PR #27617 完整报告

sgl-project/sglang

[SWA] Cache full→SWA out_cache_loc per forward across attention backends

合并时间：2026-06-10 13:57

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/27617

执行摘要

- 一句话：统一所有 Attention 后端的 SWA 写位置缓存，避免每层重复翻译
- 推荐动作：值得精读。展示了如何在多后端架构中统一 KV 缓存写路径的性能优化，特别是 CUDA-graph 下预分配 buffer + copy_ 的技巧。设计决策中的 assert 防守和 clear 尾部数据是生产级代码的良好习惯。

功能与动机

PR #27091 意图在每个 forward 中只计算一次 out_cache_loc 的 full→SWA 索引转换并传递给 KV 存储，但仅对 DeepSeek-V4 路径实现；通用 SWAKVPool 路径（如 Gemma、GPT-OSS 风格混合 SWA 模型）中，它只是移除了旧的记忆化（data_ptr/numel 缓存和 invalidate_loc_cache 机制），却没有添加替代方案，导致 SWAKVPool.set_kv_buffer 在每个 SWA 层每个 forward 都重复执行 translate_loc_from_full_to_swa(out_cache_loc)（一个未缓存的 GPU gather）。本 PR 为所有写入通用 SWAKVPool 的后端完成统一缓存。

实现拆解

1. 接口变更：SWAKVPool.set_kv_buffer 不再内部执行 translate_loc_from_full_to_swa，而是接收一个预翻译的 swa_loc 参数并直接使用；对于 SWA 层，assert 检查 swa_loc 必须提供。
2. Eager 路径缓存：每个 Attention 后端在 init_forward_metadata（或对应方法）中，当检测到使用 SWAKVPool 且 out_cache_loc 非空时，调用一次 token_to_kv_pool.translate_loc_from_full_to_swa(out_cache_loc)，将结果存入 forward_metadata.swa_out_cache_loc。涉及文件：triton_backend.py、flashinfer_backend.py、flashattention_backend.py、aiter_backend.py、trtllm_mha_backend.py、torch_native_backend.py、intel_amx_backend.py、xpu_backend.py、musa_backend.py 等。
3. CUDA-graph 路径缓存：在 init_cuda_graph_state 中预分配 self.cuda_graph_swa_out_cache_loc 缓冲区（最大 token 数 × int64）。在 init_forward_metadata_out_graph 中（replay 前），用 live out_cache_loc 的翻译结果填充该缓冲区（copy_ 并清空尾部），然后将 [:n] 视图绑定到 forward_metadata.swa_out_cache_loc。例如 triton_backend.py 新增 _fill_cuda_graph_swa_out_cache_loc 方法。

4. 消费点修改：所有调用 set_kv_buffer 的位置（包括 cp_utils.cp_allgather_and_save_kv_cache、ascend_backend._cp_allgather_and_save_kv_npu、以及各后端的 forward 方法）在传递时根据 forward_metadata.swa_out_cache_loc 为 SWA 层传入 swa_loc。draft-decode 快速返回路径（不操作 SWA pool）保持原有逻辑。
5. 测试配套：新增 test/registered/attention/unittests/swa/test_swa_out_cache_loc.py，包含三个测试用例：验证 SWA 层直接使用 swa_loc、SWA 层缺失 swa_loc 触发 AssertionError、full 层忽略 swa_loc。后端 SWA 集成测试（test/registered/attention/unittests/swa/）也通过。
6. 修复多步 draft 和 CUDA-graph 填充问题：后续 commit 修复了多步 EAGLE draft-extend 中未传递 swa_loc 的问题，以及 CUDA-graph 尾部脏数据导致 GSM8K 回归的问题（通过 zero_() 清空 padded tail）。

关键文件：

- test/registered/attention/unittests/swa/test_swa_out_cache_loc.py（模块 SWA 测试；类别 test；类型 test-coverage；符号 TestSWAKVPoolSetKVBuffer, _pool_and_record, _swa_set, _full_set）：新增 CPU 单元测试，验证 SWAKVPool.set_kv_buffer 直接使用 swa_loc 和 assert 行为，是设计契约的守护者。
- python/sglang/srt/layers/attention/triton_backend.py（模块 Triton 后端；类别 source；类型 core-logic；符号 _fill_cuda_graph_swa_out_cache_loc）：核心修改文件，展示如何将 swa_out_cache_loc 嵌入 ForwardMetadata 并在 Eager/CUDA-graph 路径中缓存。新增 _fill_cuda_graph_swa_out_cache_loc 函数处理 CUDA-graph 预分配缓冲区的填充。
- python/sglang/srt/layers/attention/flashinfer_backend.py（模块 FlashInfer 后端；类别 source；类型 dependency-wiring）：FlashInfer 后端的适配，在 DecodeMetadata 和 PrefillMetadata 中添加 swa_out_cache_loc 字段，Eager/CUDA-graph 路径缓存翻译结果。
- python/sglang/srt/layers/utils/cp_utils.py（模块 CP 通信；类别 source；类型 core-logic；符号 cp_allgather_and_save_kv_cache）：核心通信函数 cp_allgather_and_save_kv_cache 签名增加 swa_loc 参数，在 SWA 池时传递预翻译位置，确保混合模型 CP 通信后正确写入。
- python/sglang/srt/hardware_backend/npu/attention/ascend_backend.py（模块 NPU 后端；类别 source；类型 dependency-wiring）：NPU 后端适配，添加 swa_out_cache_loc 字段和 CP 函数的 swa_loc 参数，确保 Ascend NPU 场景的 SWA 缓存优化。

