

PR #2038 完整报告

radixark/miles

[feat] Enable THD packed-sequence training for DeepSeek-V4-Flash, with context parallelism

合并时间: 2026-09-09 13:04

原文链接: <http://prhub.com.cn/radixark/miles/pull/2038>

执行摘要

- 一句话: 为 DeepSeek-V4 注意力栈启用 THD packed-sequence 训练, 支持上下文并行。
- 推荐动作: 该 PR 是一个重要的功能增强, 为 DeepSeek-V4 模型在 AMD 硬件上启用了高效的 THD 训练路径。建议精读 `miles_plugins/models/deepseek_v4/ops/thd_utils.py` 中的索引辅助函数实现和 `deepseek_v4.py` 中的 THD 前向传播路径, 这些是理解 THD 支持的关键。同时关注 `indexer replay` 兼容性问题的后续修复。该 PR 的测试覆盖全面, 包括单元测试、分布式正确性测试和端到端精度对比测试。

功能与动机

DeepSeek-V4 注意力内核是基于索引驱动的, 其 THD 支持需要重建打包序列的索引和旋转位置。原有 BSHD 逻辑在 THD 下存在四个问题:

- 1) 滑动窗口夹紧到流位置 0 而非当前段起始, 导致跨样本注意力;
- 2) 旋转位置使用流宽 `orange` 而非在每个段重启;
- 3) 压缩器分组未尊重段边界;
- 4) V4 索引器对完整压缩流评分而非仅对查询自身段评分。此外, 上下文并行引入了压缩组跨越排名边界的问题。

实现拆解

1. 新增 THD 工具层: 在 `miles_plugins/models/deepseek_v4/ops/thd_utils.py` 中引入 `ThdLayout` 数据类和一组索引辅助函数 (`get_q_positions_thd`, `get_window_topk_idxes_thd`, `get_compress_cu_seqlens_thd` 等), 这些函数从 `cu_seqlens` 派生段内位置, 确保注意力窗口和压缩索引不会跨越段边界。
2. 改造主注意力前向传播: 在 `deepseek_v4.py` 的 `DeepSeekV4Attention.forward` 中, 通过 `ThdLayout.from_packed_seq_params` 检测 THD 格式, 并在 `thd_layout is not None` 时执行 THD 路径: 计算段内旋转位置、交换 CP 边界隐藏状态、执行压缩器输入紧凑化和行映射。
3. 扩展压缩器和索引器: 在 `compressor.py` 中新增 `_forward_thd` 方法, 处理打包输入的分组和重叠变换; 在 `v4_indexer.py` 中扩展 `forward` 方法以接受 `thd_layout` 参数, 并使用 `get_compress_cu_seqlens_thd` 计算每个查询的压缩行范围。
4. 启用 AMD 脚本: 在 `scripts/amd/run_deepseek_v4.py` 中将 `--qkv-format` 从 `bshd` 切换为 `thd`, 并添加关于 CP 标志的注释。
5. 全面测试覆盖: 新增三个测试文件: `test_v4_thd_compressor.py` (压缩器 THD 路径)、`test_dsv4_thd.py` (索引逻辑 CPU 单元测试) 和 `test_dsv4_thd_cp_correctness.py` (CP

分布式正确性测试）， 以及一个端到端精度对比测试 `test_dsv4_bshd_thd_rollout_parity.py`。

关键文件：

- `miles_plugins/models/deepseek_v4/ops/thd_utils.py`（模块 索引工具层；类别 `infra`；类型 `infrastructure`；符号 `ThdLayout`, `from_packed_seq_params`, `batch_of_row`, `compressed_cu_seqlens`）：新增的 THD 工具模块，包含 `ThdLayout` 数据类和所有段内索引辅助函数，是 THD 支持的核心基础设施。
- `miles_plugins/models/deepseek_v4/deepseek_v4.py`（模块 注意力前向；类别 `source`；类型 `data-contract`）：DeepSeek-V4 注意力的主前向传播，集成 THD 路径，是功能的核心集成点。
- `miles_plugins/models/deepseek_v4/ops/compressor.py`（模块 压缩器；类别 `infra`；类型 `infrastructure`；符号 `_overlap_transform_thd`, `forward`, `_forward_thd`）：DeepSeek-V4 压缩器，新增 `_forward_thd` 方法以处理打包输入的分组和重叠变换。

