PR #32858 完整报告

sgl-project/sglang

Fix DCP KV head mapping for GQA models

合并时间：2026-08-09 05:25

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/32858

执行摘要

- 一句话：DCP 组内复制 KV head，修复 GQA 模型 DCP 精度错误
- 推荐动作：值得精读。该 PR 是 DCP（decode context parallelism）下 GQA 正确性修复的范本：get_num_kv_heads 的 dcp_size 参数化与 QKVParallelLinear 的 Q/K/V 布局解耦是两个关键设计决策，默认参数回退保证了非 DCP 路径零行为变化。建议关注三点：一是 draft 与 target 在 DCP 语义上的差异及其测试表达；二是后续 --dcp-replicate-q-proj 两种模式的落地（Q 复制后可进一步省去组内权重冗余）；三是 KV 容量随 head 复制而下降的容量规划。

功能与动机

PR body 明确指出：'DCP shards the token dimension, so every rank in a DCP group must compute its token shard with the same K/V projection. The TP-only GQA layout can assign different KV heads within one DCP group, silently corrupting the merged attention result.' 精度测试印证了该静默损坏：Qwen3.5-397B-A17B-NVFP4-V2 在 TP4/DCP4 下 Triton decode 的 GSM8K 仅 0.880，修复后提升至 0.980（TP4/DCP1 参考线为 0.990），TRTLLM MHA 后端也由 0.920 升至 0.975。修复目标是让 KV 布局与 DCP 分片语义对齐，同时不引入运行时 K/V 通信。

实现拆解

该变更分五步落地：

1. 数据契约：dcp_size 进入 KV head 计算。python/sglang/srt/configs/model_config.py 的 get_num_kv_heads(tensor_parallel_size, dcp_size=1) 改为按 tp // dcp 计算 KV 并行度，max(1, total_num_kv_heads // kv_tp_size) 保证每个 GPU 至少 1 个 head；draft 模型强制 dcp_size=1，避免 draft 的 KV 池被 DCP 二次放大。
2. 线性层解耦 Q/K/V 布局。python/sglang/srt/layers/linear.py 的 QKVParallelLinear 新增可选参数 kv_tp_rank / kv_tp_size，缺省回退到 tp_rank / tp_size，因此非 DCP 调用方行为不变；num_kv_heads、num_kv_head_replicas、GGUF 切分与量化 block-scale 加载均按 shard 类型选择：Q 用 attention TP 布局，K/V 用 KV 布局。
3. 模型层落地。python/sglang/srt/models/qwen3_5.py 的 Qwen3_5AttentionDecoderLayer 计算 kv_tp_size = attn_tp_size // attn_dcp_size、kv_tp_rank = attn_tp_rank // attn_dcp_size，并以 is_nextn 区分 draft（draft 恒为 TP

切分），将 kv_tp_rank / kv_tp_size 传入 QKVParallelLinear。

4. 缓存与池尺寸联动。python/sglang/srt/mem_cache/kv_cache_configurator.py 中 MHA/SWA/ 统一池等所有 get_num_kv_heads(attn_tp_size) 调用点改为传入 attn_dcp_size；python/sglang/srt/model_executor/pool_configurator.py 的 _compute_cell_size 同步修正 cell 尺寸与 FP8 scale buffer 计算；flashinfer_backend.py 的 backend metadata 也改为 DCP-aware。
5. 测试配套与 CI 迁移。test/registered/dcp/test_dcp_layout_unit.py 新增 _kv_head_config 与三个单测，覆盖非 draft 的 DCP KV head 数、draft 保持 TP 切分、QKVParallelLinear 在 DCP 组内复制 KV；test/registered/amd/test_qwen3p5_triton_dcp.py 重命名为 test/registered/dcp/test_qwen3p5_triton_dcp.py，拓扑改为 TP4/DCP4 并注册进 CUDA nightly；attention unittest kits 的 get_num_kv_heads stub 同步补上 dcp_size 参数，避免 TypeError。