关键符号：translate_loc_from_full_to_swa, set_kv_buffer, _fill_cuda_graph_swa_out_cache_loc, cp_allgather_and_save_kv_cache, _cp_allgather_and_save_kv_npu, init_forward_metadata, init_forward_metadata_out_graph, init_cuda_graph_state

关键源码片段

test/registered/attention/unittests/swa/test_swa_out_cache_loc.py

新增 CPU 单元测试，验证 SWAKVPool.set_kv_buffer 直接使用 swa_loc 和 assert 行为，是设计契约的守护者。

```
"""Unit coverage for SWAKVPool.set_kv_buffer with a pre-translated swa_loc.

The attention backend translates out_cache_loc once per forward and passes it
in via ``swa_loc`` (cached on its forward metadata); set_kv_buffer uses it
directly for SWA layers and asserts it is provided. The per-backend cuda-graph
buffer plumbing is covered by the backend SWA integration tests.
"""

import sys
import unittest
from pathlib import Path
from types import SimpleNamespace

import torch

from sglang.srt.mem_cache.swa_memory_pool import SWAKVPool
from sglang.test.test_utils import CustomTestCase

sys.path.insert(0, str(Path(__file__).resolve().parents[1]))

from sglang.test.ci.ci_register import register_cpu_ci

register_cpu_ci(est_time=5, suite="base-a-test-cpu")


class TestSWAKVPoolSetKVBuffer(CustomTestCase):
    """set_kv_buffer: SWA layers require a pre-translated swa_loc; full layers
    use loc unchanged."""

    def _pool_and_record(self):
        # 用 object.__new__ 绕过 __init__，直接设置 layers_mapping 和子池
        pool = object.__new__(SWAKVPool)
        pool.layers_mapping = {0: (0, False), 1: (0, True)} # layer 0 = full, 1 = swa
        recorded = {}

        def _swa_set(layer, loc, k, v, k_scale, v_scale, layer_id_override):
            recorded["swa_loc"] = loc

        def _full_set(layer, loc, k, v, k_scale, v_scale, layer_id_override):
            recorded["full_loc"] = loc

        # 用 SimpleNamespace 模拟子池
        pool.swa_kv_pool = SimpleNamespace(set_kv_buffer=_swa_set)
        pool.full_kv_pool = SimpleNamespace(set_kv_buffer=_full_set)
        return pool, recorded

    def test_swa_layer_uses_swa_loc_directly(self):
        # SWA 层必须使用传入的 swa_loc，而不是内部翻译
        pool, recorded = self._pool_and_record()
```

```
        swa_loc = torch.tensor([7, 8])
        pool.set_kv_buffer(
            SimpleNamespace(layer_id=1),
            torch.tensor([3, 4]),
            None,
            None,
            swa_loc=swa_loc,
        )
        self.assertIs(recorded["swa_loc"], swa_loc)

    def test_swa_layer_requires_swa_loc(self):
        # set_kv_buffer 不再内部翻译，SWA 层必须提供 swa_loc，否则 assert 失败
        pool, _ = self._pool_and_record()
        with self.assertRaises(AssertionError):
            pool.set_kv_buffer(
                SimpleNamespace(layer_id=1), torch.tensor([3, 4]), None, None
            )

    def test_full_layer_ignores_swa_loc(self):
        # Full 层即使传入 swa_loc，也应使用原 loc
        pool, recorded = self._pool_and_record()
        loc = torch.tensor([3, 4])
        pool.set_kv_buffer(
            SimpleNamespace(layer_id=0),
            loc,
            None,
            None,
            swa_loc=torch.tensor([99, 99]),
        )
        self.assertIs(recorded["full_loc"], loc)


if __name__ == "__main__":
    unittest.main()
```

