspace[:toks]，与 issue 51252 描述的 sparse-indexer prefill buffer 预算不一致 bug 处于同一工作区；如果上层 chunker 与 workspace 行数再出现不一致，新的切片写法会改变越界行为的表现形式（虽然 torch 切片不会越界，但语义上要求 toks <= workspace 行数）。
4. 性能结论的平台局限：220x/331x 是在 NVIDIA GB300 上测得的 median cold-cache latency；ROCm/ 其他架构没有数据，CTA geometry 调优未必迁移，不能直接外推到所有平台。 - 影响：影响范围集中在 MLA 系模型的 chunked prefill / 长上下文缓存读取路径：cp_gather_cache、cp_gather_and_upconvert_fp8_kv_cache、gather_and_maybe_dequant_cache 是 DeepSeek-V3.2/V4、Kimi-K3 等 MLA 架构在长 prefill 时的高频内核。raw copy 在 60K 上下文获得 220.7x、300K 上下文 331.3x 的内核延迟改善，FP8 upconvert 基本持平（0.99x-1.10x），maybe-dequant 获得 1.39x-1.77x 提升，Nsight Compute 显示 DRAM 吞吐接近 75%-84% 峰值，对长上下文 TTFT 和 chunked prefill 的吞吐有直接帮助。对用户而言是透明的性能优化，无 API 变化；对团队而言新增了可复现的 kernel benchmark 和长上下文压力测试，后续内核优化可以直接复用；但合入时遗留的越界 / 溢出隐患意味着生产环境应等待 follow-up 修复或自行补

丁。 - 风险标记：核心内核重写，越界读风险（block_table 边界检查缺失），int32 乘法溢出风险，未运行模型级 eval，长上下文性能敏感路径

关联脉络

- PR #51252 [Bugfix] Size sparse-indexer prefill buffer by compress_ratio for DeepSeek-V4: PR body 的 Duplicate-work check 明确提到该 issue 修复 sparse-indexer prefill buffer 尺寸；与本 PR 同属 cp_gather 内核工作区正确性范畴，且都在 DeepSeek-V4 chunked prefill 路径上。
- PR #45537 block-table bounds correctness fix（PR body 提及）：PR body 提到 #45537 是 block-table 边界正确性修复；与本 PR 改写的 cp_gather 内核区域高度相关，review bot 也再次指出同类越界风险。
- PR #50484 [Kimi-K3] DCP support: 该 PR 同样修改 vllm/model_executor/layers/attention/mla_attention.py 与上下文并行 workspace 布局；本 PR 在 _context_parallel_compute_prefill_context 中收紧 cp_gather_cache 的 dst 切片，二者存在相同的 CP gather 契约。