PR #25377 完整报告

sgl-project/sglang

[HiCache][AMD] Add UMBP tiered DRAM + SSD L3 storage backend with hugepage host allocator

合并时间：2026-07-01 22:21

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/25377

执行摘要

- 一句话：新增基于 mori/UMBP 的 HiCache L3 存储后端
- 推荐动作：本 PR 设计清晰，零拷贝后端接口和严格配置解析模式值得借鉴。建议关注点：1）UMBPStore 的配置解析（严格布尔、环境变量覆盖、多 rank 适配）可作为后续后端的参考；2）mock 测试通过 fake 模块模拟外部依赖，避免 GPU 要求，提高 CI 可靠性。若正在开发 HiCache 或类似存储后端，建议精读 umbp_store.py 和 umbp_host_allocator.py。

功能与动机

在 MI355X 上，HBM 容量是长上下文多 DP 服务的瓶颈。HiCache 现有 HBM 层（L2）在 agentic 和多轮工作负载中无法保持足够的重复前缀 KV，导致大量可缓存 prefill 被重计算。本 PR 通过 mori 库的 UMBP 子系统添加宿主端 L3 层（DRAM + 可选 SSD），扩展 HiCache 层级，减少重计算，降低 TTFT。

实现拆解

1. 实现 UMBPStore 类：在 python/sglang/srt/mem_cache/storage/umbp/umbp_store.py 中新增 UMBPStore，遵循 HiCache 零拷贝 v1 接口（batch_get_v1 / batch_set_v1），类似 MooncakeStore 模式，支持 MHA / MLA / split-heads。
2. 实现 UMBPHostTensorAllocator：在 umbp_host_allocator.py 中包装 mori 的 UMBPHostMemAllocator，支持 hugepage（2MB 默认）、NUMA 绑定、预故障，并通过 pool_host/common.py 的 get_allocator_from_storage 注册。
3. 注册后端：在 backend_factory.py 中调用 StorageBackendFactory.register_backend("mori", ...)，并在 _create_builtin_backend 中添加 mori 分支。
4. 参数支持：在 server_args.py 的 --hicache-storage-backend choices 中添加 mori。
5. 配套修复：在 cache_controller.py 中添加 import os 以支持 DP rank 环境变量读取。
6. 单元测试：新增 test_umbp_store.py 和 test_umbp_host_allocator.py，通过 mock mori 库实现无 GPU 测试，注册到 CPU CI 套件（base-a-test-cpu）。

关键文件：

- python/sglang/srt/mem_cache/storage/umbp/umbp_store.py（模块 存储后端；类别 source；类型 dependency-wiring；符号 _import_umbp_client, _optional_env_int, _optional_env_str, _strict_bool）：核心新增文件：实现 UMBPStore 类，提供零拷贝 v1

接口和丰富的配置解析逻辑，是整个后端的核心。

- python/sglang/srt/mem_cache/storage/umbp/umbp_host_allocator.py（模块 宿主分配器；类别 source；类型 dependency-wiring；符号 _bool_env, _int_env, UMBPHostTensorAllocator, init）：核心新增文件：实现 UMBPHostTensorAllocator，使用 mori 的 UMBPHostMemAllocator 分配宿主内存，支持 hugepage/NUMA。
- test/registered/unit/mem_cache/test_umbp_store.py（模块 单元测试；类别 test；类型 test-coverage；符号 MockStorageConfig, MockHostKVCache, get_page_buffer_meta, fill_page）：新增单元测试：通过 MockHostKVCache 模拟宿主 KV 缓存，验证 UMBPStore 的 set/get 行为。
- test/registered/unit/mem_cache/test_umbp_host_allocator.py（模块 单元测试；类别 test；类型 test-coverage；符号 FakeBacking, FakeHandle, init, bool）：新增单元测试：通过 FakeHostMemAllocator 模拟 mori 分配器，验证 UMBPHostTensorAllocator 的分配和释放。
- python/sglang/srt/mem_cache/pool_host/common.py（模块 池基础；类别 source；类型 dependency-wiring；符号 get_allocator_from_storage）：修改文件：在 get_allocator_from_storage 中增加 "mori" 分支，支持动态切换到 UMBP 分配器。
- python/sglang/srt/mem_cache/storage/backend_factory.py（模块 后端工厂；类别 source；类型 core-logic；符号 StorageBackendFactory.register_backend, _create_builtin_backend）：修改文件：注册 "mori" 后端到 StorageBackendFactory，并在 _create_builtin_backend 中添加分支。
- python/sglang/srt/managers/cache_controller.py（模块 缓存管理器；类别 source；类型 entrypoint；符号 import os）：修改文件：添加 import os 以支持读取环境变量 SGLANG_DP_RANK。

关键符号：_strict_bool, _select_rank_config_value, _cast_like, UMBPHostTensorAllocator.init, UMBPHostTensorAllocator.allocate, get_allocator_from_storage, StorageBackendFactory.register_backend

