PR #31931 完整报告

sgl-project/sglang

[NPU] Optimize DeepSeek-V4 performance

合并时间：2026-07-28 19:45

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/31931

执行摘要

- 一句话：NPU DSV4 PD 分离与性能优化
- 推荐动作：值得重点阅读的文件：dsv4_rope.py（缓存单例设计）、dsv4_req_to_token_pool.py（Mixin 抽象）和 dsv4_common_hooks.py（PD 适配）。建议关注 ascend_dsv4_backend.py 中 graph replay 的元数据刷新逻辑，这是性能关键路径。审查者间关于状态区分和跨文件修改的讨论体现了硬件后端设计中的取舍。

功能与动机

This PR improves DeepSeek-V4 serving performance on Ascend NPU by adding PD disaggregation and chunked prefill support, together with optimizations for the prefill and decode execution paths. All hardware-specific behavior is gated to the NPU + DeepSeek-V4 path, leaving CUDA, ROCm, and other model architectures unchanged.

实现拆解

1. PD 状态类型与传输：在 disaggregation/ascend/conn.py 中定义 AscendStateType 枚举（DSV4_SWA、C4、C128 等），扩展 MooncakeKVManager 以要求精确状态索引匹配。实现 get_pd_state_components 在 dsv4_memory_pool.py 中按固定顺序收集各 pool 的指针和长度，支持 KV 和 state 池的统一注册。在 dsv4_common_hooks.py 中新增 dsv4_state_payloads 函数构建每个状态类型的页面索引列表，用于 PD 传输。
2. ReqToTokenPool 重构支持 disagg decode：将 DSV4 NPU pool 的公共逻辑提取为 DSV4ReqToTokenTablesMixin，使其可被 DSV4NPUReqToTokenPool 和新增的 DSV4NPUDecodeReqToTokenPool 复用。后者继承自 DecodeReqToTokenPool，用于 disagg decode 路径。_dsv4_free 方法调用分配器释放 c4/c128 页面。
3. Chunked prefill 中压缩状态维护：在 ascend_dsv4_backend.py 中扩展 _build_npu_compress_metadata_prefill，利用 _extend_prefix_lens_cpu 感知前缀长度，按全局序列位置计算压缩块索引。新增 _apply_dsv4_graph_metadata 等方法支持 graph replay 模式的元数据刷新。在 swa.py 分配器中修复 NPU 上的 full_to_swa_index_mapping 更新（避免 aclnnIndexPut 对 int32 值的限制）。
4. 性能优化：添加 MoEGatingTopK（移除环境变量，默认启用）；简化 interleaved RoPE 处理，新增 Dsv4NpuRoPE 类管理 NPU 下的 cos/sin 缓存，避免 decode 时 repeat_interleave；支持 DeepEP decode 双流执行；优化 fused NPU compressor 和 graph replay 路径；修复 graph 更新线程产生的额外设备上下文等 bug。

5. 其他配套变更：在 scheduler.py 中添加 HCCL group prewarm，在 deepseek_v4.py 中集成新的 NPU RoPE 路径，在 deepep.py 中增加 ZBAL 路径兼容处理（条件传递 topk_weights），在 model_runner.py 中延迟加载 DWDP 以避免 NPU 上导入 cuda 包。

