PR #27657 完整报告

sgl-project/sglang

[DeepSeek V4] CP decode opt: slice repeat attention weights to local TP partition

合并时间：2026-07-24 05:06

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/27657

执行摘要

- 一句话：CP decode 消除重复注意力权重 GEMM 冗余
- 推荐动作：建议精读。该 PR 展示了在 CP 模式下消除权重冗余的优雅设计（上下文管理器 + 延迟切片缓存），review 中也暴露了 all-reduce 组、量化尺度等关键陷阱。值得学习如何在推理引擎中安全地临时替换张量视图。

功能与动机

In DeepSeek-V4's NSA prefill context parallel mode, the attention linears (wo_a, wo_b, wq_b) are initialized with tp_size=1 and weights are repeated across all CP ranks. For CP8, this means 8× redundant attention linear computation during decode. By slicing the repeated weights to only the local TP partition (matching what normal TP8 would compute), we can eliminate this redundancy and reduce decode latency.

实现拆解

1. 新增 CpDecodeAttnTpContext 类（layers/cp/cp_decode_attn_tp.py）：单例类，管理 decode 期间注意力权重的 TP 切片。初始化时根据服务器参数和 CP 配置确定是否启用；为每个需要切片的张量（权重、量化尺度）延迟计算切片并缓存；提供 _activate 将张量替换为切片视图，_restore 恢复原始值。
2. 修改 MQALayer（models/deepseek_v4.py）：添加 _local_attn_sink 方法（从 deepseek_v4_dspark.py 移入并统一），支持 TP 分区时切片注意力下沉参数；添加 maybe_use_decode_attn_tp 上下文管理器，在 decode 时临时替换 n_local_heads、n_local_groups、attn_tp_rank、attn_tp_size 属性值为分区后的大小，并调用全局上下文对象对 wq_b、wo_a、wo_b 进行切片；prefill 时自动跳过。
3. 修改 DeepseekV2Attention（models/deepseek_v2.py）：添加类似的 maybe_use_decode_attn_tp 方法，用于 GLM 等基于 DeepseekV2 架构的模型；对 q_b_proj、o_proj 以及 KV 权重进行切片。
4. 修改 RowParallelLinear（layers/linear.py）：新增 use_decode_attn_tp 标志；当该标志为真时，即使 tp_size=1 也强制执行 all-reduce，确保每个 rank 的部分结果正确聚合。
5. 添加白名单与参数验证（server_args.py）：新增 --enable-cp-decode-attn-tp 开关；在模型初始化时验证当前架构是否在白名单 CP_DECODE_ATTN_TP_SUPPORTED_ARCHS 中，否则报错。

6. 修改 DeepseekV4ForCausalLMDSpark（deepseek_v4_dspark.py）：移除原有的 _local_attn_sink（功能已移至基类），并在 forward 中用 maybe_use_decode_attn_tp 包装注意力计算。
7. 测试与配置：暂无专门的单元测试（风险点）；需用户启用新参数并搭配 --enable-prefill-context-parallel 使用。

关键文件：

- python/sglang/srt/layers/cp/cp_decode_attn_tp.py（模块 CP 加速；类别 source；类型 core-logic；符号 get_cp_decode_attn_tp_ctx, CpDecodeAttnTpContext, init, is_enabled）：核心新增文件，实现 CpDecodeAttnTpContext 类，管理 decode 期间注意力权重的 TP 切片逻辑，包括切片缓存、激活与恢复机制。
- python/sglang/srt/models/deepseek_v4.py（模块 DeepSeekV4；类别 source；类型 data-contract；符号 _local_attn_sink, maybe_use_decode_attn_tp）：修改 DeepSeek V4 的 MQALayer，添加 _local_attn_sink 和 maybe_use_decode_attn_tp，集成 CP decode 优化。
- python/sglang/srt/models/deepseek_v2.py（模块 DeepSeekV2；类别 source；类型 data-contract；符号 maybe_use_decode_attn_tp）：为 DeepseekV2Attention 添加 maybe_use_decode_attn_tp，支持 GLM 等模型。
- python/sglang/srt/models/deepseek_v4_dspark.py（模块 DSpark；类别 source；类型 data-contract；符号 _local_attn_sink）：移除旧的 _local_attn_sink，用 maybe_use_decode_attn_tp 包装 forward。
- python/sglang/srt/server_args.py（模块 配置；类别 source；类型 configuration）：添加 --enable-cp-decode-attn-tp 参数及白名单校验。
- python/sglang/srt/layers/linear.py（模块 线性层；类别 source；类型 core-logic）：修改 RowParallelLinear，支持 use_decode_attn_tp 标志强制 all-reduce。

关键符号：CpDecodeAttnTpContext.init, CpDecodeAttnTpContext.is_enabled, CpDecodeAttnTpContext.set_decode_attn_tp, CpDecodeAttnTpContext._slice, CpDecodeAttnTpContext._activate, CpDecodeAttnTpContext._restore, MQALayer._local_attn_sink, MQALayer.maybe_use_decode_attn_tp, DeepseekV2Attention.maybe_use_decode_attn_tp

