PR #36798 完整报告

sgl-project/sglang

[HiCache] Align chunked CUDA host registrations

合并时间：2026-08-29 20:36

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/36798

执行摘要

- 一句话：HiCache 大池分块注册 cudaHostRegister 并支持回滚
- 推荐动作：值得精读 common.py 的 _cuda_host_register 分块与回滚实现，以及各 pool 如何推导 registration_granularity_bytes；这是与底层 CUDA API 语义打交道的良好范例，对理解 HiCache 大规模部署的初始化路径有参考价值。建议后续在真实 CUDA 环境跑一次大池初始化 / 吞吐回归，并关注 layer-first 大池的后续处理。

功能与动机

PR body 明确指出：大型 HiCache host pool 可能超出单次 cudaHostRegister 调用的实用大小；在任意字节偏移处切分注册对 page-first 拷贝不安全，因为一个被复制的页面可能跨越两个注册区间，导致cudaMemcpyBatchAsync返回invalid argument。因此需要一种既能分块、又保证注册边界与逻辑拷贝页对齐的注册方案。

实现拆解

1. 配置入口：在 python/sglang/srt/environ.py 新增 SGLANG_HICACHE_HOST_REGISTER_CHUNK_GB 环境变量（默认 256 GiB），作为单次 cudaHostRegister 的字节上限，供 pool_host/common.py 读取。
2. 注册核心重构：python/sglang/srt/mem_cache/pool_host/common.py 中 _cuda_host_register 新增 registration_granularity_bytes 参数；chunk_bytes 默认等于整个 buffer 大小以保持旧行为，传入拷贝页粒度时向下对齐为该粒度的整数倍；循环调用 cudaHostRegister，将每个成功的 (ptr, size) 追加到 _sglang_cuda_host_registered_ranges 属性；任一块失败时通过 _cuda_host_unregister_ranges 逆序回滚并抛 RuntimeError。配套新增 _cuda_host_unregister_ranges 供 teardown 与回滚共用。
3. 各 host pool 接入：mha.py、mla.py 对 page_first / page_first_direct 布局传 page_size * layout_dim；mamba.py 传单页字节 int(np.prod(dims[1:])) * dtype.itemsize；memory_pool_host.py 中 DeepSeek V4 的 paged/state pool 传 layer_num * item_bytes 与 layer_num * state_page_bytes；dsa.py 跟进。layer-first 与未知布局不传参，维持单次注册兼容行为。
4. 测试与校验：新增 test/registered/unit/mem_cache/test_hicache_host_register.py（412 行），用 _FakeBuffer / _FakeCudart 在 CPU 上模拟 CUDA runtime，覆盖 chunk 边界、生命周期清理、注册失败回滚及全部受影响布局的 granularity 透传；同时跑通相关

host-pool 环境测试与 UMBP fallback 测试。

关键文件：

- python/sglang/srt/mem_cache/pool_host/common.py（模块 注册逻辑；类别 source；类型 core-logic；符号 _cuda_host_register, _cuda_host_unregister_ranges, _CUDA_HOST_REGISTERED_RANGES_ATTR）：分块注册核心逻辑所在，新增 granularity 对齐、范围记录、回滚与注销辅助函数。
- test/registered/unit/mem_cache/test_hicache_host_register.py（模块 测试覆盖；类别 test；类型 test-coverage；符号 _FakeBuffer, _FakeCudart, cudaHostRegister, cudaHostUnregister）：412 行新增 CPU 单测，用 fake cudart 覆盖分块边界、清理、回滚和各布局 granularity 透传，是本次变更正确性的主要保障。
- python/sglang/srt/mem_cache/pool_host/mha.py（模块 host 池；类别 source；类型 core-logic；符号 MHATokenToKVPoolHost.init_kv_buffer, MHATokenToKOnlyPoolHost.init_kv_buffer, AsymmetricMHATokenToKVPoolHost.init_kv_buffer）：MHA、K-only MHA、Asymmetric K/V 三类 page-first 布局在此注入 granularity 参数。
- python/sglang/srt/mem_cache/pool_host/mla.py（模块 host 池；类别 source；类型 core-logic；符号 MLATokenToKVPoolHost.init_kv_buffer）：MLA host pool 的 page_first/page_first_direct 布局需要同样的粒度对齐。
- python/sglang/srt/mem_cache/memory_pool_host.py（模块 内存池；类别 source；类型 core-logic；符号 DeepSeekV4PagedHostPool, DeepSeekV4StateHostPool）：DeepSeek V4 的 paged 与 state host pool 需要按 layer 页粒度对齐。
- python/sglang/srt/mem_cache/pool_host/mamba.py（模块 host 池；类别 source；类型 core-logic；符号 MambaPoolHost.init_kv_buffer）：Mamba 状态池的页粒度注册需要统一的表达式。
- python/sglang/srt/environ.py（模块 环境配置；类别 source；类型 configuration；符号 SGLANG_HICACHE_HOST_REGISTER_CHUNK_GB）：引入新的配置键控制分块大小，是行为开关入口。
- python/sglang/srt/mem_cache/pool_host/dsa.py（模块 host 池；类别 source；类型 core-logic；符号 DSAIndexerPoolHost）：DSA indexer 池也需与上述布局一致地传递粒度参数。