关键文件：

- python/sglang/srt/hardware_backend/npu/dsv4/dsv4_rope.py（模块 RoPE 缓存；类别 source；类型 dependency-wiring；符号 Dsv4NpuRoPE, init, for_freqs, _contig_real_imag）：新增文件，实现 NPU 专用的 interleaved RoPE cos/sin 缓存类 Dsv4NpuRoPE，通过单例和 buffer 注册避免 decode 时重复计算，是 RoPE 优化的核心。
- python/sglang/srt/hardware_backend/npu/attention/ascend_dsv4_backend.py（模块 注意力后端；类别 source；类型 core-logic；符号 _to_cpu_int_list, _extend_prefix_lens_cpu, _apply_dsv4_graph_metadata, _copy_2d_with_tail）：核心后端，实现压缩元数据构建、graph replay 元数据刷新、chunked prefill 支持等，重构幅度最大。
- python/sglang/srt/hardware_backend/npu/dsv4/dsv4_common_hooks.py（模块 公共钩子；类别 source；类型 dependency-wiring；符号 dsv4_state_payloads, empty_pages, pages, state_tail_range）：新增 dsv4_state_payloads 等函数构建 PD 状态负载，修复 disagg 路径未写入 per-req 表的问题，是 PD 传输的关键适配。
- python/sglang/srt/hardware_backend/npu/dsv4/dsv4_req_to_token_pool.py（模块 请求池；类别 source；类型 core-logic；符号 DSV4NPUReqToTokenPool, DSV4ReqToTokenTablesMixin, init, _init_dsv4_tables）：提取 DSV4ReqToTokenTablesMixin 实现 per-req 表逻辑复用，新增 DSV4NPUDecodeReqToTokenPool 支持 disagg decode，是池抽象的重构核心。
- python/sglang/srt/hardware_backend/npu/dsv4/dsv4_memory_pool.py（模块 内存池；类别 source；类型 core-logic；符号 get_contiguous_buf_infos, get_pd_state_components, kv_entry, state_entry）：添加 get_pd_state_components 方法按固定顺序收集各 pool 的指针和长度，用于 PD 注册，是状态传输的数据基础。
- python/sglang/srt/disaggregation/ascend/conn.py（模块 状态传输；类别 source；类型 core-logic；符号 AscendStateType, _requires_exact_state_index_match, _is_generic_kvcache_state_type）：定义 AscendStateType 枚举，覆盖 _requires_exact_state_index_match，统一注册所有 buffer（包括 state 池）到传输引擎，是 PD 传输的入口。
- python/sglang/srt/hardware_backend/npu/dsv4/dsv4_allocator.py（模块 分配器；类别 source；类型 core-logic；符号 _wrap_full_alloc, alloc_extend_swa_tail）：优化分配器以支持 chunked prefill 中的前缀感知分配和 state 释放路径。
- python/sglang/srt/models/deepseek_v4.py（模块 模型集成；类别 source；类型 data-contract；符号 _get_npu_rope_position_cache）：集成新的 NPU RoPE 接口，替换原有的 v4_rope_inplace_npu，并移除对 multi-stream overlap 的 NPU 禁用。

关键符号：Dsv4NpuRoPE, for_freqs, ensure_tables, DSV4ReqToTokenTablesMixin, _init_dsv4_tables, _dsv4_free, dsv4_state_payloads, write_dsv4_prealloc_tables, get_pd_state_components, AscendStateType, _requires_exact_state_index_match, _is_generic_kvcache_state_type, _build_npu_compress_metadata_prefill,

_extend_prefix_lens_cpu, _apply_dsv4_graph_metadata,
_build_dsv4_graph_replay_ctx,_refresh_graph_seq_metadata,
_refresh_graph_compress_page_tables_direct