关键源码片段

python/sglang/srt/layers/cp/cp_decode_attn_tp.py

核心新增文件，实现 CpDecodeAttnTpContext 类，管理 decode 期间注意力权重的 TP 切片逻辑，包括切片缓存、激活与恢复机制。

```
"""CP Decode Attention TP context.

When CP (Context Parallel) mode sets tp_size=1 (repeat weights), decode can
partition attention weights across CP ranks matching normal TP behavior.
"""
from __future__ import annotations
import logging
```

```
from typing import TYPE_CHECKING, Dict, Optional, Tuple
import torch

logger = logging.getLogger(__name__)

if TYPE_CHECKING:
    from sglang.srt.model_executor.forward_batch_info import ForwardBatch

class CpDecodeAttnTpContext:
    """Slices replicated attention weights across CP ranks during decode."""

    def __init__(self):
        enable_attn_tp = get_server_args().enable_cp_decode_attn_tp

        # 仅在 CP size > 1 且参数启用时设置 rank/size
        if enable_attn_tp and get_parallel().attn_cp_size > 1:
            self.decode_tp_rank = get_parallel().attn_cp_rank
            self.decode_tp_size = get_parallel().attn_cp_size
            logger.info("Enable CP decode attention TP")
        else:
            self.decode_tp_rank = None
            self.decode_tp_size = None
            logger.info("Disable CP decode attention TP")

        self.use_decode_attn_tp = False # 由 set_decode_attn_tp 更新
        self._slice_cache: Dict = {} # ( 对象 id, 属性名 ) -> ( 原始张量 , 切片 , 是否为 Parameter)

    @property
    def is_enabled(self) -> bool:
        """判断是否已配置解码 TP 分区。"""
        return self.decode_tp_size is not None and self.decode_tp_size > 1

    def set_decode_attn_tp(self, forward_batch: ForwardBatch):
        """根据 batch 类型决定是否启用 TP 分区： prefill 期间跳过， decode 时激活。"""
        if not self.is_enabled:
            self.use_decode_attn_tp = False
            return
        # 跳过 prefill CP 阶段（需要全 heads） , 其他情况（主要为 decode）启用
        self.use_decode_attn_tp = not is_cp_v2_active(
            forward_batch
        ) and not dsa_use_prefill_cp(forward_batch)

    def _slice(self, tensor: torch.Tensor, dim: int) -> torch.Tensor:
        """沿 dim 维将张量等分为 decode_tp_size 份， 返回本 rank 的切片。"""
        assert dim in (0, 1)
        chunk = tensor.shape[dim] // self.decode_tp_size
        sliced = tensor.narrow(dim, self.decode_tp_rank * chunk, chunk)
        # 若 dim=1 则返回 contiguous 以便后续计算
        return sliced if dim == 0 else sliced.contiguous()
```

```
    def _activate(self, obj, attr_name: str, dim: int):
        """将对象的属性替换为 TP 切片后的视图（仅首次触发时切分并缓存）。"""
        tensor = getattr(obj, attr_name, None)
        if tensor is None:
            return
        is_param = isinstance(tensor, torch.nn.Parameter)
        raw = tensor.data if is_param else tensor
        assert isinstance(raw, torch.Tensor) and raw.dim() > dim, \
            f"CP decode attn TP: {type(obj).__name__}.{attr_name} 不可切片"
        assert raw.shape[dim] % self.decode_tp_size == 0, \
            f"CP decode attn TP: 形状 {raw.shape} 沿 dim {dim} 不能被 {self.decode_tp_size} 整除"

        cache_key = (id(obj), attr_name)
        cache = self._slice_cache.get(cache_key)
        if cache is None:
            cache = (raw, self._slice(raw, dim), is_param)
            self._slice_cache[cache_key] = cache

        # 用切片视图替换原数据
        if cache[2]:
            tensor.data = cache[1]
        else:
            setattr(obj, attr_name, cache[1])

    def _restore(self, obj, attr_name: str):
        """恢复被切片替换的属性到原始值。"""
        cache = self._slice_cache.get((id(obj), attr_name))
        if cache is None:
            return
        orig, _, is_param = cache
        if is_param:
            getattr(obj, attr_name).data = orig
        else:
            setattr(obj, attr_name, orig)
```

