PR #42604 完整报告

vllm-project/vllm

DeepSeekV4-Pro enable cuda graph full and piecewise mode

合并时间：2026-05-15 16:45

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/42604

执行摘要

- 一句话：DSV4 Pro 启用 FULL_AND_PIECEWISE CUDA 图模式
- 推荐动作：值得精读，尤其是为 CUDA 图提供稳定元数据 buffer 的做法以及处理嵌套编译冲突的思路。对于动态 shape 切片的潜在性能问题可进一步优化，但作为快速启用已足够。后续 PR 应关注准确率修复和单元测试补充。

功能与动机

PR 标题和描述明确目的：为 DeepSeek-V4 Pro 启用 FULL_AND_PIECEWISE CUDA 图模式。Commit 消息补充说明：需要让稀疏 MLA 元数据在 FULL decode 图回放时保持稳定，以及移除嵌套 @torch.compile 以使图捕获启动正常。

实现拆解

1. 在 rocm_aiter_mla_sparse_dsv4.py 中新增 _copy_ragged_to_graph_buffers 函数，将动态的 ragged indices 和 indptr 拷贝到预分配的持久 buffer 中，返回固定地址的视图。
2. 在 DeepseekV4ROCMAiterMLASparseMetadataBuilder 和 DeepseekV4ROCMAiterSparseSWAMetadataBuilder 的 __init__ 中根据 max_num_batched_tokens 和 c128a_max_compressed / window_size 预分配 indices 和 indptr 的持久 buffer。
3. 在 build 方法中，在生成 ragged 元数据后调用 _copy_ragged_to_graph_buffers 将其拷贝到持久 buffer，确保 CUDA 图捕获的 tensor 地址稳定。
4. 在 mhc.py 中移除对 vllm.platforms.current_platform 的导入以及 forward_hip 上的 @torch.compile 装饰器，避免嵌套编译导致的参数丢失问题。
5. 测试配套：PR 描述提供了服务器启动、客户端基准和 GSM8K 准确率测试命令，但未包含单元测试文件变更。

关键文件：

- vllm/v1/attention/backends/mla/rocm_aiter_mla_sparse_dsv4.py（模块 注意力模块；类别 source；类型 core-logic；符号 _copy_ragged_to_graph_buffers, DeepseekV4ROCMAiterMLASparseMetadataBuilder.init, DeepseekV4ROCMAiterSparseSWAMetadataBuilder.init）：核心改动：新增 _copy_ragged_to_graph_buffers 函数和元数据构建器的 __init__ 方法，为 FULL_AND_PIECEWISE CUDA 图提供稳定的 tensor 地址。

- vllm/model_executor/layers/mhc.py（模块 模型层；类别 source；类型 configuration；符号 forward_hip）：清理嵌套编译：移除 @torch.compile 装饰器和相关导入，避免 CUDA 图启动时参数丢失。

关键符号：_copy_ragged_to_graph_buffers, DeepseekV4ROCMAiterMLASparseMetadataBuilder.init, DeepseekV4ROCMAiterSparseSWAMetadataBuilder.init, DeepseekV4ROCMAiterMLASparseMetadataBuilder.build, MHCHeadOp.forward_hip

关键源码片段

vllm/v1/attention/backends/mla/rocm_aiter_mla_sparse_dsv4.py

核心改动：新增 _copy_ragged_to_graph_buffers 函数和元数据构建器的 __init__ 方法，为 FULL_AND_PIECEWISE CUDA 图提供稳定的 tensor 地址。

```
# vllm/v1/attention/backends/mla/rocm_aiter_mla_sparse_dsv4.py

def _copy_ragged_to_graph_buffers(
    ragged_indices: torch.Tensor,
    ragged_indptr: torch.Tensor,
    ragged_indices_buffer: torch.Tensor,
    ragged_indptr_buffer: torch.Tensor,
    num_rows: int,
    max_entries_per_row: int,
) -> tuple[torch.Tensor, torch.Tensor]:
    """将动态 ragged 元数据拷贝到持久 CUDA 图 buffer 中。

    FULL decode 图会捕获 kernel 参数地址，因此需要将 tensor 存放在
    稳定存储中，indptr 继续用于边界读取。
    使用 non_blocking 拷贝，避免 host 同步开销。
    """
    indptr_out = ragged_indptr_buffer[:num_rows + 1]
    indptr_out.copy_(ragged_indptr, non_blocking=True)

    max_entries = max(num_rows * max_entries_per_row, 1)
    ragged_out = ragged_indices_buffer[:max_entries]
    nnz = ragged_indices.numel()
    if nnz > 0:
        ragged_out[:nnz].copy_(ragged_indices, non_blocking=True)
    # 返回固定大小视图（动态切片可能导致图重捕获，但确保地址稳定）
    return ragged_out, indptr_out


class DeepseekV4ROCMAiterMLASparseMetadataBuilder(FlashMLASparseMetadataBuilder):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.c128a_decode_topk_ragged_indices_buffer: torch.Tensor | None = None
        self.c128a_decode_topk_ragged_indptr_buffer: torch.Tensor | None = None
        if self.is_deepseek_v4 and self.compress_ratio == 128:
            max_tokens = self.vllm_config.scheduler_config.max_num_batched_tokens
```

```
            self.c128a_decode_topk_ragged_indices_buffer = torch.empty(
                max_tokens * self.c128a_max_compressed,
                dtype=torch.int32,
                device=self.device,
            )
            self.c128a_decode_topk_ragged_indptr_buffer = torch.empty(
                max_tokens + 1, dtype=torch.int32, device=self.device
            )

        def build(self, common_prefix_len, common_attn_metadata, fast_build=False):
            base = super().build(...)
            ragged_indices, ragged_indptr = None, None
            dense_decode = base.c128a_global_decode_topk_indices
            decode_lens = base.c128a_decode_topk_lens
            if dense_decode is not None and decode_lens is not None:
                # 先构造密集→稀疏转换
                ragged_indices, ragged_indptr = build_ragged_indices_from_dense(
                    dense_decode.reshape(dense_decode.shape[0], -1), decode_lens
                )
                # 再拷贝到持久 buffer 以稳定图地址
                ragged_indices, ragged_indptr = _copy_ragged_to_graph_buffers(
                    ragged_indices, ragged_indptr,
                    self.c128a_decode_topk_ragged_indices_buffer,
                    self.c128a_decode_topk_ragged_indptr_buffer,
                    dense_decode.shape[0],
                    self.c128a_max_compressed,
                )
            return DeepseekV4ROCMAiterMLASparseMetadata(
                **vars(base),
                c128a_decode_topk_ragged_indices=ragged_indices,
                c128a_decode_topk_ragged_indptr=ragged_indptr,
            )
```