关键符号：`ThdLayout.from_packed_seq_params`, `get_window_topk_idxes_thd`, `get_compress_topk_idxes_thd`, `get_compress_cu_seqlens_thd`, `compressed_cu_seqlens`, `batch_of_row`, `to_rank_major_rows`, `DeepSeekV4Compressor._forward_thd`, `_overlap_transform_thd`, `V4Indexer.forward`

关键源码片段

`miles_plugins/models/deepseek_v4/ops/thd_utils.py`

新增的 THD 工具模块，包含 `ThdLayout` 数据类和所有段内索引辅助函数，是 THD 支持的核心基础设施。

```
# 从 miles_plugins/models/deepseek_v4/ops/thd_utils.py 整理
@dataclass
class ThdLayout:
    """描述当前 rank 的打包流布局；None 表示未打包格式。

    前三个字段来自打包序列参数，其余字段在前向传播中填充：
    cu_seqlens_compressed 在调用压缩器之前计算，
    而 CP 下的紧凑化字段仅在压缩组可能跨越拆分时填充。
    """
    cu_seqlens: Tensor
    global_start: int
    max_seqlen: int
    hidden_compact: Tensor | None = None
    compressed_group_ids: Tensor | None = None
    seq_to_rank_row: Tensor | None = None
    cu_seqlens_compressed: Tensor | None = None

    @classmethod
    def from_packed_seq_params(cls, packed_seq_params, *, cp_rank: int, seqlen_local: int):
        """从打包序列参数创建布局；对于任何非 thd 格式返回 None。"""
        if packed_seq_params is None or packed_seq_params.qkv_format != "thd":
```

```

        return None
    return cls(
        cu_seqlens=packed_seq_params.cu_seqlens_q,
        # CP 将打包流连续拆分，因此此 rank 的行从这里全局开始。
        global_start=cp_rank * seqlen_local,
        max_seqlen=packed_seq_params.max_seqlen_q,
    )

```

从 miles_plugins/models/deepseek_v4/ops/thd_utils.py 整理

```

def get_window_topk_idxes_thd(
    cu_seqlens: Tensor, *, window_size: int, total_tokens: int, global_start: int = 0
) -> Tensor:
    """获取打包流的窗口 topk 索引。

```

窗口被夹紧到查询自身段的起始位置，因此不会像流宽夹紧到 0 那样进入前一个段。
在 CP 下，窗口可能到达另一个 rank 产生的行，这些行已被 KV all-gather 交付。

返回：

```

    [1, total_tokens, window_size] token 索引，超过窗口的为 -1。
    """
    device = cu_seqlens.device
    batch_ids = batch_of_row(cu_seqlens, total_tokens, global_start)
    token_idx = torch.arange(total_tokens, device=device) + global_start
    window_start = torch.maximum(token_idx - window_size + 1, cu_seqlens[batch_ids])
    k_pos = window_start.unsqueeze(1) + torch.arange(window_size, device=device)
    topk_idxes = torch.where(k_pos > token_idx.unsqueeze(1), -1, k_pos)
    return topk_idxes.unsqueeze(0)

