PR #28185 完整报告

sgl-project/sglang

[GDN][KDA][mem_cache] int8 checkpoint pool for the linear-attn prefix cache

合并时间：2026-06-18 11:41

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/28185

执行摘要

- 一句话：为线性注意力模型添加 int8 压缩 checkpoint 池，使缓存前缀容量翻倍。
- 推荐动作：值得精读，特别是量化策略选择（int8 vs fp8）和策略无关的 donate 钩子设计。建议关注后续 PR #28813 中对测试文件和 CI 标签的清理。

功能与动机

线性注意力模型每序列维护一个紧凑的循环状态（state），SGLang 使用 MambaRadixCache 缓存这些状态以实现前缀复用。问题在于容量：活动的 bf16 MambaPool 大小固定，当不同的缓存前缀数量超过池大小时，复用率骤降，状态被驱逐并重算。对于多文档 RAG、多短会话等高区分前缀的工作负载，这是主要成本。本 PR 通过引入独立的 int8 checkpoint 池来解耦缓存与工作集，在相同显存下缓存约 2 倍的前缀，从而将复用崩溃的拐点向外推移 2 倍。

实现拆解

1. 新建 int8 checkpoint 池类：在 mamba_checkpoint_pool.py 中定义 Int8CheckpointStore（编码 / 解码核心）和 MambaCheckpointPool（管理池生命周期）。量化采用每 (head, k-channel) 对称 int8，scale 在 float32 中计算以避免精度损失。
2. 与 radix 缓存集成：修改 mamba_radix_cache.py 的 cache_finished_req 和 cache_unfinished_req，在插入 radix 树时通过 _commit_int8_checkpoint 将活动状态量化存储，在缓存命中时通过 copy_to_pool 去量化回 bf16。新增钩子兼容两种调度策略（no_buffer / extra_buffer）。
3. CLI 参数与配置验证：在 server_args.py 添加 --enable-int8-mamba-checkpoint 和 --int8-mamba-ckpt-size，并验证与 --enable-hierarchical-cache 及自定义 radix 缓存后端的冲突。
4. 内存池初始化：在 memory_pool.py 的 _init_mamba_pool 中根据参数创建 int8 checkpoint 池，并封装为 maybe_init_int8_mamba_checkpoint_pool 函数（移至 mamba_checkpoint_pool.py）。
5. 不变量检查与统计：更新 invariant_checker.py 添加 _check_mamba_pool_with_int8，对活动池和 int8 池独立检查；更新 pool_stats_observer.py 支持双池统计。
6. 测试配套：新增 test_int8_checkpoint_store.py 测试编解码精度、存储 / 加载往返、COW 辅助函数及内存占用；新增 test_int8_mamba_checkpoint_e2e.py 端到端测试（

Qwen3-Next 模型，验证 KL 散度）。

7. 基准测试：新增 bench_int8_checkpoint_reuse.py，通过测量多个不同前缀下的缓存命中率与吞吐量来对比 int8 与 bf16 路径。