关键文件：

- python/sglang/srt/configs/model_config.py（模块 模型配置；类别 source；类型 data-contract；符号 get_num_kv_heads）：KV head 数量计算的数据契约入口，新增 dcp_size 参数并处理 draft 特例，是所有 KV 池与后端元数据的基准。
- python/sglang/srt/layers/linear.py（模块 线性层；类别 source；类型 core-logic；符号 _load_qkv_block_scale, weight_loader, weight_loader_v2）：QKVParallelLinear 为 Q 与 K/V 解耦并行布局，权重加载、量化 scale、GGUF 全路径按 shard 类型选择 rank，是实现 KV 组内复制的核心执行点。
- python/sglang/srt/models/qwen3_5.py（模块 模型定义；类别 source；类型 data-contract；符号 Qwen3_5AttentionDecoderLayer）：Qwen3.5 模型层落地新的 KV 布局推导，并处理 is_nextn draft 的 TP 切分特例。
- python/sglang/srt/mem_cache/kv_cache_configurator.py（模块 缓存配置；类别 source；类型 core-logic；符号 KVCacheConfigurator, _init_unified_mamba_pools, _init_unified_swa_pools, _build_oot_mha_kv_pool）：所有 MHA/SWA/ 统一 KV 池的 head 数与后端 metadata 改为 DCP-aware，否则池尺寸与模型布局不一致导致分配错误。
- test/registered/dcp/test_dcp_layout_unit.py（模块 DCP 测试；类别 test；类型 test-coverage；符号 _kv_head_config, test_model_config_uses_non_dcp_tp_size_for_kv_heads, test_a_draft_keeps_kv_heads_tp_sharded_under_dcp, test_gqa_qkv_loader_replicates_kv_within_dcp_group）：新增 CPU 单测固定 DCP 下 KV head 数与权重加载布局，覆盖非 draft、draft、QKV loader 三种情况，是防回归的核心测试。
- test/registered/dcp/test_qwen3p5_triton_dcp.py（模块 DCP 集成；类别 test；类型 rename-or-move）：从 AMD-only 目录迁到通用 DCP 目录并改为 TP4/DCP4 拓扑，让四 rank 回归测试进入 CUDA nightly。

关键符号：ModelConfig.get_num_kv_heads, QKVParallelLinear.init, QKVParallelLinear.weight_loader, QKVParallelLinear.weight_loader_v2, QKVParallelLinear._load_qkv_block_scale, Qwen3_5AttentionDecoderLayer.init, KVCacheConfigurator._build_oot_mha_kv_pool, PoolConfigurator._compute_cell_size

关键源码片段

python/sglang/srt/configs/model_config.py

KV head 数量计算的数据契约入口，新增 dcp_size 参数并处理 draft 特例，是所有 KV 池与后端元数据的基准。

```
def get_num_kv_heads(self, tensor_parallel_size: int, dcp_size: int = 1) -> int:
    # 每个 GPU 上实例化的 KV head 数量。
    # DCP 按 token 分片，组内 rank 必须投影出完全相同的 K/V，
    # 因此 KV head 只在 tp // dcp 个组之间切分，组内各 rank 复制同一份。
    # draft 不参与 DCP 组，始终按 TP 切分。
    total_num_kv_heads = self.get_total_num_kv_heads()
    if self.is_draft_model:
        # 若沿用 target 的 dcp_size，DCP 会被重复计入，
        # 导致 KV 池过度分配（曾引发 CI OOM）。
        dcp_size = 1
    kv_tensor_parallel_size = tensor_parallel_size // dcp_size
    # head 数少于有效并行组数时，每个 GPU 至少保留 1 个 KV head。
    return max(1, total_num_kv_heads // kv_tensor_parallel_size)
```