vllm/model_executor/layers/mhc.py

清理嵌套编译：移除 @torch.compile 装饰器和相关导入，避免 CUDA 图启动时参数丢失。

```
# vllm/model_executor/layers/mhc.py
# 移除：from vllm.platforms import current_platform

class MHCHeadOp(CustomOp):
    # ...

    def forward_hip(self, hidden_states, hc_fn, hc_scale, hc_base, rms_norm_eps, hc_eps):
        # 原装饰器 @torch.compile(backend=current_platform.simple_compile_backend)
        # 已被移除，避免嵌套 Dynamo 编译导致参数 hc_eps 丢失
        hc_mult, hidden_size = hidden_states.shape[-2:]
        outer_shape = hidden_states.shape[:-2]
        hs_flat = hidden_states.view(-1, hc_mult, hidden_size)
        num_tokens = hs_flat.shape[0]
```

```
    out = torch.empty(num_tokens, hidden_size, dtype=torch.bfloat16, device=hidden_states.
    device)
    torch.ops.vllm.hc_head_triton(
        hs_flat, hc_fn, hc_scale, hc_base, out,
        hidden_size, rms_norm_eps, hc_eps, hc_mult,
    )
    return out.view(*outer_shape, hidden_size)
```

评论区精华

1. 动态形状与 CUDA 图稳定性（gemini-code-assist[bot]）：指出 _copy_ragged_to_graph_buffers 中使用 .item() 会导致 host-device 同步，且动态 nnz 切片会导致图重捕获。但最终代码仍然保留了该实现，可能接受了一定的性能代价。
2. mhc.py 变更与图模式关系（tjtanaa）：询问移除导入和装饰器与 CUDA 图启用的关系。作者未直接回应，但 commit 消息解释：避免嵌套 Dynamo 编译导致 hc_eps 参数丢失。
3. 准确率下降问题（jeejeelee, tjtanaa）：jeejeelee 指出 GSM8K 结果偏低；tjtanaa 回应低并发（如 8）时准确率正常，高并发时略有下降，后续需关注 FP8/FNUZ 处理和 UE8M0 scale 处理。

- 动态形状导致 CUDA 图重捕获 (performance): 最终代码保留原实现，可能接受了性能代价或后续优化。
- mhc.py 变更与图模式的关系 (design): Commit 消息说明避免嵌套 Dynamo 编译导致 hc_eps 参数丢失。
- 高并发下 GSM8K 准确率偏低 (correctness): 明确问题：高并发准确率下降，后续需在 FP8/FNUZ 处理和 UE8M0 scale 上改进。

风险与影响

- 风险：
  1. 高并发准确率下降：PR 描述中 GSM8K 准确率在高并发时偏低（93.7% 对比可能更高的期望），可能与 CUDA 图重放或内核调度有关。
  2. CUDA 图动态形状稳定性：_copy_ragged_to_graph_buffers 中的 ragged_out[:nnz] 是动态形状切片，可能导致 CUDA 图缓存未命中或重捕获，降低性能收益。
  3. 缺少测试覆盖：没有针对新 buffer 逻辑的单元测试，依赖人工验证。
  4. 平台特异性：ROCm 专用改动（AITER、triton 后端），可能在 NVIDIA 上未充分验证，但标签包括 nvidia。 - 影响：影响范围限于 DeepSeek-V4 Pro 模型在 ROCm 平台上的推理性能，主要提升 decode 阶段的吞吐量。低并发准确率保持正常，高并发略有下降但仍在可接受范围。对系统其他模型无影响。团队需关注后续 FP8/FNUZ 兼容性改进。 - 风险标记：高并发准确率下降，动态形状导致 CUDA 图重捕获，缺少测试覆盖，平台特异性

关联脉络

- PR #42444 [Model Runner V2][Bug Fix][DSV4] Ensure lazy attention state initializations happen during cudagraph capture: 同属 DeepSeek-V4 的 CUDA 图相关

修复，解决惰性初始化与图捕获的兼容性问题。

- PR #42258 [Core][DSV4] Skip caching SWA blocks that can never serve a prefix-cache hit: DSV4 的 prefix caching 优化，与本 PR 的 CUDA 图模式共同提升推理性能。