PR #41433 完整报告

vllm-project/vllm

[Perf][2/n] Eliminate GPU<->CPU syncs in pooling code

合并时间：2026-05-05 10:43

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/41433

执行摘要

- 一句话：消除 pooler 中 GPU->CPU 同步点
- 推荐动作：值得精读，尤其是理解如何通过将索引和元数据保留在 CPU、异步回传 GPU 来消除同步点。设计决策清晰（显式切片 vs. torch.split），重构思路可推广到其他类似场景。

功能与动机

PR body 指出这是 PR#40561 发现的第二批次不必要的 GPU->CPU 同步。同步点会阻塞 GPU 流水线，影响吞吐。

实现拆解

1. DispatchPooler 切片 hidden_states（special.py）：引入 token_offset 和 pooling_metadata.pooling_cursor，为每个分组切片出子 hidden_states 并修正索引偏移，避免子 Pooler 访问超大张量导致同步。
2. AllPool 避免 .item() 同步（tokwise/methods.py）：移除 first_token_indices_gpu[0].item() 和 last_token_indices_gpu[-1].item()，直接使用已在 CPU 上的 num_scheduled_tokens_cpu.tolist() 配合 torch.split 分割张量，去掉条件分支。
3. StepPool 使用 CPU token ID（tokwise/methods.py）：改用 get_prompt_token_ids_cpu() 获取 CPU 副本，先通过 nonzero 得到索引再 non_blocking 回 GPU，避免布尔索引触发同步。
4. MeanPool 在 CPU 构建 segment_ids（seqwise/methods.py）：将 prompt_lens_cpu 直接用于 repeat_interleave 和 arange，避免数据依赖的输出长度推断同步，然后 non_blocking 上传。

关键文件：

- vllm/model_executor/layers/pooler/special.py（模块 Pooler；类别 source；类型 data-contract；符号 DispatchPooler.forward）：核心入口，引入 cursor 和 token_offset 切片逻辑，彻底改变子 Pooler 接收的 hidden_states 范围。
- vllm/model_executor/layers/pooler/tokwise/methods.py（模块 Pooler；类别 source；类型 data-contract；符号 AllPool.forward, StepPool.forward）：AllPool 和 StepPool 的核心改进：去除同步点和使用 CPU 张量。
- vllm/model_executor/layers/pooler/seqwise/methods.py（模块 Pooler；类别 source；类型 data-contract；符号 MeanPool.forward）：MeanPool 中通过 CPU 构建

segment_ids 避免同步，同时保留语义。

关键符号：DispatchPooler.forward, AllPool.forward, StepPool.forward, MeanPool.forward

关键源码片段

vllm/model_executor/layers/pooler/special.py

核心入口，引入 cursor 和 token_offset 切片逻辑，彻底改变子 Pooler 接收的 hidden_states 范围。

```
# vllm/model_executor/layers/pooler/special.py
import dataclasses
from collections.abc import Mapping, Set
from itertools import groupby

import torch

from vllm.config import PoolerConfig
from vllm.model_executor.layers.pooler import PoolingParamsUpdate
from vllm.tasks import PoolingTask
from vllm.v1.pool.metadata import PoolingMetadata

from .abstract import Pooler, PoolerOutput
from .common import ClassifierFn
from .seqwise import (
    SequencePoolingFn, SequencePoolingMethod,
    pooler_for_classify, pooler_for_embed,
)
from .tokwise import AllPool, pooler_for_token_classify, pooler_for_token_embed


class DispatchPooler(Pooler):
    # ... (for_embedding, for_classify, __init__ unchanged) ...

    def forward(
        self,
        hidden_states: torch.Tensor,
        pooling_metadata: PoolingMetadata,
    ) -> PoolerOutput:
        poolers_by_task = self.poolers_by_task
        cursor = pooling_metadata.pooling_cursor # 新增：整体 batch 的 cursor
        outputs = list[torch.Tensor | None]()
        offset = 0
        token_offset = 0 # 新增：跟踪当前分组在 hidden_states 中的 token 起始位置
        for task, group in groupby(pooling_metadata.tasks):
            if not (pooler := poolers_by_task.get(task)):
                raise ValueError(...)
            num_items = len(list(group))
```

```
            group_metadata = pooling_metadata[offset : offset + num_items]
            if cursor is None:
                group_hidden_states = hidden_states
            else:
                # 切片出本分组对应的 token，避免子 Pooler 看到无关 token
                group_cursor = group_metadata.pooling_cursor
                # 使用 CPU 上的 num_scheduled_tokens_cpu 计算 token 数，避免同步
                num_group_tokens = int(group_cursor.num_scheduled_tokens_cpu.sum())
                group_hidden_states = hidden_states[
                    token_offset : token_offset + num_group_tokens
                ]
                if token_offset:
                    # 修正 first/last_token_indices_gpu 为相对切片起始的偏移
                    pooling_cursor = dataclasses.replace(
                        group_cursor,
                        first_token_indices_gpu=(
                            group_cursor.first_token_indices_gpu - token_offset
                        ),
                        last_token_indices_gpu=(
                            group_cursor.last_token_indices_gpu - token_offset
                        ),
                    )
                    group_metadata = dataclasses.replace(
                        group_metadata, pooling_cursor=pooling_cursor
                    )
                token_offset += num_group_tokens

            group_output: PoolerOutput = pooler(group_hidden_states, group_metadata)
            outputs.extend(group_output)
            offset += num_items

        return outputs
```