关键文件：

- python/sglang/srt/mem_cache/mamba_checkpoint_pool.py（模块 缓存池；类别 source；类型 core-logic；符号 Int8CheckpointStore, MambaCheckpointPool, quantize, dequantize）：核心实现文件，包含 Int8CheckpointStore（量化 / 去量化编解码）和 MambaCheckpointPool（池管理与 COW 辅助函数），所有 int8 压缩逻辑于此定义。
- python/sglang/srt/mem_cache/mamba_radix_cache.py（模块 前缀缓存；类别 source；类型 core-logic；符号 int8_ckpt_pool, _alloc_int8_ckpt_slot, _commit_int8_checkpoint, _free_mamba_value）：radix 缓存集成点，修改 cache_finished_req 和 cache_unfinished_req 以支持 int8 路径，并新增 _commit_int8_checkpoint 等辅助函数。
- python/sglang/srt/server_args.py（模块 配置；类别 source；类型 configuration；符号 _handle_int8_mamba_checkpoint）：新增 CLI 参数 enable_int8_mamba_checkpoint 和 int8_mamba_ckpt_size，并包含与分层缓存 / 自定义 radix 后端的互斥检查。
- python/sglang/srt/managers/scheduler_components/invariant_checker.py（模块 校验器；类别 source；类型 core-logic；符号 _check_mamba_pool_with_int8）：新增 _check_mamba_pool_with_int8 方法，独立验证活动 bf16 池和 int8 缓存池的不变量，避免双池计数混乱。
- python/sglang/srt/mem_cache/memory_pool.py（模块 内存池；类别 source；类型 dependency-wiring）：初始化 int8 checkpoint 池的入口，根据配置创建 MambaCheckpointPool 实例并挂接到 req_to_token_pool。
- benchmark/bench_linear_attention/bench_int8_checkpoint_reuse.py（模块 基准测试；类别 bench；类型 test-coverage；符号 make_prefix, make_suffix, main, send）：提供可复现的基准测试，对比 int8 与 bf16 路径在不同前缀数量下的缓存命中率和延迟，验证 2 倍容量提升。
- test/srt/mem_cache/test_int8_checkpoint_store.py（模块 单元测试；类别 test；类型 test-coverage；符号 _rand_state, TestInt8CheckpointCodec, TestInt8CheckpointDecodeError）：单元测试覆盖编解码误差界、对称性 / 零值、store/load 往返、COW 辅助函数及内存估计准确性。
- test/registered/radix_cache/test_int8_mamba_checkpoint_e2e.py（模块 端到端测试；类别 test；类型 test-coverage；符号 TestInt8MambaCheckpointE2E, test_gsm8k）：端到端测试在实际 GDN 混合模型（Qwen3-Next）上验证 int8 checkpoint 池的启用与精度影响（KL 散度）。

关键符号：Int8CheckpointStore.quantize, Int8CheckpointStore.dequantize, Int8CheckpointStore.store, Int8CheckpointStore.load, Int8CheckpointStore.copy_to_pool, Int8CheckpointStore.store_from_pool, MambaCheckpointPool.init, MambaRadixCache._commit_int8_checkpoint, MambaRadixCache._alloc_int8_ckpt_slot, ServerArgs._handle_int8_mamba_checkpoint, InvariantChecker._check_mamba_pool_with_int8