```

miles_plugins/models/deepseek_v4/deepseek_v4.py

DeepSeek-V4 注意力的主前向传播，集成 THD 路径，是功能的核心集成点。

从 miles_plugins/models/deepseek_v4/deepseek_v4.py 整理

```

def forward(
    self,
    hidden_states: torch.Tensor,
    attention_mask=None,
    inference_context=None,
    rotary_pos_emb=None,
    rotary_pos_cos=None,
    rotary_pos_sin=None,
    rotary_pos_cos_sin=None,
    attention_bias=None,
    packed_seq_params=None,
    sequence_len_offset=None,
) -> tuple[torch.Tensor, None]:
    if self.sequence_parallel:
        hidden_states = gather_from_sequence_parallel_region(
            hidden_states, tensor_parallel_output_grad=False, group=self.tp_group
        )

```

```

x = einops.rearrange(hidden_states, "s b d -> b s d")

bsz, seqlen_local, _ = x.size()
thd_layout = ThdLayout.from_packed_seq_params(
    packed_seq_params, cp_rank=self.cp_rank, seqlen_local=seqlen_local
)
rope_base = self.config.csa_compress_rotary_base if self.compress_ratio else self.config.
rotary_base
freqs_cis = wrapped_precompute_freqs_cis(
    self.config, self.rope_head_dim, rope_base, not self.compress_ratio, seqlen_local * self.
    cp_size, x.device
)
if thd_layout is None:
    freqs_cis = get_freqs_cis_for_cp(freqs_cis, seqlen_local, self.cp_size, self.cp_group)
else:
    # 打包位置在每个段边界重启并索引整个表,
    # 因此在此处选择 rank 的行, 而不是先切片。
    freqs_cis = freqs_cis.index_select(
        0, get_q_positions_thd(thd_layout.cu_seqlens, seqlen_local, thd_layout.global_start)
    )
# ... ( 省略中间计算 )

kv_compress = None
if self.compress_ratio:
    # 在索引器之前, 后者重用 CP 紧凑化和行映射。
    x_sbd = einops.rearrange(x, "b s d -> s b d")
    if thd_layout is not None:
        thd_layout.cu_seqlens_compressed = compressed_cu_seqlens(thd_layout.cu_seqlens,
            ratio)
        if self.cp_size > 1:
            # 一个组可能跨越 CP 拆分, 因此拉取前一个 rank 拥有的行
            # 并将此 rank 产生的组紧凑化为固定容量的槽位。
            boundary = exchange_cp_boundary_hidden(x_sbd, ratio=ratio, cp_group=self.cp_
                group)
            c_cap = compact_group_capacity(seqlen_local, ratio)
            thd_layout.hidden_compact, thd_layout.compressed_group_ids =
                CompressorInputCompact.apply(
                    x_sbd, boundary, thd_layout.cu_seqlens, thd_layout.global_start, ratio, c_cap
                )
            thd_layout.seq_to_rank_row = compressed_rank_layout(
                thd_layout.cu_seqlens,
                thd_layout.cu_seqlens_compressed,
                l_local=seqlen_local,
                cp_size=self.cp_size,
                ratio=ratio,
                c_cap=c_cap,
            )
            x_sbd = thd_layout.hidden_compact
        kv_compress_sbd = self.core_attention.compressor(x_sbd, thd_layout)

```

```
if kv_compress_sbd is not None:
    kv_compress = einops.rearrange(kv_compress_sbd, "s b d -> b s d")
```

miles_plugins/models/deepseek_v4/ops/compressor.py

DeepSeek-V4 压缩器，新增 `_forward_thd` 方法以处理打包输入的分组和重叠变换。

```
# 从 miles_plugins/models/deepseek_v4/ops/compressor.py 整理
def _overlap_transform_thd(
    tensor: torch.Tensor, *, compress_ratio: int, head_dim: int, is_first: torch.Tensor, value=0
) -> Tensor:
    """打包 THD 组的重叠变换。
```

