PR #28287 完整报告

sgl-project/sglang

[HiCache] Optimize HiCache hash generation with bulk token byte conversion

合并时间：2026-07-01 15:44

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/28287

执行摘要

- 一句话：将 HiCache L3 哈希生成从 Python per-token 循环改为 C++ native 扩展，实现约 70× 加速
- 推荐动作：值得精读。该 PR 展示了一种典型的热点路径优化模式：识别 Python 循环瓶颈，通过 C++ 扩展和 AVX2 指令集大幅加速，同时保持 Python 调用的便利性。设计决策中关于 per-page scratch buffer 的使用、bigram 重叠对的生成、以及优先利用已有 array 缓冲区避免额外拷贝的思想，值得在其他性能敏感模块中借鉴。

功能与动机

原有的哈希生成中，每个 page 的哈希通过 Python 循环 `for t in token_ids: hasher.update(t.to_bytes(4, byteorder="little", signed=False))` 计算，当 HiCache L3 需要为大量 page 计算哈希时开销很大。该 PR 旨在通过 C++ 原生扩展来消除 per-token Python 调用，大幅提升哈希生成性能。

实现拆解

1. 新增 C++ 扩展（hash_binding.cpp）：提供核心哈希例程，支持 regular 和 bigram 模式，使用 AVX2 指令加速 uint64 到 uint32 的转换，并通过 per-page scratch buffer 避免额外内存分配。
2. 新增 Python 包装（native_hash.py）：检测平台兼容性（仅支持 little-endian Linux），通过 torch.utils.cpp_extension.load 在运行时编译加载扩展，提供 _native_hash_input 函数预处理输入数据（支持 array、列表、tuple bigram、EAGLE bigram 等格式），最终调用 C++ 的 get_hash 函数。
3. 重构调用方：修改 utils.py 中的 get_hash_str 函数，新增 page_size 参数并委托给 get_native_hash；修改 radix_cache.py 中的 TreeNode.hash_page 方法，改为调用 get_hash_str；修改 cache_controller.py 和 hybrid_cache_controller.py 中的 _storage_hit_query 方法，利用批量哈希批处理减少 Python 循环。
4. 配套测试：在 test_mem_cache_utils.py 中添加了 _legacy_get_hash_str 和 _legacy_page_hashes 函数作为参考实现，并通过 _HashKey 辅助类构造了多种输入（plain list、array('I')、array('q')、bigram 等），遍历单哈希和页面哈希的兼容性用例，确保新实现与旧实现输出完全一致。

关键文件：

- python/sglang/srt/mem_cache/cpp_utils/native_hash.py（模块 哈希引擎；类别 source；类型 dependency-wiring；符号 _cpu_supports_avx2, _load_native_hash_module, _native_hash_input, get_native_hash）：新增 Python 包装文件，负责检测平台兼容性、加载 C++ 扩展并准备输入数据，是连接 Python 调用方与 C++ 核心的桥梁。
- python/sglang/srt/mem_cache/cpp_utils/hash_binding.cpp（模块 哈希引擎；类别 source；类型 dependency-wiring；符号 hash_page, fill_regular_page, fill_bigram_page, checked_u32）：新增 C++ 核心实现，包含 SHA256 上下文的复用、per-page scratch buffer 的填充逻辑，以及 AVX2 优化的 uint64 到 uint32 压缩转换。
- python/sglang/srt/mem_cache/utils.py（模块 缓存工具；类别 source；类型 core-logic；符号 get_hash_str）：核心修改文件，将原有的 get_hash_str 函数替换为委托给 get_native_hash，并新增 page_size 参数以支持批量页面哈希计算。
- python/sglang/srt/mem_cache/radix_cache.py（模块 Radix 缓存；类别 source；类型 dependency-wiring；符号 TreeNode.hash_page）：修改了 TreeNode.hash_page 方法，由原来的内联 SHA256 改为调用 get_hash_str，并移除了直接的 hashlib 依赖。
- python/sglang/srt/managers/cache_controller.py（模块 缓存控制器；类别 source；类型 entrypoint；符号 CacheController._storage_hit_query）：修改了 _storage_hit_query 方法，将原先的内部循环批量查询改为一次性调用 get_hash_str(page_size=...) 获取所有页面哈希后再切片处理，减少了 Python 循环。
- python/sglang/srt/mem_cache/hybrid_cache/hybrid_cache_controller.py（模块 混合缓存；类别 source；类型 entrypoint）：类似 cache_controller.py，更新了混合缓存的哈希查询路径，使用新的批量哈希接口。
- test/registered/unit/mem_cache/test_mem_cache_utils.py（模块 测试；类别 test；类型 test-coverage；符号 TestGetEvictionStrategy, _legacy_get_hash_str, _legacy_page_hashes, _HashKey）：大幅扩展的测试文件，新增了 _legacy_get_hash_str、_legacy_page_hashes 参考实现和 _HashKey 辅助类，覆盖了多种输入格式的兼容性验证。

关键符号：get_hash_str, get_native_hash, _native_hash_input, TreeNode.hash_page, CacheController._storage_hit_query, compute_node_hash_values

评论区精华

review 主要由 gemini-code-assist[bot] 提出两条中等优先级建议：

1) 使用 Union[str, List[str]] 代替 str | List[str] 以兼容 Python <3.10； 2) 使用 itertools.chain.from_iterable 优化嵌套生成器。 但这两个建议未被作者采纳，最终 PR 仍保留了 str | List[str] 和嵌套生成器写法，由 xiezhq-hermann 批准合并。

- 关于 Python 版本兼容性使用 Union 的建议 (style): 作者未采纳建议，PR 仍保留 str | List[str] 和嵌套生成器写法，由 xiezhq-hermann 批准合并。

风险与影响

- 风险： 1) C++ 扩展依赖 OpenSSL 的 -lcrypto 及 pybind11，在国内环境或某些 Docker 镜像下可能导致编译失败。 2) 仅支持 little-endian Linux 系统，其他平台（如 macOS、Windows）会抛出 RuntimeError。 3) 哈希算法虽然逻辑一致，但新实现必须保证与旧实现

的精确输出一致，以免导致缓存 miss 或数据不一致。测试中已通过_legacy_get_hash_str 进行了兼容性验证，但生产环境下仍可能存在边缘 case。 4) 新扩展通过 torch.utils.cpp_extension.load 在运行时编译，可能引入首次调用时的延迟。

- 影响：直接提升 HiCache L3 哈希生成性能约 70x，降低缓存查询开销。对系统其余部分无影响。所有调用方均在内部委托给新实现，对外 API 保持向后兼容（get_hash_str 新增了可选参数 page_size）。测试覆盖了单页面哈希和批量页面哈希的兼容性。
- 风险标记：编译依赖 OpenSSL/pybind11，仅限 little-endian Linux，首次编译延迟，哈希一致性需验证

关联脉络

- 暂无明显关联 PR