关键符号：_cuda_host_register, _cuda_host_unregister_ranges, MHATokenToKVPoolHost.init_kv_buffer, MHATokenToKOnlyPoolHost.init_kv_buffer, AsymmetricMHATokenToKVPoolHost.init_kv_buffer, MLATokenToKVPoolHost.init_kv_buffer, MambaPoolHost.init_kv_buffer, DeepSeekV4PagedHostPool.init, DeepSeekV4StateHostPool.init, DSAIndexerPoolHost.init_kv_buffer

关键源码片段

python/sglang/srt/mem_cache/pool_host/common.py

分块注册核心逻辑所在，新增 granularity 对齐、范围记录、回滚与注销辅助函数。

```
def _cuda_host_register(
    buffer: torch.Tensor, registration_granularity_bytes: int | None = None
) -> None:
    """分块注册 host 内存，避免单次 cudaHostRegister 调用过大。

    传入 registration_granularity_bytes（逻辑拷贝页的字节数）时，块大小会
    向下对齐到该粒度的整数倍，保证一个拷贝页不会跨两个注册区间。
    """
    cudart = torch.cuda.cudart()
    base = buffer.data_ptr()
    total = buffer.numel() * buffer.element_size()
    chunk_limit_bytes = (
        max(envs.SGLANG_HICACHE_HOST_REGISTER_CHUNK_GB.get(), 1) * 1024**3
    )

    # 保留旧行为：不传粒度时一次注册整个 buffer，兼容 layer-first 等未知布局。
    chunk_bytes = total
    if registration_granularity_bytes is not None:
        if registration_granularity_bytes <= 0:
            raise ValueError(
                "registration_granularity_bytes must be positive, got "
                f"{registration_granularity_bytes}"
            )
        if registration_granularity_bytes > chunk_limit_bytes:
            raise ValueError(
                "Host registration granularity exceeds the configured chunk limit: "
                f"granularity={registration_granularity_bytes}, "
                f"chunk_limit={chunk_limit_bytes}"
            )
        # 向下取整到 page 粒度的整数倍，避免一页跨两个块。
        chunk_bytes = (
            chunk_limit_bytes // registration_granularity_bytes
        ) * registration_granularity_bytes

    registered_ranges: list[tuple[int, int]] = []
    try:
        offset = 0
        while offset < total:
            size = min(chunk_bytes, total - offset)
            ptr = base + offset
            rc = int(cudart.cudaHostRegister(ptr, size, 0))
            if rc != 0:
                raise RuntimeError(
                    f"cudaHostRegister failed (rc={rc}, "
                    f"{cudart.cudaGetErrorString(rc)}) at offset={offset} "
                    f"size={size} (total={total}, chunk_limit={chunk_bytes}); "
                    f"host buffer is not pinned and device transfers may "
                    f"silently return stale data."
                )
```

```
            registered_ranges.append((ptr, size))
            offset += size

        # 记录每个注册基址；CUDA 要求 cudaHostUnregister 必须收到各块基址。
        setattr(buffer, _CUDA_HOST_REGISTERED_RANGES_ATTR, registered_ranges)
    except Exception:
        # 逆序回滚已成功注册的块，避免留下部分 pinned 的 host 内存。
        remaining_ranges = _cuda_host_unregister_ranges(
            cudart, registered_ranges, operation="registration rollback"
        )
        if remaining_ranges:
            setattr(buffer, _CUDA_HOST_REGISTERED_RANGES_ATTR, remaining_ranges)
        raise
```

