# PR #34602 完整报告

sgl-project/sglang

feat(unified-memory): dense KV views for uniform-row MHA/SWA models

合并时间：2026-08-31 06:10

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/34602

## 执行摘要

- 一句话：统一池 MHA/SWA 改 dense 逐层视图，解锁非 Triton 后端前提
- 推荐动作：值得精读的架构级 PR。以下设计决策最有借鉴价值：

1. 原子切换的边界意识：视图、id 空间与尺寸计算三者必须同批切换，任何中间态都会把一套 id 写进另一套视图的行——作者把这一不变量写进 commit message 并以此组织提交顺序。
2. 布局事实收敛到 spec：tail_pad、blocks_per_page 均由 SubPoolSpec 派生，消除了

## 功能与动机

PR body 明确阐述了动机：统一内存池以 page-major 信封存储 MHA 与 SWA 的 KV，每层切片是 strided 的，所有注意力后端都必须重建 4D 视图，而实际只有 Triton 能做到——这正是统一池对这些模型族仅限 Triton 的原因。MLA 子池已经使用 dense 逐层视图，本 PR 将 MHA 与 SWA 统一到同一布局，使每层 K/V 通过一个连续张量 + 一个共享 block table 读取，这是任何其他后端读取该池的前提条件。此外，作者通过拆分声明了 PR 范围：原 write-location 重构被移出，SWA tombstone 修复已被上游 #35773 取代，系列不携带任何 speculative/ 改动。

## 实现拆解

按 5 步拆解实现过程：

1. 新增 dense MHA 视图构建器（python/sglang/srt/mem_cache/layout/page_major.py，首个 commit babe256e）：新增 build_mha_views，将 [L0_K*ps | L0_V*ps | L1_K*ps | ...] 信封视为 2L 个等宽行块的均匀数组，推导出 kernel_id(t) = (t // ps) * (ps * 2L) + t % ps，层 l 的 K 位于块 2l、V 位于块 2l+1；每个视图通过 torch.as_strided 折叠块起点为 storage_offset，成为连续 (n_rows, head_num, head_dim) 张量。此阶段为纯加法，不接入任何调用方。
2. 统一池原子切换（unified_memory_pool.py）：view_tail_pad_bytes 从外部构造参数改为 max(spec.view_tail_pad_bytes(page_size)) 派生，杜绝构造点少分配尾部填充；blocks_per_page() 下沉到 SubPoolSpec 基类（默认 1），MHA 返回 2 * layer_num、MLA 返回 layer_num；新增 unified_memory_supported_for_model（use_mla_backend or not has_asymmetric_kv）与 _assert_kernel_id_bound（kernel-facing 行数必须小于 2^31，防止撑爆 int32 read-index buffer）。所有构造函数、init_unified_swa_pools 等内部逻辑同步迁移到新 id 空间。

3. 分配器 id 空间统一（multi_ended_allocator.py）：translate_kv_loc_dense 更名为 translate_kv_loc_for_kernel，并重写为统一的页数学实现（pages/offsets 拆分 + stride = ps * kernel_page_multiplier + tombstone 钳制到 0）；kernel_page_multiplier 构造参数由默认 1 改为 None，从 spec.blocks_per_page() 派生——这直接回应评审中