python/sglang/srt/models/deepseek_v4.py

修改 DeepSeek V4 的 MQALayer，添加 _local_attn_sink 和 maybe_use_decode_attn_tp，集成 CP decode 优化。

```
from contextlib import contextmanager
from sglang.srt.layers.cp.cp_decode_attn_tp import get_cp_decode_attn_tp_ctx

class MQALayer(MqaAttentionBase):
    # ... 其他部分 ...

    def _local_attn_sink(self) -> torch.Tensor:
        """返回当前 rank 的注意力下沉（attn sink）向量。
        在 TP 分区时，从完整 sink 中切片出本地 heads 对应的部分。
        """
```

```
    if self.attn_tp_size == 1:
        return self.attn_sink
    if self._attn_sink_local is None:
        rank = self.attn_tp_rank
        num_heads = self.n_local_heads
        # padding 逻辑与 decode 内核期望对齐，当 heads≤64 时 pad 到 64
        padded_num_heads = 64 if num_heads <= 64 else self.n_heads
        sink = self.attn_sink.new_zeros(padded_num_heads)
        sink[:num_heads] = self.attn_sink[
            rank * num_heads : (rank + 1) * num_heads
        ]
        self._attn_sink_local = sink
    return self._attn_sink_local

@contextmanager
def maybe_use_decode_attn_tp(self, forward_batch: ForwardBatch):
    """上下文管理器：decode 时对注意力权重进行 TP 切片，自动恢复。"""
    ctx = get_cp_decode_attn_tp_ctx()
    # 确定本次使用的注意力模块（MQA 或标准 attention）
    attn = self.attn_mqa if isinstance(self, MQALayer) else self.attn
    with ctx.maybe_use_decode_attn_tp(
        forward_batch,
        [self.wq_b, self.wo_a, self.wo_b], # 需要切片的线性层
        radix_attn=attn,
    ):
        if ctx.use_decode_attn_tp:
            # 备份原有属性
            orig = (
                self.n_local_heads,
                self.n_local_groups,
                self.attn_tp_rank,
                self.attn_tp_size,
            )
            decode_tp_size = ctx.decode_tp_size
            # 将属性临时替换为分区后的大小
            self.n_local_heads = self.n_heads // decode_tp_size
            self.n_local_groups = self.n_groups // decode_tp_size
            self.attn_tp_rank = ctx.decode_tp_rank
            self.attn_tp_size = decode_tp_size
            try:
                yield
            finally:
                # 恢复原始属性
                (
                    self.n_local_heads,
                    self.n_local_groups,
                    self.attn_tp_rank,
                    self.attn_tp_size,
                ) = orig
```

```
        else:
            yield
```

评论区精华

Review 中存在多个关键讨论：

- All-reduce 组问题（critical）：gemini-code-assist[bot] 指出当 use_decode_attn_tp 激活时，权重被分区，必须对注意力 TP 组执行 all-reduce，否则 tp_group（size=1）会导致 no-op。作者通过向 RowParallelLinear 添加 use_decode_attn_tp 标志并调整 all-reduce 条件予以修复。
- 量化尺度切片 IndexError（high）：gemini-code-assist[bot] 指出如果量化尺度是 per-tensor 标量，切片时可能越界。作者已在 _slice 中添加维度检查。
- 模型白名单：Fridge003 要求添加支持模型的白名单，防止误用。作者添加了 CP_DECODE_ATTN_TP_SUPPORTED_ARCHS 并在 server_args.py 中校验。
- 解耦 DSA 依赖：Fridge003 建议使用通用 CP 激活检查代替 DSA 特定函数。作者保留了现有检查，但通过服务器参数作为主开关。
- 动态修改属性：Fridge003 最初认为动态修改 n_local_heads 等属性过于 hacky，作者坚持但通过上下文管理器隔离副作用，最终被接受。
  - use_decode_attn_tp 时 all-reduce 组必须为注意力 TP 组 (correctness): 作者在 RowParallelLinear 中添加 use_decode_attn_tp 标志，调整 all-reduce 条件，现在当该标志为真时强制 all-reduce。
  - 切片量化尺度时检查维度存在性 (correctness): 作者在 _activate 中添加了维度检查（assert raw.dim() > dim），并仅在维度存在时切片。
  - 添加支持模型的白名单 (design): 作者添加了 CP_DECODE_ATTN_TP_SUPPORTED_ARCHS 元组，并在 server_args.py 中校验，不匹配则报错。
  - 使用通用 CP 激活检查代替 DSA 特定函数 (design): 作者保留了 is_cp_v2_active 和 dsa_use_prefill_cp，但增加了 get_server_args().enable_cp_decode_attn_tp 作为主要开关。Fridge003 最终批准。
  - 避免动态修改 n_local_heads 等属性 (design): 保留现有设计，但通过上下文管理器隔离副作用，被接受。

风险与影响

- 风险：
  - 核心路径变更：修改了 DeepSeekV2/V4 的注意力 forward 路径，任何回归都直接影响生成质量。
  - 缺少测试覆盖：PR 未包含针对 CpDecodeAttnTpContext 或模型行为的单元测试，回归风险较高。
  - 量化尺度兼容性：_slice 虽然检查了维度，但不同量化格式（如 FP8、AWQ）的 scale 形状差异可能未被完全覆盖。
  - 跨模型白名单约束：白名单限制了适用模型范围，若新模型架构类似则需手动添加，否则会误报错。

- 与其他特性互斥：未验证与 speculative decoding、DSA 的混合使用。
- 影响：对使用 DeepSeek V4 CP 的用户有显著性能提升（1.1-1.24× 中位 ITL 降低），但需显式启用新标志。白名单校验防止了误用，但维护者需关注新架构的支持。代码侵入性中等，但动态属性替换可能给后续调试带来困难。潜在影响范围限于 DeepSeek V4 和 GLM 系列。
- 风险标记：核心路径变更，缺少测试覆盖，量化尺度兼容性，跨模型白名单约束

关联脉络

- 暂无明显关联 PR