与 `:func:`_overlap_transform`` 相同的重排，但组沿 dim 0 扁平化，且段的第一个组没有前驱可拉取。

参数:

```
    tensor: ``[total_comp, ratio, batch, 2 * head_dim]``
    is_first: ``[total_comp]`` bool，每个段第一个组为 True。
    """
    total_comp, ratio, bsz, _ = tensor.size()
    new_tensor = tensor.new_full((total_comp, 2 * ratio, bsz, head_dim), value)
    new_tensor[:, ratio:] = tensor[:, :, :, head_dim:]
    prev = torch.roll(tensor[:, :, :, :head_dim], shifts=1, dims=0)
    prev[is_first] = value
    new_tensor[:, :ratio] = prev
    return new_tensor
```

```
class DeepSeekV4Compressor(nn.Module):
```

```
    # ... ( 省略初始化 )
```

```
    def _forward_thd(self, x: torch.Tensor, thd_layout: ThdLayout):
```

```
        """压缩 THD 打包流，在每个段内分组 token。
```

段末尾的 `seqlen % compress_ratio` 个 token 不产生压缩条目，依赖滑动窗口。这与推理行为匹配，避免了训练/推理不匹配。

参数:

```
    x: [total, batch, dim] 打包 SBHD 布局，或 [c_cap * ratio, batch, dim]
        当布局携带 compressed_group_ids 时已分组。
    thd_layout: 打包流布局。
```

返回:

```
    k [total_comp, batch, head_dim]，当没有段达到 compress_ratio 时返回 None。
    """
```

```
    cu_seqlens = thd_layout.cu_seqlens
    compressed_group_ids = thd_layout.compressed_group_ids
    max_seqlen = thd_layout.max_seqlen
```

```
    # ... ( 省略权重断言 )
```

```
    total_tokens = x.size(0)
```

```
ratio, overlap = self.compress_ratio, self.overlap
dtype = x.dtype
```

```
pre_grouped = compressed_group_ids is not None
```

```
if pre_grouped:
```

```
    if max_seqlen is None:
```

```
        raise ValueError(
```

```
            "Pre-grouped compressor input needs max_seqlen: its group ids address "
```

```
            "positions past the compacted row count."
```

```
        )
```

```
    local_pos = compressed_group_ids
```

```
    total_comp = local_pos.size(0)
```

```
else:
```

```
    cu_seqlens_compressed = compressed_cu_seqlens(cu_seqlens, ratio)
```

```
    total_comp = int(cu_seqlens_compressed[-1])
```

```
    if total_comp == 0:
```

```
        return None
```

```
kv = linear_bf16_fp32(x, self.linear_wkv.weight)
```

```
score = linear_bf16_fp32(x, self.linear_wgate.weight)
```

```
if pre_grouped:
```

```
    # 紧凑化已将行布局为 [g * ratio, (g + 1) * ratio); 无需 gather。
```

```
    kv = kv.unflatten(0, (total_comp, ratio))
```

```
    score = score.unflatten(0, (total_comp, ratio))
```

```
    self.ape.view(1, ratio, 1, -1)
```

```
else:
```

```
    batch_ids = batch_of_row(cu_seqlens_compressed, total_comp)
```

```
    local_pos = torch.arange(total_comp, device=x.device) - cu_seqlens_compressed[batch_
ids]
```

```
    gather_idx = (cu_seqlens[batch_ids] + local_pos * ratio).unsqueeze(1) + torch.arange(
        ratio, device=x.device
```

```
    )
```

```
    kv = kv[gather_idx]
```

```
    score = score[gather_idx]
```

```
    self.ape.view(1, ratio, 1, -1)
```

```
if overlap:
```

```
    is_first = local_pos == 0
```

```
    kv = _overlap_transform_thd(kv, compress_ratio=ratio, head_dim=self.head_dim, is_
first=is_first, value=0)
```

```
    score = _overlap_transform_thd(
```

```
        score, compress_ratio=ratio, head_dim=self.head_dim, is_first=is_first, value=float("-
inf"),
```

```
    )
```

```
# ... ( 省略后续处理 )
```

