PR #29421 完整报告

sgl-project/sglang

[Feat][GLM5.2] Add DSA Cache Layer Split under Prefill CP

合并时间：2026-07-09 18:03

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/29421

执行摘要

- 一句话：DSA 缓存层分片，减少预填 CP 下每 rank 的 KV 内存
- 推荐动作：建议精读，尤其是对 ML 推理内存优化与分布式缓存设计感兴趣的工程师。PR 中关于专用 NCCL 通信器、异步广播、延迟同步的设计值得借鉴。当前主机池部分暂未完整，对于非 GLM5.2 的部署无需关注；若计划在生产环境启用，建议等待主机池改造完成后再全面验证。

功能与动机

在 CP 预填中，每个 rank 持有所有层的 KV 缓存造成大量浪费。对于 GLM5.2（78 层，cp_size=4），每 rank 的 KV 内存可从 0.77 GB 降至 0.20 GB。通过层分片，每个 rank 只存储自身拥有的层，不拥有的层通过广播获取，从而大幅节约内存。

实现拆解

1. 重构 MLA KV 写入路径（memory_pool.py）：提取 _write_mla_kv_buffer 方法，统一处理不同硬件后端的分支，避免在 set_mla_kv_buffer 中重复层索引操作。
2. 新增层分片池类（dsa_cache_layer_split.py）：LayerSplitDSATokenToKVPool 继承 DSATokenToKVPool，添加层所有权判断、层范围计算及基于专用 NCCL 通信器的广播机制；对非自有层分配零页面，读取时从拥有者广播。
3. CP 工具函数集中（layers/cp/utils.py）：新增 get_layer_shard_range、get_layer_owner、is_glm_dsa_cache_layer_split_enabled 等函数，统一提供层分片查询接口，移除散落在各模块的重复逻辑。
4. 主机池层分片感知（pool_host/base.py、memory_pool_host.py）：基类添加 _is_device_layer_sharded、_host_layer_index 等方法；load_to_device_per_layer 和新增的 _backup_from_device_per_layer 按需跳过非自有层，主机缓存索引映射到紧凑范围。
5. PD 传输适配（common/conn.py、mooncake/conn.py 等）：在 PrefillServerInfo 中传递 enable_dsa_cache_layer_split，enable_all_cp_ranks_for_transfer 逻辑合并层分片条件，确保 decode 从所有 CP rank 拉取数据。
6. 配置验证与测试（server_args.py）：添加 CLI 标志，对非 DSA 模型、decode 模式、缺少 mooncake 后端等情况报错；新增单元测试（test_dsa_layer_shard_utils.py）与端到端测试（test_dsa_glm52_cache_layer_split.py）。

关键文件：

- python/sglang/srt/mem_cache/dsa_cache_layer_split.py（模块 DSA 层分片池；类别 source；类型 dependency-wiring；符号 LayerSplitDSATokenToKVPool, init, _local_layer_idx, _owned_local_layer_range）：新增核心类 LayerSplitDSATokenToKVPool，实现层分片、所有权判断与广播逻辑，是整个特性的核心。
- python/sglang/srt/layers/cp/utils.py（模块 CP 工具集；类别 source；类型 dependency-wiring；符号 is_glm_dsa_cache_layer_split_enabled, get_glm_dsa_cp_layer_shard_info, get_glm_dsa_layer_split_effective_num_layers, get_layer_shard_range）：集中提供层分片相关的工具函数，是各模块的依赖入口，标准化了分片范围计算和所有权判断。
- python/sglang/srt/mem_cache/memory_pool.py（模块 核心缓存池；类别 source；类型 core-logic；符号 set_mla_kv_buffer, _write_mla_kv_buffer, _index_buffer_shape, _create_index_buffers）：重构了 MLA KV 写入路径，提取 _write_mla_kv_buffer 为后续层分片池复用做准备；添加 _create_index_buffers 方法分离索引缓冲区创建。
- python/sglang/srt/mem_cache/pool_host/base.py（模块 主机池基类；类别 source；类型 core-logic；符号 _is_device_layer_sharded, _device_owned_layer_range, _effective_host_layer_num, _is_device_layer_owned）：基类添加层分片感知方法，所有主机池子类（MLA、MLAHybrid）获得统一的分片判断与层索引映射。
- test/registered/unit/mem_cache/test_dsa_layer_shard_utils.py（模块 层分片测试；类别 test；类型 test-coverage；符号 TestDSALayerShardUtils, test_balanced_layer_ranges_cover_all_layers_once, test_owner_matches_uneven_layer_ranges, test_empty_tail_shards_have_empty_ranges）：单元测试覆盖层分片范围计算、所有者判定、广播回退与 pending 状态提升，验证工具函数正确性。