关键源码片段

python/sglang/srt/hardware_backend/npu/dsv4/dsv4_rope.py

新增文件，实现 NPU 专用的 interleaved RoPE cos/sin 缓存类 Dsv4NpuRoPE，通过单例和 buffer 注册避免 decode 时重复计算，是 RoPE 优化的核心。

```
"""NPU interleaved RoPE cos/sin cache for DeepSeek-V4 on Ascend.

One Dsv4NpuRoPE per freqs_cis (singleton by id). Tables are built once at
init and registered as buffers on the shared rotary_emb, so model.to() moves
them and a captured aclgraph sees stable tensors; decode only does index_select.
"""
from typing import Optional
import torch


class Dsv4NpuRoPE:
    """Interleaved cos/sin tables, layout [c0,c0,c1,c1,...] / [s0,s0,s1,s1,...]."""

    # 通过 id(freqs_cis) 缓存实例，freqs_cis 是模型 buffer，与模型生命周期一致
    _instances: dict[int, "Dsv4NpuRoPE"] = {}

    def __init__(
        self, freqs_cis: torch.Tensor, rotary_emb: Optional[object] = None
    ) -> None:
        self.freqs_cis = freqs_cis
        # 将 cos/sin 注册为此模块的 buffer（None 时回退到 _tables）
        self.rotary_emb = rotary_emb
        # contiguous real/imag halves of complex freqs_cis [max_pos, rope_dim/2];
        # .real/.imag 是 stride 视图，一次性物化以避免每次 aclnnIndex 调用产生 StridedSlice
        self._real_imag: Optional[tuple[torch.Tensor, torch.Tensor]] = None
        self._tables: dict[
            tuple[torch.dtype, torch.device], tuple[torch.Tensor, torch.Tensor]
        ] = {}

    @classmethod
    def for_freqs(
        cls, freqs_cis: torch.Tensor, rotary_emb: Optional[object] = None
    ) -> "Dsv4NpuRoPE":
        # 共享已 warm-up 的 freqs_cis 的调用者可以省略 rotary_emb
        inst = cls._instances.get(id(freqs_cis))
        if inst is None or inst.freqs_cis is not freqs_cis:
            inst = cls(freqs_cis, rotary_emb)
            cls._instances[id(freqs_cis)] = inst
```

```
    return inst

def ensure_tables(
    self, dtype: torch.dtype, *, allow_build: bool = True
) -> tuple[torch.Tensor, torch.Tensor]:
    # 返回 [max_pos, rope_dim] 表。初始化时 allow_build=True;
    # decode 使用 allow_build=False（避免在 captured graph 内进行 repeat_interleave）
    expected_shape = (self.freqs_cis.shape[0], self.freqs_cis.shape[1] * 2)

    if self.rotary_emb is not None:
        cos_name, sin_name = self._buffer_names(dtype)
        cos = getattr(self.rotary_emb, cos_name, None)
        sin = getattr(self.rotary_emb, sin_name, None)
        if (
            cos is not None
            and sin is not None
            and tuple(cos.shape) == expected_shape
            and tuple(sin.shape) == expected_shape
            and cos.dtype == dtype
            and sin.dtype == dtype
            and cos.device == self.freqs_cis.device
            and sin.device == self.freqs_cis.device
        ):
            return cos, sin
    else:
        cached = self._tables.get((dtype, self.freqs_cis.device))
        if cached is not None:
            cos, sin = cached
            if (
                tuple(cos.shape) == expected_shape
                and tuple(sin.shape) == expected_shape
            ):
                return cached
    # 若不允许构建则报错，确保 decode 路径不会引入 repeat_interleave
    if not allow_build:
        raise RuntimeError(
            "NPU interleaved RoPE cache is missing in a no-build path. "
            "Initialize it before forward to keep decode free of repeat_interleave."
        )
    # 构建并注册 cos/sin 表
    real_contig, imag_contig = self._contig_real_imag()
    cos = real_contig.repeat_interleave(2, dim=-1).to(dtype=dtype).contiguous()
    sin = imag_contig.repeat_interleave(2, dim=-1).to(dtype=dtype).contiguous()
    if self.rotary_emb is not None:
        cos_name, sin_name = self._buffer_names(dtype)
        self._register_or_set_buffer(cos_name, cos)
        self._register_or_set_buffer(sin_name, sin)
    else:
        self._tables[(dtype, self.freqs_cis.device)] = (cos, sin)
```

```
        return cos, sin
```

python/sglang/srt/hardware_backend/npu/dsv4/dsv4_req_to_token_pool.py

提取 DSV4ReqToTokenTablesMixin 实现 per-req 表逻辑复用，新增 DSV4NPUDecodeReqToTokenPool 支持 disagg decode，是池抽象的重构核心。

```
class DSV4ReqToTokenTablesMixin:
    """共享 DSV4-NPU per-req 表逻辑，用于 prefill/normal 池和 disagg decode 池。
    宿主类需先调用 super().__init__(...)（以便 _alloc_size 存在），
    然后调用 self._init_dsv4_tables(...)；free 应在委托基类 free 之前调用 self._dsv4_free(req)。
    """

    def _init_dsv4_tables(
        self, max_context_len: int, device: str, enable_memory_saver: bool
    ) -> None:
        memory_saver_adapter = TorchMemorySaverAdapter.create(
            enable=enable_memory_saver
        )
        # 分配器的反向引用，在两者都存在后通过 register_dsv4_allocator 连接
        self._dsv4_allocator = None
        with memory_saver_adapter.region(GPU_MEMORY_TYPE_KV_CACHE):
            for name, cols in (
                ("req_to_token_swa", max_context_len),
                ("req_to_token_c4", max(1, max_context_len // 4)),
                ("req_to_token_c128", max(1, max_context_len // 128)),
                ("req_to_token_c4_state", max_context_len),
                ("req_to_token_c128_state", max_context_len),
            ):
                setattr(
                    self,
                    name,
                    torch.zeros(
                        (self._alloc_size, cols),
                        dtype=torch.int32,
                        device=device,
                    ),
                )

    def _dsv4_free(self, req) -> None:
        # 通过分配器的统一释放路径释放 c4/c128 页面
        if self._dsv4_allocator is not None:
            self._dsv4_allocator.free(req=req, req_to_token_pool=self)


class DSV4NPUReqToTokenPool(DSV4ReqToTokenTablesMixin, ReqToTokenPool):
    # 正常模式的池，选型由 model_runner_kv_cache_mixin 决定
    ...

class DSV4NPUDecodeReqToTokenPool(DSV4ReqToTokenTablesMixin, DecodeReqToTokenPool):
```

```
# disagg decode 模式的池，继承自 DecodeReqToTokenPool
...
```