python/sglang/srt/layers/attention/triton_backend.py

核心修改文件，展示如何将 swa_out_cache_loc 嵌入 ForwardMetadata 并在 Eager/CUDA-graph 路径中缓存。新增 _fill_cuda_graph_swa_out_cache_loc 函数处理 CUDA-graph 预分配缓冲区的填充。

```
# 文件：python/sglang/srt/layers/attention/triton_backend.py
# 关键变更：ForwardMetadata 新增字段 + CUDA-graph 预分配缓冲填充

@dataclass
class ForwardMetadata:
    attn_logits: torch.Tensor
    attn_lse: torch.Tensor
    max_extend_len: int
    num_kv_splits: torch.Tensor
```

```
    kv_indptr: torch.Tensor
    kv_indices: torch.Tensor
    qo_indptr: torch.Tensor
    custom_mask: torch.Tensor
    mask_indptr: torch.Tensor
    # Sliding window
    window_kv_indptr: torch.Tensor
    window_kv_indices: torch.Tensor
    window_num_kv_splits: torch.Tensor
    window_kv_offsets: torch.Tensor
    # Separate attn_logits for SWA layers when v_head_dim differs
    swa_attn_logits: Optional[torch.Tensor] = None
    # full->SWA translated out_cache_loc (SWA KV-store write target)
    # 由 init_forward_metadata 或 CUDA-graph replay 填充，set_kv_buffer 直接使用
    swa_out_cache_loc: Optional[torch.Tensor] = None


class TritonAttnBackend(AttentionBackend):
    # ... 省略其他代码 ...

    def _fill_cuda_graph_swa_out_cache_loc(
        self, forward_batch: ForwardBatch
    ) -> Optional[torch.Tensor]:
        """Refill the SWA write-target buffer from the live out_cache_loc and
        return the [:n] view (None for non-SWA / multi-step draft).

        在 cuda-graph replay 前调用：使用 live out_cache_loc 翻译并填充到预分配缓冲区，
        然后将视图绑定到 forward_metadata，使 replay 能读取最新槽位。
        """
        if not self.use_sliding_window_kv_pool:
            return None
        out_cache_loc = forward_batch.out_cache_loc
        # 如果 out_cache_loc 为 None 或比缓冲区大，跳过（不应发生）
        if (
            out_cache_loc is None
            or out_cache_loc.shape[0] > self.cuda_graph_swa_out_cache_loc.shape[0]
        ):
            return None
        n = out_cache_loc.shape[0]
        # 清空尾部，防止前一次重放遗留的脏数据被意外读取
        self.cuda_graph_swa_out_cache_loc[n:].zero_()
        # 翻译并复制到预分配缓冲区的前 n 个元素
        self.cuda_graph_swa_out_cache_loc[:n].copy_(
            self.token_to_kv_pool.translate_loc_from_full_to_swa(out_cache_loc)
        )
        # 返回视图，forward_metadata 将引用这个视图
        return self.cuda_graph_swa_out_cache_loc[:n]
```

评论区精华

本 PR 无 Review 评论。作者在 PR body 中详细描述了准确性测试（GPT-OSS-120B + EAGLE3 平均接受长度 ~2.41）、注意力单元测试（27 passed, 55 subtests passed）和速度基准，未发现退化。

- 暂无高价值评论线程

风险与影响

- 风险：
  1. 跨后端覆盖风险：修改涉及 10+ 个后端文件，任何遗漏的后端或特殊路径（如自定义后端）未正确传递 swa_loc 将触发 AssertionError，导致服务启动失败。
  2. CUDA-graph buffer 填充风险：cuda_graph_swa_out_cache_loc 的尾部清零通过 zero_() 实现，若 max_num_tokens 估算不足可能导致越界（已有 shape[0] 检查）。
  3. 多步 draft 回归：已修复，但其他 speculative decoding 模式（如 topk>1）可能仍有隐患。
  4. 性能 Regression：非 SWA 模型不受影响；SWA 模型因消除重复翻译应获得提升，无负面性能影响。 - 影响：对 用户：无 API 变动，透明受益。对 系统：混合 SWA 模型（如 Gemma、GPT-OSS）的每 forward 计算量减少（SWA 层数越多收益越大），尤其在预填充大 batch 时更明显。对 团队：统一了 out_cache_loc 的缓存设计，未来新增后端必须遵循此模式，代码可维护性提升。 - 风险标记：跨后端一致性风险，CUDA-graph 缓冲区管理

关联脉络

- PR #27695 Bundle set_kv_buffer write targets into KVWriteLoc (loc + swa_loc): 同期对 set_kv_buffer 接口的重构，将 SWA 写位置参数合并为 KVWriteLoc 结构体，与本站点缓存设计互补。