评论区精华

1. 审查者 yueming-yuan 指出 indexer replay 兼容性问题: Under THD the V4 indexer's top-k indices become stream-absolute... but the R3 indexer replay still feeds per-sample compressed-group ids rebased by token offsets. 2. 关于压缩器旋转位置的讨论: yueming-yuan 建议直接使用 `apply_rotary_emb(kv.transpose(0, 1)[..., -self.rope_head_dim :], freqs_cis.index_select(0, rope_positions))` 并移除 `thd` 函数, 作者同意并已修改。
 2. 重复计算 `compressed_cu_seqlens`: yueming-yuan 指出该函数在 `forward`, `compressor` 和 `indexer` 中被重复计算, 作者通过创建子 PR 合并修复。
 3. 辅助函数命名: yueming-yuan 建议将 `get_indexer_cu_seqlens_thd` 重命名为更通用的名称, 作者将其重命名为 `get_compress_cu_seqlens_thd`。
- Indexer replay 与 THD 格式的兼容性 (correctness): 该问题已识别, 但修复范围超出本 PR (涉及 replay 契约), 将在后续 PR 中单独修复。
 - 压缩器旋转位置简化 (design): 作者同意并已实现简化。
 - 重复计算 `compressed_cu_seqlens` (performance): 通过将 `compressed_cu_seqlens` 计算提前到 `ThdLayout` 中解决, 减少重复计算。
 - 辅助函数命名 (design): 作者将其重命名为 `get_compress_cu_seqlens_thd`, 并在 `get_compress_topk_idxes_thd` 中调用它。

风险与影响

- 风险:
 1. Indexer replay 不兼容: 如 review 中指出, THD 模式下 V4 indexer 的 top-k 索引变为流绝对值, 但 R3 indexer replay 仍提供按样本偏移的压缩组 ID, 这可能导致回放索引错误。该问题已被识别但未在本 PR 中解决。
 2. 训练 / 推理不匹配: 对于每个段末尾的 `len % ratio` 个 token, THD 路径不产生压缩条目, 这与推理行为匹配, 但需要确保所有下游逻辑正确处理。
 3. 通信不平衡: 在 `CP=1` 时, THD 增加了 RCCL 时间 (+25.5%), 但通过动态批处理可以缓解。
 4. 内核编译延迟: THD 在首次使用时编译自己的内核变体, 导致第一步比 BSHD 慢得多 (CP1 THD 为 467.6s vs BSHD 251.2s)。
 - 影响:
 1. 性能提升: 在 4 节点 × 8 MI355X 集群上, THD 格式将训练 TFLOPS 提升高达 38.5%, 步时间减少高达 14.9% (CP1) 和 23.2% (CP2)。
 2. AMD 硬件支持: 通过在 AMD 脚本中启用 THD, 为 MI355X 硬件提供了优化的训练路径。
 3. 向后兼容性: 实现通过 `ThdLayout.from_packed_seq_params` 检测格式, 现有 BSHD 配置保持当前行为。
 4. 测试基础设施: 新增的测试覆盖了压缩器、索引逻辑、CP 正确性和端到端精度, 为 THD 功能提供了全面的验证。
- 风险标记: 核心路径变更, 训练 / 推理兼容性, 通信不

平衡，内核编译延迟

关联脉络

- PR #3100 Support `--colocate-memory-peak-device gpu` under LoRA: 该 PR 也涉及训练内存优化，与 THD 的性能提升目标相关，可能共同优化训练吞吐量。
- PR #3141 Dump debug train data once per DP/CP shard instead of on every rank: 该 PR 优化训练调试数据写入策略，减少磁盘占用，与 THD 的性能优化方向一致。
- PR #2854 [AMD] docker: retire the rocm700 variants, add a rocm10-mi35x variant: 该 PR 为 AMD 硬件添加新 Docker 构建变体，THD 功能在 AMD MI355X 上验证，可能共享硬件支持基础设施。