关键符号：LayerSplitDSATokenToKVPool.init, LayerSplitDSATokenToKVPool._broadcast_tensor_from_owner, LayerSplitDSATokenToKVPool.prefetch_kv_buffer, LayerSplitDSATokenToKVPool._get_broadcastable_kv_buffer, get_layer_shard_range, get_layer_owner, is_glm_dsa_cache_layer_split_enabled, MLATokenToKVPool._write_mla_kv_buffer, MLATokenToKVPool._create_index_buffers, BaseHostPool._is_device_layer_sharded

关键源码片段

python/sglang/srt/mem_cache/dsa_cache_layer_split.py

新增核心类 LayerSplitDSATokenToKVPool，实现层分片、所有权判断与广播逻辑，是整个特性的核心。

```
class LayerSplitDSATokenToKVPool(DSATokenToKVPool):
    """DSA KV pool that shards layers across CP ranks with owner-broadcast reads."""

    def __init__(
        self,
        *args,
        layer_shard_rank: int,
        layer_shard_size: int,
```

```
    **kwargs,
):
    assert (
        layer_shard_rank is not None and layer_shard_size > 1
    ), "LayerSplitDSATokenToKVPool requires layer_shard_size > 1"
    # CP rank 和分片总数，用于判断当前 rank 拥有的层
    self.layer_shard_rank = layer_shard_rank
    self.layer_shard_size = layer_shard_size
    self.layer_shard_enabled = True
    self.layer_broadcast_comm = None # 延迟初始化专用广播 NCCL 通信器
    super().__init__(*args, **kwargs)
    # 计算当前 rank 在图中的全局起始层号（供 PD 传输使用）
    my_start, _ = self._owned_local_layer_range()
    self.layer_shard_start = self.start_layer + my_start

# ---- 层所有权帮助方法 ----------------------------------------
def _local_layer_idx(self, layer_id: int) -> int:
    """将全局 layer_id 转换为相对于本池起始的本地索引"""
    return layer_id - self.start_layer

def _owned_local_layer_range(self) -> tuple[int, int]:
    """返回当前 rank 拥有的本地层范围 [start, end)"""
    return get_layer_shard_range(
        self.layer_shard_rank, self.layer_shard_size, self.layer_num
    )

def _is_layer_owned(self, layer_id: int) -> bool:
    """当前 rank 是否拥有 layer_id 对应的层"""
    local_idx = self._local_layer_idx(layer_id)
    owned_start, owned_end = self._owned_local_layer_range()
    return owned_start <= local_idx < owned_end

def _get_layer_owner_rank(self, layer_id: int) -> int:
    """返回 layer_id 所属的 CP rank"""
    return get_layer_owner(
        self._local_layer_idx(layer_id), self.layer_shard_size, self.layer_num
    )

# ---- 广播通信初始化 ------------------------------------------
def _init_layer_broadcast_comm(self) -> None:
    """设置基于 CP group 的专用 PyNcclCommunicator，用于层广播"""
    cp_group = get_attention_cp_group()
    if cp_group.world_size <= 1 or cp_group.pynccl_comm is None:
        return
    from sglang.srt.distributed.device_communicators.pynccl import (
        PyNcclCommunicator,
    )
    # 此处省略完整实现：基于 cp_group.ranks 创建新通信器
    # self.layer_broadcast_comm = PyNcclCommunicator(...)
```

python/sglang/srt/mem_cache/memory_pool.py

重构了 MLA KV 写入路径，提取 _write_mla_kv_buffer 为后续层分片池复用做准备；添加 _create_index_buffers 方法分离索引缓冲区创建。

```
def _write_mla_kv_buffer(
    self,
    dst_buffer: torch.Tensor,
    loc: torch.Tensor,
    cache_k_nope: torch.Tensor,
    cache_k_rope: torch.Tensor,
) -> None:
    """
    将 cache_k_nope 和 cache_k_rope 写入 dst_buffer 的指定位置 loc。
    根据硬件后端和量化配置选择不同内核路径。
    """
    if _is_hip and self.use_dsa and self.dtype == fp8_dtype:
        # HIP FP8 路径：合并 BF16/FP16 到 FP8 转换与分页 KV 写入
        set_mla_kv_buffer_triton_fp8_quant(
            dst_buffer, loc, cache_k_nope, cache_k_rope, fp8_dtype,
        )
    elif self.dsa_kv_cache_store_fp8:
        # 分离量化路径：分别对 k_nope 和 k_rope 进行 FP8 量化
        cache_k_nope_fp8, cache_k_rope_fp8 = quantize_k_cache_separate(
            cache_k_nope, cache_k_rope, self.dtype, self.store_dtype
        )
        set_mla_kv_buffer_triton(
            dst_buffer, loc, cache_k_nope_fp8, cache_k_rope_fp8,
        )
    else:
        # 普通路径：直接转换 store_dtype 后写入
        if cache_k_nope.dtype != self.store_dtype:
            cache_k_nope = cache_k_nope.to(self.store_dtype)
            cache_k_rope = cache_k_rope.view(self.store_dtype)
        set_mla_kv_buffer_triton(
            dst_buffer, loc, cache_k_nope, cache_k_rope,
        )


def set_mla_kv_buffer(
    self,
    layer: RadixAttention,
    loc: torch.Tensor,
    cache_k_nope: torch.Tensor,
    cache_k_rope: torch.Tensor,
):
    """公共入口：执行越界检查后委托 _write_mla_kv_buffer 写入本层缓存"""
    maybe_detect_oob(loc, 0, self.size + self.page_size, "set_mla_kv_buffer (MLA)")
    layer_id = layer.layer_id
```

```
        self._write_mla_kv_buffer(
            self.kv_buffer[layer_id - self.start_layer],
            loc,
            cache_k_nope,
            cache_k_rope,
        )
```