关键源码片段

python/sglang/srt/mem_cache/mamba_checkpoint_pool.py

核心实现文件，包含 Int8CheckpointStore（量化 / 去量化编解码）和 MambaCheckpointPool（池管理与 COW 辅助函数），所有 int8 压缩逻辑于此定义。

```
class Int8CheckpointStore:
    """int8 store for cached multi-layer linear-attention states.

    Per slot: qdata int8 [L, H, d_v, d_k], scale [L, H, 1, d_k].
    Quantization occurs over d_v per (head, k-channel) — matching
    the per-k-channel decay diag(alpha) alignment.
    All math is performed in float32 to avoid precision loss when
    input is bf16/fp16.
    """

    QMAX = 127

    def __init__(self, *, num_layers, num_slots, num_heads, head_v_dim, head_k_dim, device,
    scale_dtype=torch.bfloat16):
        # Preallocate tensors on the target device
        self.qdata = torch.empty(num_layers, num_slots, num_heads, head_v_dim, head_k_dim,
        dtype=torch.int8, device=device)
        self.scale = torch.empty(num_layers, num_slots, num_heads, 1, head_k_dim, dtype=scale_
        dtype, device=device)

    @classmethod
    def quantize(cls, state: torch.Tensor):
        """symmetric per-channel quantize: state [..., H, d_v, d_k] -> (qint8, scale [..., H, 1, d_k]).
        """
        # Cast to float32 for safe accumulation
        state_fp32 = state.to(torch.float32)
        amax = state_fp32.abs().amax(dim=-2, keepdim=True).clamp(min=1e-8)
        scale = amax / cls.QMAX
        # Round and clamp to int8 range
        q = torch.round(state_fp32 / scale).clamp(-cls.QMAX - 1, cls.QMAX).to(torch.int8)
        return q, scale.to(state.dtype) # keep scale in original dtype storage

    @classmethod
    def dequantize(cls, q: torch.Tensor, scale: torch.Tensor, out_dtype: torch.dtype):
        """Dequantize: q * scale -> float32 -> out_dtype."""
        return (q.to(torch.float32) * scale.to(torch.float32)).to(out_dtype)

    def store(self, slots: torch.Tensor, state: torch.Tensor):
        """Quantize and write states for given slots."""
        slots = torch.as_tensor(slots, device=self.device)
        q, scale = self.quantize(state)
        self.qdata[:, slots] = q
        self.scale[:, slots] = scale
```

```
    def load(self, slots: torch.Tensor, out_dtype: torch.dtype):
        """Load and dequantize states from slots."""
        slots = torch.as_tensor(slots, device=self.device)
        return self.dequantize(self.qdata[:, slots], self.scale[:, slots], out_dtype)

    def store_from_pool(self, active_pool, active_slots, ckpt_slots):
        """Quantize active slotted states into checkpoint slots."""
        state = active_pool[:, active_slots].detach()
        self.store(ckpt_slots, state)

    def copy_to_pool(self, target_pool, ckpt_slots, target_slots):
        """Dequantize checkpoint into target pool slots (cache-hit COW)."""
        target_pool[:, target_slots] = self.load(ckpt_slots, target_pool.dtype)
```

注释：量化在 float32 中完成以保证精度；store_from_pool 和 copy_to_pool 是外部调用的主要 COW 入口。

python/sglang/srt/mem_cache/mamba_radix_cache.py

radix 缓存集成点，修改 cache_finished_req 和 cache_unfinished_req 以支持 int8 路径，并新增 _commit_int8_checkpoint 等辅助函数。

```
  def cache_finished_req(self, req: Req, is_insert: bool = True) -> None:
    # ... (existing logic up to mamba value selection)
    if self.enable_mamba_extra_buffer:
        mamba_ping_pong_track_buffer_to_keep = (
            self.req_to_token_pool.get_mamba_ping_pong_keep_idx(req))
        src_active = req.mamba_ping_pong_track_buffer[
            mamba_ping_pong_track_buffer_to_keep].unsqueeze(-1)
        # int8 path: quantize the active slot into the checkpoint pool
        if self.int8_ckpt_pool is not None:
            mamba_value = self._commit_int8_checkpoint(src_active)
            # After quantization, the ping-pong track buffer is no longer needed
            mamba_ping_pong_track_buffer_to_keep = None
        else:
            mamba_value = src_active.clone()
    else:
        # no_buffer path
        if self.int8_ckpt_pool is not None:
            mamba_value = self._commit_int8_checkpoint(
                req.mamba_pool_idx.unsqueeze(-1))
        else:
            mamba_value = req.mamba_pool_idx.unsqueeze(-1).clone()
        mamba_ping_pong_track_buffer_to_keep = None

    result = self.insert(InsertParams(
        key=RadixKey(token_ids[:page_aligned_len], req.extra_key),
        value=page_aligned_kv_indices,
        mamba_value=mamba_value,
        prev_prefix_len=req.cache_protected_len,
```

```
    ))
    mamba_exist = result.mamba_exist
    # If the state already existed in radix (dedup), free the redundant int8 slot
    if mamba_exist and self.int8_ckpt_pool is not None:
        self.int8_ckpt_pool.free(mamba_value)

    # With int8 checkpoints, the active mamba slot must always be returned
    free_mamba_cache = (
        True if (self.enable_mamba_extra_buffer or self.int8_ckpt_pool is not None)
        else mamba_exist
    )
    if free_mamba_cache:
        self.req_to_token_pool.free_mamba_cache(
            req, mamba_ping_pong_track_buffer_to_keep=mamba_ping_pong_track_buffer_to_keep)
    self.dec_lock_ref(req.last_node)
```