关键文件：

- python/sglang/srt/mem_cache/unified_memory_pool.py（模块 统一内存池；类别 source；类型 core-logic；符号 view_tail_pad_bytes, blocks_per_page, unified_memory_supported_for_model, _assert_kernel_id_bound）：统一池的原子切换核心：view_tail_pad_bytes 改为从 spec 派生、blocks_per_page 下沉到 SubPoolSpec、新增 unified_memory_supported_for_model 与 _assert_kernel_id_bound 两道启动门禁，全部构造点同步进入 dense id 空间。
- python/sglang/srt/mem_cache/multi_ended_allocator.py（模块 分配器；类别 source；类型 core-logic；符号 translate_kv_loc_dense, translate_kv_loc_for_kernel, kernel_page_multiplier, full_v2p_page_table）：kernel-facing id 空间的中枢：translate_kv_loc_dense 更名为 translate_kv_loc_for_kernel 并统一页数学实现，kernel_page_multiplier 改为从 spec 派生，消除工厂外构造点把物理 id 写入 dense 行的风险。
- python/sglang/srt/mem_cache/layout/page_major.py（模块 布局构建；类别 source；类型 core-logic；符号 build_page_major_mha_views, build_mha_views, build_dense_mla_views, build_mla_views）：布局构建器的核心改造：build_page_major_mha_views（4D strided）替换为 build_mha_views（3D dense 逐层视图），并推导 kernel_id 寻址公式；build_dense_mla_views 更名为 build_mla_views。
- python/sglang/srt/mem_cache/memory_pool.py（模块 内存池；类别 source；类型 core-logic；符号 _store_kv_layer, _move_kv_cache_impl）：PageMajorMHATokenToKVPool 从可构造池变为显式报错的类壳，删除 strided 4D 视图的写入路径与 move_kv_cache_native 的 4D 分支，避免静默误索引。
- test/registered/unit/mem_cache/test_unified_mha_views.py（模块 视图测试；类别 test；类型 test-coverage；符号 _mha_spec, _kernel_id, _make_raw, _build_views）：新增的核心测试（531 行）：用独立的 strided 参考构建器作为 oracle，逐字节交叉验证 dense 视图寻址与信封公式一致，并覆盖 asymmetric 拒绝、tail-pad 缺失、K/V 共享 id 不混叠等关键不变量。
- python/sglang/kernels/ops/kvcache/cache_move.py（模块 KV 内核；类别 infra；类型 infrastructure；符号 store_cache_4d_kernel, store_cache_4d）：删除 store_cache_4d_kernel / store_cache_4d 共 182 行：旧的 4D strided 信封写入 kernel 随布局切换整体退役，统一池改走普通 3D 赋值路径。

关键符号：build_mha_views, build_mla_views, translate_kv_loc_for_kernel, unified_memory_supported_for_model, _assert_kernel_id_bound, SubPoolSpec.blocks_per_page, SubPoolSpec.view_tail_pad_bytes, UnifiedKVPool.init

## 关键源码片段

### python/sglang/srt/mem_cache/unified_memory_pool.py

统一池的原子切换核心：view_tail_pad_bytes 改为从 spec 派生、blocks_per_page 下沉到 SubPoolSpec、新增 unified_memory_supported_for_model 与 _assert_kernel_id_bound 两道启动门禁，全部构造点同步进入 dense id 空间。

```
# SubPoolSpec 基类新增两个由布局自描述的默认方法：

def view_tail_pad_bytes(self, page_size: int) -> int:
    """Bytes this sub-pool's views reach PAST its last page envelope."""
    return 0


def blocks_per_page(self) -> int:
    """Row-blocks one page holds in this sub-pool's kernel-facing id space.

    The page envelope is a uniform array of equally wide row-blocks, so a
    kernel-facing id is the physical page scaled by this count (见
    MultiEndedAllocator.translate_kv_loc_for_kernel). 1 表示 kernel-facing id
    就是物理 id。
    """
    return 1


@dataclass(frozen=True, kw_only=True)
class MHASubPoolSpec(SubPoolSpec):
    # ... head_num / head_dim / store_dtype / v_head_dim 字段略 ...

    def view_tail_pad_bytes(self, page_size: int) -> int:
        # 最后一个视图越过最后一页信封 (2L-1)*ps 行，需要一整个页信封的尾填充。
        return page_size * self.entry_bytes()

    def blocks_per_page(self) -> int:
        # 每页 2L 个行块：每层一个 K 块加一个 V 块，kernel-facing id 空间按此缩放。
        return 2 * self.layer_num


class UnifiedKVPool:
    def __init__(self, *, total_bytes, sub_pool_specs, device,
                 enable_memory_saver, page_size=1):
        # 尾填充不再由构造点传入，而是取所有子池 spec 的最大值；
        # 任何构造点都无法少分配视图越界所需的字节。
        self.view_tail_pad_bytes = max(
            spec.view_tail_pad_bytes(page_size) for spec in sub_pool_specs
        )
        with self.memory_saver_adapter.region(GPU_MEMORY_TYPE_KV_CACHE):
            self._raw = torch.empty(
                total_bytes + self.view_tail_pad_bytes,
                dtype=torch.uint8,
                device=device,
            )
```