python/sglang/srt/layers/linear.py

QKVParallelLinear 为 Q 与 K/V 解耦并行布局，权重加载、量化 scale、GGUF 全路径按 shard 类型选择 rank，是实现 KV 组内复制的核心执行点。

```
# QKVParallelLinear.__init__ 新增两个可选参数，缺省回退到 attention TP，
# 保证非 DCP 场景和既有调用方行为完全不变。
if kv_tp_rank is None:
    kv_tp_rank = tp_rank
if kv_tp_size is None:
    kv_tp_size = tp_size
self.kv_tp_rank, self.kv_tp_size = kv_tp_rank, kv_tp_size

# Q 仍按 attention TP 切分；K/V 按 tp // dcp 的有效组数切分，
# 因此同一 DCP 组内的 rank 会加载完全相同的 K/V 权重分片。
self.num_heads = divide(self.total_num_heads, tp_size)
if kv_tp_size >= self.total_num_kv_heads:
    self.num_kv_heads = 1
    self.num_kv_head_replicas = divide(kv_tp_size, self.total_num_kv_heads)
else:
    self.num_kv_heads = divide(self.total_num_kv_heads, kv_tp_size)
    self.num_kv_head_replicas = 1

# 权重加载按 shard 类型选择布局：q 用 attention TP rank，
# k / v 用 DCP 组内的 KV rank，保证组内各 rank 投影一致。
shard_tp_rank, shard_tp_size = (
    (self.kv_tp_rank, self.kv_tp_size)
    if loaded_shard_id in ("k", "v")
    else (self.tp_rank, self.tp_size)
)
```

```
shard_size = loaded_weight.size(output_dim) // shard_tp_size
start_idx = shard_tp_rank * shard_size
loaded_weight = loaded_weight.narrow(output_dim, start_idx, shard_size)
```

python/sglang/srt/models/qwen3_5.py

Qwen3.5 模型层落地新的 KV 布局推导，并处理 is_nextn draft 的 TP 切分特例。

```
# Qwen3.5 的 draft 会被改写为 MTP 结构（model_config._config_draft_model），
# 以 is_nextn 标记；draft 是纯 TP 切分，不参与 DCP 组内 KV 复制。
dcp_size = 1 if is_nextn else get_parallel().attn_dcp_size
# K/V 的有效并行度 = attention TP / DCP 组数；组内 rank 共享 kv_tp_rank，
# 从而用完全相同的 KV 投影计算各自分到的 token。
self.kv_tp_size = self.attn_tp_size // dcp_size
self.kv_tp_rank = self.attn_tp_rank // dcp_size
self.total_num_kv_heads = config.num_key_value_heads
if self.total_num_kv_heads >= self.kv_tp_size:
    assert self.total_num_kv_heads % self.kv_tp_size == 0
else:
    assert self.kv_tp_size % self.total_num_kv_heads == 0
self.num_kv_heads = max(1, self.total_num_kv_heads // self.kv_tp_size)

# Q 保持按 attention TP 分片；K/V 按 DCP 感知的布局实例化，
# 权重加载会依据 shard 类型自动选择对应 rank。
self.qkv_proj = QKVParallelLinear(
    config.hidden_size,
    self.head_dim,
    self.total_num_heads * (1 + self.attn_output_gate),
    self.total_num_kv_heads,
    bias=False,
    quant_config=quant_config,
    tp_rank=self.attn_tp_rank,
    tp_size=self.attn_tp_size,
    kv_tp_rank=self.kv_tp_rank,
    kv_tp_size=self.kv_tp_size,
    prefix=add_prefix('qkv_proj', prefix),
)
```

评论区精华

review 的核心交锋集中在三处：

- 数据契约形态：kpham-sgl 在 flashinfer_backend.py 上建议直接让 get_num_kv_heads() 接受可选 dcp_size（默认 1），而非在 backend 里做模型模块扫描；YAMY1234 采纳，保持既有调用方兼容。
- Q 是否复制：kpham-sgl 提出传入 DCP-aware 的 tp_rank / tp_size 即可简化接口，但 YAMY1234 指出这会连 Q 一起复制；在 Q-replicated-off 布局下 Q 仍应跨 attention TP 分片，因此保留 Q 的 TP 参数、新增 K/V 的 kv_tp_rank / kv_tp_size。kpham-sgl 认可这一方向，并指出 --dcp-replicate-q-proj（来自 Helix Parallelism #21637）是后续需要支持