注释：关键改动在于两个 else 分支前的 int8 检查，以及在 radix 插入后对重复 int8 slot 的回收。同时强制活动 slot 必须释放。

评论区精华

- 设备放置与精度：gemini-code-assist 指出 store 方法可能因 slots 位于 CPU 导致设备不匹配，建议通过 torch.as_tensor(slots, device=self.device) 修正；量化应在 float32 中进行以避免 bf16 精度损失。作者均已采纳。
- 文件组织与 API 设计：yizhang2077 建议将 Int8CheckpointStore 合并入 mamba_checkpoint_pool.py，并添加更详细的内存估计检查；将 memory_pool.py 中过长的初始化逻辑抽出为独立函数。作者照做。
- 测试覆盖：yizhang2077 要求为混合模型（如 Qwen3.5）添加端到端测试；merrymercy 指出 test/srt 已弃用，新测试应放入 test/manual；端到端测试应标记为 extra 避免 CI 超时。作者回应已通过后续 PR #28813 处理测试位置与标签。
- Tensor 整形健壮性：gemini-code-assist 建议使用 .view(-1) 代替 .unsqueeze(0) 以兼容 0D 和 1D tensor，已采纳。
  - 设备放置与精度 (correctness): 作者均已采纳。
  - 文件组织与 API 设计 (design): 作者已将所有逻辑合并到 mamba_checkpoint_pool.py，并添加了 estimate_mem_usage_bytes 和 mem_usage_bytes 方法。
  - 测试覆盖与 CI 标签 (testing): 作者通过后续 PR #28813 处理了测试文件位置与 extra 标签。
  - Tensor 整形健壮性 (style): 已采纳。

风险与影响

- 风险：
  - 量化精度风险：int8 量化引入单次舍入误差，但设计保证误差仅在存储和加载时各出现一次，不进入循环递推。测试显示相对误差小于 1%。

    - 双池不变量一致性：活动 bf16 池与 int8 缓存池的 slot 计数耦合复杂，新增的 _check_mamba_pool_with_int8 通过独立检查两池状态降低风险，但仍有遗漏边界情况（如并发场景）的可能。
    - 兼容性限制：int8 checkpoint 池与 --enable-hierarchical-cache（主机卸载）及自定义 radix 缓存后端不兼容，已在 server_args.py 中显式抛出 ValueError。
    - 默认关闭的维护负担：功能默认关闭，但引入了新代码路径和配置项，长期维护中需确保双路径测试覆盖不退化。
- 影响：
    - 用户：为线性注意力模型用户提供可选的显存优化，在相同 HBM 下缓存前缀数量翻倍，对多前缀场景（RAG、多会话）效果明显。默认不开启，不影响现有用户。
    - 系统：增加内存池初始化和量化 / 去量化计算开销，但仅发生在缓存写入 / 命中时，不堵塞 decode 主路径。基准测试显示吞吐影响可忽略。
    - 团队：代码结构清晰，int8 相关逻辑集中在 mamba_checkpoint_pool.py，对原有调度路径改动最小。未来支持 speculative 路径时可直接复用 donate 钩子。
    - 风险标记：量化精度风险（单次舍入，误差可控），双池不变量一致性（独立校验但并发边界待观察），兼容性限制（不与 hierarchical cache 和自定义 radix cache 共用），默认关闭导致后续维护易疏漏

关联脉络

- PR #28813 Move int8 checkpoint tests under test/manual and tag as extra: 处理了本 PR 中测试文件位置和 CI 标签的 review 反馈。