## python/sglang/srt/mem_cache/multi_ended_allocator.py

kernel-facing id 空间的中枢：translate_kv_loc_dense 更名为 translate_kv_loc_for_kernel 并统一页数学实现，kernel_page_multiplier 改为从 spec 派生，消除工厂外构造点把物理 id 写入 dense 行的风险。

```
class MultiEndedAllocator(BaseTokenToKVPoolAllocator):
    def __init__(self, *, kvcache, unified_buffer, sub_pool_name, device,
                 is_id_owner, page_size=1, need_sort=False, forward_stream=None,
                 lazy_compaction=False, kernel_page_multiplier=None):
        spec = unified_buffer.spec(sub_pool_name)
        # 关键审查点：multiplier 不再默认为 1，而是从 spec 派生。
        # 视图永远是 dense 的（每页 2L 个行块），谁显式传 1 谁就把
        # 物理 id 写进 dense 行 —— 只有测试钉住 multiplier-1 时允许覆盖。
        self.kernel_page_multiplier = (
            spec.blocks_per_page()
            if kernel_page_multiplier is None
            else kernel_page_multiplier
        )
        # ...

    def translate_kv_loc_for_kernel(self, virt_tokens, *, out=None):
        """Virtual token ids -> kernel-facing ids:

            kernel_id(t) = (t // ps) * (ps * kernel_page_multiplier) + t % ps

        内部机制（compaction、inflight 写集合）必须继续使用
        translate_kv_loc：kernel-facing id 只供 kernel 使用。
        """
        ps = self.page_size
        stride = ps * self.kernel_page_multiplier # 页步长按行块数缩放
        with record_function("MultiEndedAlloc.translate_kv_loc_for_kernel"):
            pages = virt_tokens if ps == 1 else virt_tokens // ps
            offsets = None if ps == 1 else virt_tokens % ps
            if out is None:
                phys = self.virtual_to_physical[pages]
                ids = phys * stride if offsets is None else phys * stride + offsets
                return ids.clamp_(min=0) # 墓碑 -1 钳制到 0（page-0 水槽）
            # out= 路径：cuda-graph 捕获下缓冲区必须稳定；
            # pages 与 out 同 buffer 时经 torch.take 中转，避免自混叠。
            if pages.dtype != torch.int64:
                pages = pages.to(torch.int64)
            if pages is virt_tokens:
                out.copy_(torch.take(self.virtual_to_physical, pages))
            else:
                torch.take(self.virtual_to_physical, pages, out=out)
            out.mul_(stride)
            if offsets is not None:
                out.add_(offsets)
```

```
        return out.clamp_(min=0)
```

## python/sglang/srt/mem_cache/layout/page_major.py

布局构建器的核心改造：build_page_major_mha_views（4D strided）替换为 build_mha_views（3D dense 逐层视图），并推导 kernel_id 寻址公式；build_dense_mla_views 更名为 build_mla_views。

```
def build_mha_views(
    raw: torch.Tensor,
    *,
    layer_num: int,
    head_num: int,
    head_dim: int,
    v_head_dim: int,
    store_dtype: torch.dtype,
    page_size: int,
    num_pages: int,
    anchor_bytes: int = 0,
) -> Tuple[List[torch.Tensor], List[torch.Tensor]]:
    """Per-layer K/V views over raw for uniform-row MHA.

    信封 [L0_K*ps | L0_V*ps | L1_K*ps | ...] 在 K/V 行等宽时是
    2*layer_num 个行块的均匀数组，因此它本身就是一个有效的 paged pool，
    遵循 kernel_id(t) = (t // ps) * (ps * 2 * layer_num) + t % ps：
    层 l 的 K 在块 2l，V 在块 2l+1。每个视图是连续的
    (num_pages * 2 * layer_num * ps, head_num, head_dim)。
    """
    # 行块数组仅在 K/V 行等宽时存在；ServerArgs 启动时已筛掉
    # asymmetric-KV 模型，这里兜住直达 builder 的调用方。
    assert head_dim == v_head_dim, (
        f"build_mha_views requires uniform rows (head_dim == v_head_dim); "
        f"got head_dim={head_dim}, v_head_dim={v_head_dim}. Asymmetric-KV "
        "models cannot use the unified pool (screened out at startup)."
    )
    itemsize = store_dtype.itemsize
    row_elems = head_num * head_dim
    row_bytes = row_elems * itemsize
    blocks = 2 * layer_num # 每页 2L 个行块
    page_bytes = page_size * blocks * row_bytes
    n_rows = num_pages * blocks * page_size # kernel-facing id 空间行数
    assert anchor_bytes % itemsize == 0

    # 最后一个视图越过最后一页信封 (blocks-1)*ps 行，必须落在
    # UnifiedKVPool.view_tail_pad_bytes 提供的尾填充内，否则越界读。
    last_view_end = (
        anchor_bytes + (blocks - 1) * page_size * row_bytes + n_rows * row_bytes
    )
    assert last_view_end <= raw.numel() * raw.itemsize, (
        f"build_mha_views: block {blocks - 1}'s view ends at byte "
```

```
            f"{last_view_end} but the raw buffer holds only "
            f"{raw.numel() * raw.itemsize} bytes; allocate the tail pad "
            f"(one page envelope = {page_bytes} B) via view_tail_pad_bytes"
        )

        as_dtype_view = raw.view(store_dtype)
        k_buffer: List[torch.Tensor] = []
        v_buffer: List[torch.Tensor] = []
        for layer in range(layer_num):
            # 层 l 的 K 从块 2l 起点开始，V 紧跟偏移 ps 行；
            # storage_offset 折叠块起点，使所有视图共享同一 kernel-facing id。
            k_base_bytes = anchor_bytes + (2 * layer) * page_size * row_bytes
            v_base_bytes = k_base_bytes + page_size * row_bytes
            for base_bytes, out in ((k_base_bytes, k_buffer), (v_base_bytes, v_buffer)):
                assert base_bytes % itemsize == 0
                out.append(
                    torch.as_strided(
                        as_dtype_view,
                        size=(n_rows, head_num, head_dim),
                        stride=(row_elems, head_dim, 1),
                        storage_offset=base_bytes // itemsize,
                    )
                )
        return k_buffer, v_buffer
```