评论区精华

- gemini-code-assist 指出 swa.py 导入和 bundle 安全检查缺失：torch_npu 在通用 allocator 中直接使用但未导入，且 ascend_dsv4_backend.py 中移除了 bundle is not None 检查可能导致 AttributeError。Talantan 回应 torch_npu 已条件导入并还原了 swa.py 改动；bundle 检查最终可能在调用侧保障。
- randgun 要求修正 deepseek_v4.py RoPE 调用：qk_rope_head_dim 应改为 qk_nope_head_dim，Talantan 已完成。
- randgun 和 iforgetmyname 要求限制 deepseek_v2.py 双流功能到 NPU decode：通过添加 is_dsv4_npu 判断，现已隔离。
- iforgetmyname 质疑 mooncake/conn.py 状态区分必要性：randgun 解释 NPU 因算子限制无法像 GPU 通过映射获取状态页面索引，必须独立分配，故保留 AscendStateType。
- iforgetmyname 要求不修改 decode.py/prefill.py：最终 PR 仍包含有限修改（通过 extra_kwargs 适配），审查者最终批准。
  - swa.py import 和 bundle None 检查 (correctness): swa.py 改动还原；bundle 检查缺失仍存在风险，但调用侧可能保证 bundle 非 None。
  - deepseek_v4.py RoPE 接口修改 (correctness): 已修复。
  - NPU 双流解码限制 (design): 已按 review 修改，添加 is_dsv4_npu 保护。
  - mooncake/conn.py 状态区分必要性 (design): 接受解释，保留 AscendStateType 及方法覆盖。
  - decode.py 和 prefill.py 修改争议 (design): 最终 PR 仍包含修改，但通过条件判断限制影响，iforgetmyname 最终批准。

风险与影响

- 风险：
  - NPU 专属代码可能影响兼容性：虽已条件隔离，但新增路径与 CUDA/ROCm 共享部分基类（如 ReqToTokenPool 和 CompressorBackendMixin），若条件判断遗漏可能污染通用路径。
  - bundle 为 None 时缺少安全检查：在 ascend_dsv4_backend.py 的 _build_npu_compress_metadata_prefill 中直接使用 bundle.out_c4_loc 等属性，若 out_cache_loc_dsv4 为 None 则崩溃。虽然调用侧可能保证，但防御性缺失。
  - 状态同步正确性依赖全局位置对齐：chunked prefill 中压缩状态索引需精确对应全局序列位置，计算逻辑在 _extend_prefix_lens_cpu 和 _build_npu_compress_metadata_prefill 中，误算可能导致错误或卡顿。
  - 测试覆盖不足：仅提及 10 个 targeted unit tests，未看到完整测试套件，风险较高。
  - 多提交协作增加回归风险：24 个提交来自 4 位作者，包含多处 merge 冲突解决，可能隐含不一致。

- 影响：影响范围：仅影响 NPU + DeepSeek-V4 配置路径。用户：在 Ascend A3 环境部署 DSV4 的用户可获得 PD 分离和 chunked prefill 带来的吞吐提升，但需按文档配置环境变量。系统：增强了 NPU 后端的完备性，为后续优化奠定基础。团队：需维护新增的 dsv4_rope.py、dsv4_req_to_token_pool.py 等模块，但设计上通过 Mixin 和条件导入保持了可维护性。
- 风险标记：NPU 专属路径，缺少测试覆盖，状态同步复杂度，bundle 安全检查潜在遗漏

关联脉络

- 暂无明显关联 PR