评论区精华

- 正确性：gemini-code-assist 发现 torch.distributed.broadcast 使用了组内 rank 而非全局 rank，导致多 GPU 场景下可能挂起或崩溃，修复为通过 cp_group.ranks[owner_rank] 获取全局 rank。
- 性能：同一助手指出 _get_broadcastable_index_buffer 的 _broadcast_tensor_from_owner 调用缺少 use_layer_broadcast_comm=True，未使用专用通信器，可能导致与其他 CP 操作冲突，已补充。
- 设计：Fridge003 建议将层分片工具函数统一移到 cp/utils.py，避免散落在 memory_pool.py、pool_configurator.py 等位置，作者已采纳。
- 正确性：hzh0425 指出主机池的层分片实现不完整（仍按全层分配），且 PageFirst 布局支持不完整，建议回滚相关改动；作者同意回滚，由 hzh0425 后续完善。
- 设计：ShangmingCai 建议简化 enable_all_cp_ranks_for_transfer 条件合并，作者采纳。
- 测试：Fridge003 要求添加 e2e 测试验证层分片正确性，作者基于 test_dsa_glm52_tp_mtp.py 改造添加。
  - broadcast 使用组内 rank 而非全局 rank (correctness): 修复为通过 cp_group.ranks[owner_rank] 映射为全局 rank。
  - index buffer 广播缺少 use_layer_broadcast_comm (performance): 添加 use_layer_broadcast_comm=True 参数。
  - Host pool 层分片实现不完整 (correctness): 暂时回滚主机池相关修改，由 hzh0425 后续单独 PR 完善。
  - prefetch 与 MoE A2A 的带宽竞争 (performance): Dovis01 解释 prefetch 在专用 stream 上异步执行，同步点延迟到下一 full attention 层，可与 MoE 计算重叠，不会阻塞。
  - 工具函数集中到 cp/utils.py (design): 作者采纳，完成迁移。

风险与影响

- 风险：
  - 核心路径变更：KVCache ABC 新增 layer_shard_enabled 类属性，影响所有子类；memory_pool.py 中 _write_mla_kv_buffer 的提取修改了多处调用，若分支覆盖不足可能导致 FP8 或 HIP 路径退化。
  - NCCL 通信正确性：广播实现依赖全局 rank 映射，若 cp_group.ranks 获取不正确或组初始化失败，可能导致挂起或数据错误。

- Hicache 不完整：主机池相关改动（pool_host/base.py、memory_pool_host.py）已被 hzh0425 认为不完整，回滚后 layer-split 的 host 卸载功能缺失，需后续补充。
- 后端依赖：当前仅支持 mooncake 传输后端，mori/nixl 后端使用时会主动报错；用户需确保使用 mooncake。
- 兼容性：该特性要求 --enable-prefill-cp 和 --cp-strategy interleave，且仅在 DSA MLA 模型的 prefill worker 上生效，错误组合会被静默禁用（但会告警）；PP 兼容性尚未充分验证。

- 影响：
  - 用户：GLM5.2 用户可通过 --enable-dsa-cache-layer-split 显著减少每 rank 的 KV 内存，尤其在长序列（8192 tokens）下内存节省明显。必须配合预填 CP 和 interleave 策略使用，且后端限于 mooncake。
  - 系统：增加广播通信开销，但通过专用 stream 和 NCCL 通信器实现异步，预期对延迟影响可控。对于非 DSA 模型或 decode worker，特性自动禁用，无影响。
  - 团队：需要维护 LayerSplitDSATokenToKVPool 类及新工具函数；主机池部分计划由 hzh0425 后续重构，短期保持回滚状态。
  - 风险标记：核心路径变更，NCCL 通信正确性，Hicache 不完整，后端依赖，ABC 基类修改

关联脉络

- PR #29161 Support prefill backend trtllm for CP + PD + DSA MLA: 该 PR 添加了层分片功能所需的前向依赖修复，被本 PR body 明确标记为必须依赖。
- PR #29166 Fix fail to cuda graph capture when open CPU offload: 修复 CPU offload 下 CUDA graph 捕获失败，本 PR 依赖此修复以支持完整功能。
- PR #30492 Remove old accessors from dp_attention.py: 删除了 get_attention_cp_size 等函数，本 PR 在 cp/utils.py 和 dsa_cache_layer_split.py 中仍 import 这些函数，导致 CI ImportError。