test/registered/unit/mem_cache/test_hicache_host_register.py

412 行新增 CPU 单测，用 fake cudart 覆盖分块边界、清理、回滚和各布局 granularity 透传，是本次变更正确性的主要保障。

```
class _FakeCudart:
    """模拟 CUDA runtime，记录注册/注销调用并支持注入失败。"""

    def __init__(self, fail_on_registration: int | None = None):
        self.registrations = []
        self.unregistrations = []
        self.fail_on_registration = fail_on_registration

    def cudaHostRegister(self, ptr: int, size: int, flags: int) -> int:
        self.registrations.append((ptr, size, flags))
        # 第 N 次调用返回错误码 1，用于验证回滚路径。
        if len(self.registrations) == self.fail_on_registration:
            return 1
        return 0

    def cudaHostUnregister(self, ptr: int) -> int:
        self.unregistrations.append(ptr)
        return 0

    def cudaGetErrorString(self, rc: int) -> str:
        return "injected error"


class TestHiCacheHostRegister(unittest.TestCase):
    def test_page_first_direct_mla_uses_page_registration_granularity(self):
        # 通过 mock 注入 ALLOC_MEMORY_FUNCS，捕获 init_kv_buffer 传给
        # alloc_func 的 registration_granularity_bytes 参数是否正确。
        pool = MLATokenToKVPoolHost.__new__(MLATokenToKVPoolHost)
        pool.layout = "page_first_direct"
        pool.page_num = 4
        pool.layer_num = 3
```

```
        pool.page_size = 2
        pool.kv_cache_dim = 5
        pool.dtype = torch.float16
        pool.device_pool = SimpleNamespace(device="cuda")
        pool.device = "cpu"
        pool.pin_memory = True
        pool.allocator = object()
        alloc = mock.Mock(return_value=object())

        with mock.patch.dict(mla_pool_host.ALLOC_MEMORY_FUNCS, {"cuda": alloc}):
            pool.init_kv_buffer()

        # 期望粒度 = page_size * layer_num * kv_cache_dim * dtype.itemsize。
        self.assertEqual(
            alloc.call_args.kwargs["registration_granularity_bytes"],
            pool.page_size * pool.layer_num * pool.kv_cache_dim * pool.dtype.itemsize,
        )
```

评论区精华

本 PR 没有公开 review 评论（review_comments_count=0），审核者 hzh0425 直接 APPROVED。值得注意的演进来自 commit 历史：[HiCache] Fix chunked host registration cleanup 与 [HiCache] Use conservative 256 GiB register chunks 两次修正表明作者在合入前收敛了回滚清理逻辑，并把默认块大小定在保守的 256 GiB。由于没有评论线程，未发现遗留争议点。

- 合并审查：无公开讨论 (other): 无未解决问题，已批准合并。

风险与影响

- 风险：
  - 核心路径变更：pool_host/common.py 的注册逻辑是所有 host pool 初始化的必经之路，分块循环、属性记录和回滚一旦有误会影响整个 HiCache 启动。
  - 真实 CUDA 行为未直接验证：新测试基于 fake cudart，无法捕获真实 cudaHostRegister 在分块下的对齐 / 页大小限制；建议在真实 CUDA 环境补充一次大池初始化验证。
  - 配置冲突风险：当 registration_granularity_bytes 大于 chunk 上限（例如超大 page 尺寸叠加小 chunk 配置）时会直接抛 ValueError，用户需显式调大 SGLANG_HICACHE_HOST_REGISTER_CHUNK_GB。
  - layer-first 兼容性：layer-first 布局保持单次注册，超大 layer-first pool 仍可能触发原始的单次调用过大问题。
  - 性能影响：分块后注册调用次数从 1 次变为 N 次，超大 pool 初始化时间可能增加；但相比失败重试是更可控的权衡。
- 影响：

- 用户：HiCache 大池部署用户不再因单次注册过大而启动失败，page-first 拷贝不再因跨注册区间报 invalid argument。
- 系统：host pool 初始化从单次注册变为可配置分块注册，teardown 需逐块注销，生命周期管理更复杂但更健壮。
- 团队：registration_granularity_bytes 作为新约定需要各 pool 维护者遵循；测试提供了可复用的 fake cudart 模式。
- 影响范围：mem_cache/pool_host 下 MHA、MLA、DSA、Mamba、DeepSeek V4 系列及 environ.py 配置。
- 风险标记：核心路径变更，真实 CUDA 行为未直接验证，新增环境变量配置，layer-first 保持旧行为

关联脉络

- PR #36834 [HiCache] buffer mode: decide staged-fetch fate against the live tree: 同属 HiCache 内存池与调度改造，改动 mem_cache 与 scheduler 核心路径，与本 PR 的 host pool 注册生命周期同处一条演进线。
- PR #36958 [misc] Keep req.kv non-optional and key KV ownership on req_pool_idx: 梳理 KV 缓存所有权与生命周期，与本 PR 的注册 / 注销生命周期管理同属 KV 缓存可靠性主题。