关键源码片段

python/sglang/srt/mem_cache/storage/umbp/umbp_store.py

核心新增文件：实现 UMBPStore 类，提供零拷贝 v1 接口和丰富的配置解析逻辑，是整个后端的核心。

```
def _strict_bool(value, key):
    # 严格布尔解析：避免 bool("false") 意外为 True
    if isinstance(value, bool):
        return value
    if isinstance(value, int) and value in (0, 1):
        return bool(value)
    if isinstance(value, str):
        norm = value.strip().lower()
        if norm in ("1", "true", "yes", "on"):
            return True
        if norm in ("0", "false", "no", "off"):
```

```
            return False
    raise ValueError(f"extra_config[{key!r}] must be a boolean-like value, got {value!r}")


def _select_rank_config_value(value, rank_index, field_name, cast_type,
                              auto_increment_scalar=False):
    # 支持逗号分隔列表的多 rank 分配
    if value is None:
        raise ValueError(f"{field_name} must not be None")
    candidates = value
    if isinstance(value, str) and "," in value:
        candidates = [item.strip() for item in value.split(",")]
    if isinstance(candidates, (list, tuple)):
        if not candidates:
            raise ValueError(f"{field_name} must not be empty")
        if rank_index >= len(candidates):
            raise ValueError(f"{field_name} has {len(candidates)} entries, rank_index={rank_index}")
        return cast_type(candidates[rank_index])
    selected = cast_type(candidates)
    if auto_increment_scalar:
        selected = cast_type(selected + rank_index)
    return selected
```

python/sglang/srt/mem_cache/storage/umbp/umbp_host_allocator.py

核心新增文件：实现 UMBPHostTensorAllocator，使用 mori 的 UMBPHostMemAllocator 分配宿主内存，支持 hugepage/NUMA。

```
class UMBPHostTensorAllocator(HostTensorAllocator):
    # 使用 mori 的 UMBPHostMemAllocator 分配宿主内存
    def allocate(self, dims, dtype, device="cpu"):
        if device != "cpu":
            raise ValueError(f"UMBP allocator only supports CPU, got {device}")
        nbytes = math.prod(int(d) for d in dims) * torch.empty((), dtype=dtype).element_size()
        # 根据环境变量决定 hugepage 或匿名映射
        backing = (self._mod.UMBPHostBufferBacking.AnonymousHugetlb
                   if self._use_hugepage else self._mod.UMBPHostBufferBacking.Anonymous)
        handle = self._allocator.alloc(nbytes, backing, self._hugepage_size,
                                       self._numa_node, self._prefault)
        if not handle:
            raise RuntimeError(f"UMBP alloc failed for {nbytes} bytes")
        self._handles[int(handle.ptr)] = handle
        c_array = (ctypes.c_byte * nbytes).from_address(handle.ptr)
        tensor = torch.frombuffer(c_array, dtype=torch.uint8, count=nbytes).view(dtype)
        logger.info("Allocated %.2f GB at 0x%x ...", nbytes/1e9, handle.ptr)
        if self._use_hugepage and handle.actual_backing != backing:
            logger.warning("Hugepage request demoted to 4KiB pages")
        return tensor.view((dims))
```

评论区精华

CI 覆盖争议

- amd-bot指出 PR CI 没有执行任何新代码，绿色结果无意义。作者随后将 mock 单元测试注册到 test/registered/unit/mem_cache/ 并加入 CPU CI 套件，解决了该问题。

回归验证

- 作者在 NVIDIA B200 上完整运行现有 HiCache CUDA 测试套件（6 个用例），全部通过，确认共享代码修改无回归。

代码审查建议

- gemini-code-assist[bot]提出三点建议：① 在 allocate 中让 tensor 持有 allocator 引用防止 GC；② 在 KVEventsSubscriber 中添加空 hash 检查减少网络开销；③ 使用具体异常类捕获后台线程异常。这些建议未在合并前明确采纳，但整体设计已合并。
- CI 覆盖争议与 mock 测试注册 (testing): 已解决：mock 测试已注册并在 PR CI 中执行。
- 张量应持有分配器引用以防止过早 GC (correctness): 未明确采纳，可能因为项目当前所有权模型已确保 allocator 生命周期。
- 空哈希检查避免不必要远程调用 (performance): 未明确采纳，但属防御性编程建议。
- 后台线程异常应具体化 (correctness): 未明确采纳，但日志记录已提供基本诊断能力。
- NVIDIA B200 回归确认 (other): 已验证通过。

风险与影响

- 风险：
  1. 平台依赖：mori 库是 AMD 专用，非 AMD 平台需通过 mock 测试跳过构建。
  2. 布局强制：--hicache-mem-layout page_first 是硬性要求，layer_first 布局会断言失败。
  3. 写策略限制：write_through 策略在部分配置下导致 detokenizer 死锁，推荐使用 write_back。
  4. SSD 路径未充分验证：SSD 配置代码已包含，但仅做过烟雾测试（capacity=0），实际 SSD 卸载功能缺乏端到端验证。
  5. 性能风险：初始实现中 batch_set_v1 / batch_get_v1 的 INFO 日志在高吞吐下造成开销，已在提交中降级为 DEBUG；pin_memory 设置已还原避免其他后端回归。
- 影响：
  - 用户：AMD MI3xx 用户可通过 --enable-hierarchical-cache --hicache-storage-backend mori 启用 L3 缓存，显著改善长上下文场景 TTFT 和命中率。非 AMD 用户无影响。
  - 系统：新增环境变量 SGLANG_HICACHE_HOST_HUGEPAGE（默认 True）、SGLANG_HICACHE_HOST_HUGEPAGE_SIZE（2MB）、SGLANG_HICACHE_HOST_NUMA_NODE（-1）、SGLANG_HICACHE_HOST_PREFAULT（True）用于微调宿主分配器。
  - 团队：需为 mori 库提供安装文档和版本依赖管理；后续维护需跟踪 mori API 变更。
  - 风险标记：依赖 AMD 专用库 mori，write_back 策略必需，page_first 布局强制，SSD 路径未充分验证

关联脉络

- PR #25556 [AMD] Fix correctness for AITER MLA backend with --page-size > 1: 本 PR 包含该 fix 的 cherry-pick 以支持 MLA 正确性，PR body 中也明确提到。
- PR #27898 RFC: HiCache storage backend architecture: 讨论中提到的 RFC issue，涉及 HiCache 存储后端架构设计。
- PR #28287 [HiCache] Optimize HiCache hash generation with bulk token byte conversion: 同为 HiCache 性能优化，共享 MemCache 模块。