vllm/model_executor/layers/pooler/tokwise/methods.py

AllPool 和 StepPool 的核心改进：去除同步点和使用 CPU 张量。

```
# vllm/model_executor/layers/pooler/tokwise/methods.py

class AllPool(TokenPoolingMethod):
    def __init__(self):
        super().__init__()
        vllm_config = get_current_vllm_config()
        self.enable_chunked_prefill = vllm_config.scheduler_config.enable_chunked_prefill

    def forward(self, hidden_states, pooling_metadata):
        pooling_cursor = pooling_metadata.get_pooling_cursor()
        # 直接使用已在 CPU 上的 num_scheduled_tokens_cpu，避免 .item() 触发同步
        hidden_states_lst = list(
            torch.split(hidden_states, pooling_cursor.num_scheduled_tokens_cpu.tolist())
```

```
        )

        if not self.enable_chunked_prefill:
            return hidden_states_lst

        pooling_states = pooling_metadata.pooling_states
        for p, hs_chunk in zip(pooling_states, hidden_states_lst):
            p.hidden_states_cache.append(hs_chunk)

        output_list = []
        for p, finished in zip(pooling_states, pooling_cursor.is_finished()):
            if finished:
                cache = p.hidden_states_cache
                if len(cache) == 1:
                    output_list.append(cache[0])
                else:
                    output_list.append(torch.concat(cache, dim=0))
                p.clean()
            else:
                output_list.append(None)
        return output_list


class StepPool(AllPool):
    def forward(self, hidden_states, pooling_metadata):
        pooled_data_lst = super().forward(hidden_states, pooling_metadata)
        # 调用新的返回 CPU 张量的方法，避免布尔索引触发 GPU->CPU 同步
        prompt_token_ids_cpu = pooling_metadata.get_prompt_token_ids_cpu()
        pooling_params = pooling_metadata.pooling_params

        pooled_data = []
        for data, token_id_cpu, pooling_param in zip(
            pooled_data_lst, prompt_token_ids_cpu, pooling_params
        ):
            if data is None:
                pooled_data.append(None)
            else:
                step_tag_id = pooling_param.step_tag_id
                returned_token_ids = pooling_param.returned_token_ids
                if returned_token_ids is not None and len(returned_token_ids) > 0:
                    data = data[:, returned_token_ids]
                if step_tag_id is not None:
                    # 在 CPU 上计算索引，然后 non_blocking 回 GPU
                    idx_cpu = (token_id_cpu == step_tag_id).nonzero(as_tuple=True)[0]
                    idx = idx_cpu.to(data.device, non_blocking=True)
                    data = data[idx]
                pooled_data.append(data)
        return pooled_data
```

vllm/model_executor/layers/pooler/seqwise/methods.py

MeanPool 中通过 CPU 构建 segment_ids 避免同步，同时保留语义。

```
# vllm/model_executor/layers/pooler/seqwise/methods.py

class MeanPool(SequencePoolingMethod):
    def forward(self, hidden_states, pooling_metadata):
        pooling_cursor = pooling_metadata.get_pooling_cursor()
        assert not pooling_cursor.is_partial_prefill(), (
            "partial prefill not supported with MEAN pooling"
        )

        prompt_lens_cpu = pooling_cursor.prompt_lens_cpu
        num_seqs = prompt_lens_cpu.numel()
        hidden_size = hidden_states.shape[-1]

        if num_seqs == 0:
            return hidden_states.new_empty((0, hidden_size), dtype=torch.float32)

        # 在 CPU 上构建 segment_ids，因为 repeat_interleave 需要知道输出长度
        # 数据依赖，如果在 GPU 上做会触发同步获取长度。这里在 CPU 计算后
        # 一次 non_blocking 上传 GPU，消除同步。
        segment_ids = torch.repeat_interleave(
            torch.arange(num_seqs, dtype=torch.long),
            prompt_lens_cpu,
        ).to(hidden_states.device, non_blocking=True)
        prompt_lens = prompt_lens_cpu.to(
            hidden_states.device, dtype=torch.int64, non_blocking=True
        )
        segment_sums = torch.zeros(
            (num_seqs, hidden_size),
            dtype=torch.float32,
            device=hidden_states.device,
        )
        # ... 余下 index_add_ 循环 ...
```

评论区精华

claude[bot] 和 gemini-code-assist[bot] 自动评论。gemini-code-assist[bot] 指出 tokwise/methods.py 中移除显式切片逻辑可能导致 torch.split 总和与 hidden_states 维度不匹配的回归，但该评论未被人工审核采纳（可能实际不会出现多组情况）。人工审核者 noooop 和 yewentao256 均给出 LGTM/Approved，未提出实质问题。

- tokwise/methods.py 中 torch.split 回归风险 (correctness): 人工审核未采纳，实际由于 DispatchPooler 已切片，不会出现问题。

风险与影响

- 风险：回归风险：AllPool 的 torch.split 要求 num_scheduled_tokens_cpu.tolist() 总和等于 hidden_states 第一维大小，若 DispatchPooler 传入的 hidden_states 包含多个分组外的 token 则会抛出 RuntimeError。但当前代码中 DispatchPooler 会先切片，因此实际上不会出现不匹配。性能：全 CPU 操作和异步传输可降低延迟，但 CPU 端计算仍需代价。
- 影响：影响范围仅限于 pooler 模块的三个文件，不会影响其他路径。对使用 pooling 功能的用户（包括 embedding、classify）会有性能提升，降低 GPU 空闲时间。没有接口或配置变更，向后兼容。
- 风险标记：核心路径变更，缺少测试覆盖

关联脉络

- PR #40561 Eliminate GPU<->CPU syncs in pooling code [1/n]: 第一篇消除同步的 PR，本 PR 是其直接延续。