PR #23893 完整报告

sgl-project/sglang

[NPU]pp support mla kv transfer

合并时间：2026-05-13 09:10

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/23893

执行摘要

- 一句话：NPU Ascend 后端支持 PP 下 MLA KV 传输
- 推荐动作：建议同后端开发人员和关注 disaggregation 模块的工程师精读，特别是 get_mla_kv_ptrs_with_pp 中的层切片算法和 setup_state_kv_args 中 NPUMLA 的处理方式，该设计从硬编码演进为结构化参数，具有参考价值。

功能与动机

NPU Ascend 后端在流水线并行（PP）模式下，MLA 的 KV 传输之前只支持 MHA，需要适配 MLA 特有的层组织方式（可能包含多个 buffer groups 和 draft 层）。该 PR 解决了 PP 场景下 MLA KV 传输的正确性和性能问题。

实现拆解

1. 新增数据结构字段：在 disaggregation/base/conn.py 的 KVArgs dataclass 中增加了 kv_buf_groups 和 total_kv_layers 两个 int 字段，用于描述 MLA buffer 分组和 decode 端总层数，注释明确标注为 NPU 专用。
2. NPU 内存池暴露状态信息：在 hardware_backend/npu/memory_pool_npu.py 的 NPUMLATokenToKVPool 中新增 get_state_buf_infos 方法，返回 index_k_buffer 的指针、大小和 item 大小。同时将 torch_npu 导入改为条件导入（if is_npu():），避免非 NPU 环境导入错误。
3. 配置流调整：在 disaggregation/utils.py 的 setup_state_kv_args 函数中增加 total_kv_layers 参数，并将 NPUMLATokenToKVPool 识别为与 NSATokenToKVPool 并列的类型。当识别到 NPUMLA 池时，不直接追加 state 组件，而是设置 kv_args.kv_buf_groups（由 kv_data_ptrs 长度除以层数计算）和 kv_args.total_kv_layers，供后续传输层切片使用。
4. 核心层映射实现：在 disaggregation/ascend/conn.py 的 AscendKVManager 中新增 get_mla_kv_ptrs_with_pp 方法，根据 prefill_start_layer、kv_buf_groups 和 total_kv_layers 将 decode 端 KV 指针列表切片，准确匹配 prefill 的层范围。在 send_kvcache 中根据 self.is_mla_backend 分支分别调用 MLA 或 MHA 的层映射逻辑。
5. 调用方适配：在 decode.py 和 prefill.py 的 _init_kv_manager 中，调用 setup_state_kv_args 时传入 total_kv_layers（值为 self.scheduler.model_config.num_hidden_layers），确保 decode 端能获知总层数。

6. 测试与配套：本次变更未包含直接新增的测试文件，但通过 CI 的 run-ci 标签进行回归覆盖。

关键文件：

- python/sglang/srt/disaggregation/ascend/conn.py（模块 通信层；类别 source；类型 core-logic；符号 get_mla_kv_ptrs_with_pp）：核心实现文件，新增 get_mla_kv_ptrs_with_pp 方法处理 MLA PP 层映射，修改 send_kvcache 分支。
- python/sglang/srt/hardware_backend/npu/memory_pool_npu.py（模块 内存池；类别 source；类型 core-logic；符号 get_state_buf_infos）：新增 get_state_buf_infos 方法暴露 index_k_buffer 信息，条件导入 torch_npu 避免非 NPU 环境报错。
- python/sglang/srt/disaggregation/utils.py（模块 配置层；类别 source；类型 dependency-wiring）：修改 setup_state_kv_args 以处理 NPUMLA 池，传递 total_kv_layers 参数，设置 kv_buf_groups。
- python/sglang/srt/disaggregation/base/conn.py（模块 数据类；类别 source；类型 core-logic）：在 KVArgs dataclass 中新增 kv_buf_groups 和 total_kv_layers 字段，作为 NPU MLA PP 传输的数据结构支持。
- python/sglang/srt/disaggregation/decode.py（模块 解码端；类别 source；类型 core-logic）：在 _init_kv_manager 中调用 setup_state_kv_args 时传入 total_kv_layers 参数。
- python/sglang/srt/disaggregation/prefill.py（模块 预填充端；类别 source；类型 core-logic）：在 _init_kv_manager 中调用 setup_state_kv_args 时传入 total_kv_layers 参数。