的开关。

- draft 的 OOM 回归：CI 出现一致性 OOM，YAMY1234 定位为 draft worker 从 DCP-aware 配置推导本地 KV head 数导致 DCP 被重复计算、KV 池过度分配；最终由 kpham-sgl 提交 'Keep draft KV heads TP-sharded under DCP' 将 draft 的 dcp_size 强制为 1，并回退了一版更局部的修复。
  - get_num_kv_heads 增加可选 dcp_size 参数 (design): YAMY1234 采纳：ModelConfig.get_num_kv_heads() 现在接受可选 dcp_size=1，DCP-aware 调用点传 attn_dcp_size，并移除了此前对模型模块的扫描。
  - Q/K/V 布局是否用同一套 DCP-aware tp 参数 (design): 维持 kv_tp_rank/kv_tp_size 方向；kpham-sgl 认可并指出后续需支持 --dcp-replicate-q-proj 开关。
  - draft worker 的 KV 池过度分配（CI OOM） (correctness): 以 RadixAttention 实例化的 KV head 数为准；最终由 commit eef5ed1 在 get_num_kv_heads 中强制 draft 忽略 dcp_size，并保留单测防回归。
  - Q-replicated on/off 支持的后续工作 (design): YAMY1234 建议保持本 PR 聚焦精度修复，后续单独 PR 支持两种模式并做性能对比；kpham-sgl 同意并 approve。
  - 测试从 AMD-only 迁移到通用 DCP (testing): YAMY1234 提交 087f9b5 完成迁移，注册 CUDA nightly，并以 torch.version.hip 门控 ROCm 环境变量。
  - 单测命名 (style): YAMY1234 重命名为 test_gqa_qkv_loader_replicates_kv_within_dcp_group，并覆盖两个 DCP 组的 KV 分片。

风险与影响

- 风险：
  1. 数据契约改变影响面广：get_num_kv_heads 的语义从 TP-only 变为 DCP-aware，所有 KV 池、cell size、attention backend 的 head 数都会随 attn_dcp_size 变化；非 Qwen3.5 模型若组合 DCP + GQA，布局也一并改变，虽已由默认参数兼容，但尚未覆盖全部模型路径。
  2. 物理 KV 容量下降：每 token 的 KV 张量形状变大（组内复制的 head 数更多），在总字节预算不变的前提下，物理 token 容量按比例降低，高并发下可能更早触达容量上限。
  3. draft 布局易回归：draft 必须保持 TP 切分，任何调用 get_num_kv_heads(tp, dcp) 的新路径都可能重蹈过度分配；单测虽覆盖了 ModelConfig 层，但 draft worker 的实际池分配仍需依赖 RadixAttention 实例。
  4. 权重加载多分支修改：GGUF、量化 block-scale、presharded 权重三条路径都涉及 tp_rank 到 kv_tp_rank 的切换，遗漏任一分支都会造成权重错位。
  5. Q-replicated on 未实现：--dcp-replicate-q-proj 开关打开时（Kimi K3 有先例），本实现假设 Q 保持分片，两种模式并存前需避免混用。 - 影响：用户侧：DCP + GQA 大模型（如 Qwen3.5-397B）的精度从异常值恢复到接近 DCP1 参考线（GSM8K 0.880 → 0.980），且 DCP4 相对 DCP1 输出吞吐提升 39.75%、完成时长缩短 28.44%（16 GPU、TP16、CC1024 压力测试），DCP 终于可以安全提速。系统侧：KV pool 尺寸、后端 metadata、FP8 scale buffer 全部改为 DCP-aware，以复制的 KV head 为基准；总显存字节预算不变，但物理 token 容量下降，容量规划需要一并考虑。团队侧：确立

了 DCP 组内 KV 复制的统一约定，并通过单测（CPU）+ 集成（TP4/DCP4 四 rank）+ 精度 / 性能基线三层验证覆盖；测试从 AMD-only 扩展到 CUDA nightly，DCP 布局问题进入常规 CI 防护范围。 - 风险标记：核心数据契约变更，多后端联动，KV 物理容量下降，draft 布局易回归，Q 复制模式未支持

关联脉络

- PR #32795 TRTLLM MHA backend support: PR body 的精度与速度测试均以 #32795 为对照，修复后 TRTLLM MHA 后端 GSM8K 由 0.920 提升至 0.975，并与本 PR 做了 16 GPU 堆叠压测。
- PR #21637 Helix Parallelism with --dcp-replicate-q-proj: 讨论中 kpham-sgl 指出该 PR 引入 Q-replicated on/off 开关，Kimi K3 默认 off，后续需为 Qwen 支持同样行为。
- PR #32800 High-concurrency output-buffer capacity guard: PR body 说明 CC1024 压测结果包含了 #32800 及并发输出缓冲容量保护，后者不属于本 PR，解读性能数据时需注意。