## 评论区精华

评审与讨论的核心交锋如下：

> kernel_page_multiplier 默认值 1 是最大残余风险（ch-wan，multi_ended_allocator.py）："Tail-pad 已正确地从 sub-pool specs 派生，但 id 空间缩放仍是默认为 1 的自由构造参数……一个走工厂外门的构造点会把物理 id 写入 dense 行，这正是 PR 声称必须不存在的中间态。" 作者最终采纳：kernel_page_multiplier 改为 None 时从 spec.blocks_per_page() 派生，仅测试保留显式覆盖。
>
> HIP 分支的 fused set_kv_buffer 无防护（ch-wan，models/utils.py）：CUDA 分支由 isinstance(pool, SWAKVPool) 排除 unified-SWA，但 HIP 分支没有同等 guard，create_fused_set_kv_buffer_arg 会把虚拟 id 分散写入 dense-paged 布局（gpt-oss、sdar + triton 在 HIP 上会踩中）。该问题先于本 PR 存在，但视图转 dense 后风险被放大；作者未在最终版处理 HIP 分支。
>
> triton_backend.py 分支过多与 MTP 复杂度（ch-wan）："I feel that this refactor makes the code logic more complicated. It adds many new if-else branches"，并建议 revert MTP 改动（"No one is going to use triton + mtp + unified memory"）。作者通过 PR 拆分解掉：本 PR 不再触碰 triton_backend.py，读路径翻译移入 #35247 的 prefix-only 页表构建，写路径在 #35245 的 ForwardBatch 构造时翻译一次，分支结构性消失。

> 测试同义反复（ch-wan，test_unified_mha_views.py）：test_dense_blocks_is_k_and_v_per_layer 断言的是自身一行实现，只有同时改测试与实现才会变红，建议删除；作者以 test_addressing_matches_strided_reference 等逐字节交叉验证作为真正锚点。
>
> assert 放置边界（ch-wan 追问，unified_memory_pool.py）：统一行断言从 MHASubPoolSpec.__post_init__ 移除，只保留 ServerArgs 用户报错与 build_mha_views 构造期 assert——前者给出含四个 head dims 的可读信息，后者兜住直达 builder 的调用方（" 其寻址算术真正会坏的地方 "）。

- kernel_page_multiplier 默认 1 会把物理 id 写进 dense 行 (design): 采纳。MultiEndedAllocator 的 kernel_page_multiplier 改为 None 时从 spec.blocks_per_page() 派生，显式覆盖仅保留给测试钉住 multiplier-1 折叠。
- HIP 分支 fused set_kv_buffer 无 SWAKVPool guard (correctness): 未解决。作者未在最终版处理 HIP 分支；ch-wan 建议要么补 HIP guard，要么删除具有误导性的解说注释。
- triton_backend.py 分支复杂度与 MTP 支持范围 (design): 通过重组解决。本 PR 不再触碰 triton_backend.py，系列携带零 speculative 改动。
- test_dense_blocks_is_k_and_v_per_layer 是同义反复 (testing): 采纳。最终版以 test_spec_offsets_equal_block_origins、test_addressing_matches_strided_reference 等逐字节验证替代。
- two_batch_overlap.py 的 getattr 与 swa_out_cache_loc 轨道 (style): 通过拆分解决，本 PR 内不再存在该问题。
- 统一行断言应放在哪一层 (question): 采纳。MHASubPoolSpec.__post_init__不再断言统一行，保留 ServerArgs 与 builder 两个边界。
- _assert_kernel_id_bound 为何与 layer_num 相关 (question): 已解释清楚，检查保留。