关键符号：get_mla_kv_ptrs_with_pp, get_state_buf_infos, setup_state_kv_args

关键源码片段

python/sglang/srt/disaggregation/ascend/conn.py

核心实现文件，新增 get_mla_kv_ptrs_with_pp 方法处理 MLA PP 层映射，修改 send_kvcache 分支。

```
# 文件：python/sglang/srt/disaggregation/ascend/conn.py
class AscendKVManager(MooncakeKVManager):
    def get_mla_kv_ptrs_with_pp(
        self, src_kv_ptrs: List[int], dst_kv_ptrs: List[int]
    ) -> Tuple[List[int], List[int], int]:
        """计算 MLA 场景下 prefill 和 decode 端的 KV 指针层映射。
        因为 decode 端可能包含比 prefill 更多的层（如 speculative 算法添加的 draft 层），
        且 MLA 使用 kv_buf_groups 分组 k_data、v_data 和可选的 index_k_data。
        通过 prefill_start_layer 和 total_kv_layers 进行切片。
        """
        start_layer = self.kv_args.prefill_start_layer
        kv_buf_groups = getattr(self.kv_args, "kv_buf_groups", 1) # 每组包含的指针数
        total_kv_layers = getattr(self.kv_args, "total_kv_layers", 0) # decode 总层数
        src_layers = len(src_kv_ptrs) // kv_buf_groups # prefill 实际层数
```

```
        # 当 decode 端启用了 speculative 算法时，KV 会比 prefill 多一层 draft，需跳过
        dst_total_layers = (
            min(len(dst_kv_ptrs) // kv_buf_groups, total_kv_layers)
            if total_kv_layers
            else len(dst_kv_ptrs) // kv_buf_groups
        )
        end_layer = start_layer + src_layers

        if src_layers == dst_total_layers:
            sliced_dst_kv_ptrs = dst_kv_ptrs
        else:
            sliced_dst_kv_ptrs = []
            for i in range(kv_buf_groups):
                layer_offset = i * dst_total_layers
                sliced_dst_kv_ptrs.extend(
                    dst_kv_ptrs[layer_offset + start_layer: layer_offset + end_layer]
                )
        layers_current_pp_stage = len(src_kv_ptrs)
        return src_kv_ptrs, sliced_dst_kv_ptrs, layers_current_pp_stage

    def send_kvcache(self, ...):
        # ... 省略上下文 ...
        if self.pp_size > 1:
            if self.is_mla_backend:
                # MLA 分支使用新方法获取映射后的指针
                src_kv_ptrs, sliced_dst_kv_ptrs, layers_current_pp_stage = (
                    self.get_mla_kv_ptrs_with_pp(self.kv_args.kv_data_ptrs, dst_kv_ptrs)
                )
                layers_params = [
                    (
                        src_kv_ptrs[layer_id],
                        sliced_dst_kv_ptrs[layer_id],
                        self.kv_args.kv_item_lens[layer_id],
                    )
                    for layer_id in range(layers_current_pp_stage)
                ]
            else:
                # MHA 分支保持原有逻辑
                ...
```

python/sglang/srt/hardware_backend/npu/memory_pool_npu.py

新增 get_state_buf_infos 方法暴露 index_k_buffer 信息，条件导入 torch_npu 避免非 NPU 环境报错。

```
# 文件：python/sglang/srt/hardware_backend/npu/memory_pool_npu.py
from sglang.srt.utils.common import is_npu

# 仅在 NPU 环境下才导入 torch_npu，避免在其他硬件后端报错
if is_npu():
```

```
    import torch_npu

class NPUMLATokenToKVPool(MLATokenToKVPool):
    # ... 其他方法 ...

    def get_state_buf_infos(self):
        """返回 index_k_buffer 的指针、大小和 item 大小，
        供 disaggregation 流程注册 state 组件使用。
        """
        data_ptrs = [self.index_k_buffer[i].data_ptr() for i in range(self.layer_num)]
        data_lens = [self.index_k_buffer[i].nbytes for i in range(self.layer_num)]
        item_lens = [self.index_k_buffer[i][0].nbytes for i in range(self.layer_num)]
        return data_ptrs, data_lens, item_lens
```

评论区精华

1. 硬编码与重构：审核者 ShangmingCai 指出早期版本使用 `kv_args.state_type == "nsa"` 来判断每层指针数的方式 "hard-coded and hacky"，作者通过引入 `kv_buf_groups` 字段重构设计，消除了硬编码。
2. 字段注释明确性：ShangmingCai 要求 KVArgs 新增字段注释中明确标注 NPU/Ascend 相关，作者已采纳并修改注释。
3. 类型归属讨论：ShangmingCai 询问 NPUMLATokenToKVPool 为何被标记为 "nsa" 类型，作者解释该池即将重构为 NSA 类型以与原 MLA 区分，ShangmingCai 表示理解。

- 硬编码设计 (design): 作者通过引入 kv_buf_groups 字段重构，消除了硬编码。
- 字段注释明确性 (style): 作者已添加注释 "Only used of npu"。
- 类型归属讨论 (design): 作者解释该池将重构为 NSA 类型以与原 MLA 区分，ShangmingCai 表示理解。

风险与影响

- 风险：
  1. 字段误用风险：kv_buf_groups 和 total_kv_layers 仅在 NPU 路径下设置和使用，若在其他后端意外触发对应逻辑，可能导致未定义行为。当前通过 isinstance 检查和条件导入隔离了风险。
  2. 层切片计算错误：get_mla_kv_ptrs_with_pp 中的切片逻辑依赖于 prefill_start_layer 和 total_kv_layers 的准确性，若配置错误（如层数不匹配）会导致 KV 传输数据错位，造成推理错误。
  3. 缺少单元测试：没有直接对应的单元测试覆盖新逻辑，依赖 CI 集成测试，可能遗漏边界情况（如 speculative 禁用或启用时的层数差异）。 - 影响：影响范围限定于 NPU Ascend 后端使用 MLA 模型并启用 PP 模式的 KV 分离传输场景。对非 NPU 后端、MHA 模型或单机非 PP 模式无任何影响。用户无需更改配置即可在 NPU 上获得正确的 MLA PP 传输能力。 - 风险标记：缺少直接单元测试，字段仅在 NPU 路径使用，层切片依赖外部配置准确性

关联脉络

- PR #24595 [NPU] use causal_conv1d_update_v2 for performance: 同为 NPU 后端性能优化，属于 NPU 路线的演进。
- PR #25076 Fix fused_moe import for non-NPU devices: 涉及 NPU 条件导入模式，与本 PR 的 torch_npu 条件导入有相似性。