## 风险与影响

- 风险：识别到以下风险，均具体到文件与逻辑：

1. multiplier 覆盖参数仍可制造病态组合（multi_ended_allocator.py）：虽然生产路径已从 spec.blocks_per_page() 派生，但构造函数仍保留 kernel_page_multiplier= 显式覆盖（供测试钉住 multiplier-1 collapse）；未来任何新构造点若显式传入 1，会把物理 id 写进 dense 行，属于静默 KV 损坏而非报错，依赖评审把关。
2. HIP unified-SWA 写入错误 id 空间（models/utils.py）：enable_fused_set_kv_buffer 的 HIP 分支没有 SWAKVPool isinstance guard，视图变 dense 后，HIP + unified-SWA + triton 组合会把虚拟 id 直接分散写入 dense-paged 布局，产生不可预测的 KV 错位；这是 pre-existing 漏洞，但被本 PR 放大。
3. 静态 page-major 布局暂时下线（memory_pool.py、arg_groups/kv_cache_hook.py）：--enable-page-major-kv-layout（不带 unified memory）启动即失败，依赖该 flag 的存量部署必须先切换到 --enable-unified-memory 或回退布局。
4. asymmetric-KV 模型被拒（unified_memory_pool.py 的 unified_memory_supported_for_model）：MiMoV2 这类 head_dim != v_head_dim 的模型无法再使用 unified memory，启动报错信息提供了四个 head dims 与相关 flag，属预期

行为收紧但需要用户感知。

5. int32 行数边界（_assert_kernel_id_bound）：kernel-facing id 行数 = num_pages * blocks_per_page * page_size，MHA 每页 2L 块；超大层数 + 大 max_total_num_tokens 会触发启动断言，需在容量规划时留意。
6. 删除符号的兼容影响：store_cache_4d / store_cache_4d_kernel / build_page_major_mha_views 被整体移除，任何未同步迁移的内部或第三方调用方将直接 import 失败（属内部 API 破坏，影响可控）。
7. 性能小幅波动：作者 benchmark 显示 gpt-oss-20b ITL 中位数 -1.01%（其余模型为正或持平），负向幅度在噪声边缘，建议在真实生产配置下观察。 - 影响：影响评估：

- 用户侧：启用 --enable-unified-memory 的用户获得布局统一与后续后端接入的可能；asymmetric-KV 模型被启动门禁拒绝，需改用非 unified 布局；静态 page-major 布局（非 unified）使用者需要迁移。
- 系统侧：统一池 MHA/SWA 视图从 strided 4D 变 dense 3D 后，Triton 读写字节布局不变（benchmark 6 模型族准确率 68/68 cells 干净），但 KV transfer、CPU offload、PD 传输在 unified MHA 池上改为显式 NotImplementedError；int32 read-index 的边界现在有启动期断言保护。
- 团队 / 架构侧：净删除约 509 行代码，移除一个 Triton kernel 与两条 4D 特殊分支，长期维护面缩小；同时建立了一个清晰的依赖栈（本 PR → #35247 → #35245 → #34613），使评审可逐层把关，削减了此前布局 + 重构混杂带来的认知负担。
- 性能：按作者提供的产品配置 serving benchmark，ITL 变化在 -1.01% ~ +1.22% 之间，无显著可感知影响。
- 风险标记：核心路径布局切换，原子切换（视图 /ID/ 尺寸联动），asymmetric-KV 模型启动被拒，静态 page-major arm 暂时下线，HIP 分支 fused set_kv_buffer 无防护，multiplier 覆盖参数残留风险

## 关联脉络

- PR #35245 refactor(unified-memory): translate the KV write location once, at ForwardBatch construction: PR body 声明的依赖栈成员：本 PR 的 write-location 重构拆分后落到 #35245，翻译一次发生在 ForwardBatch 构造期。
- PR #35247：PR body 声明的依赖栈中间层（Stack order: this PR → #35247 → #35245 → #34613），承接读路径的 kernel-facing 页表构建（prefix-only 翻译）。
- PR #34613：PR body 声明的依赖栈末端，系列最终形态所在。
- PR #35773：PR body 说明 SWA tombstone 修复已被上游 #35773（clear_full_to_swa_mapping 修复同一 bug）取代，本系列不再携带该修复。