PR #33085 完整报告

sgl-project/sglang

perf(hisparse): 128-bit non-temporal swap-in copy on ROCm

合并时间：2026-08-11 04:27

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/33085

执行摘要

- 一句话：HiSparse ROCm miss 拷贝改 128 位非临时加载，decode 提速
- 推荐动作：值得精读，重点看三处：(1) 门控条件的推导——为什么 128 位拷贝在 512B item 下反而更慢，item_size_bytes >= WARP_SIZE * 16 背后的 lane 利用率模型；(2) __builtin_nontemporal_load 与缓存 store 的取舍——源只读一次用 non-temporal，目标被后续 attention 内核读所以保留缓存；(3) 接缝测试的参数设计——覆盖宽路径与 64 位 / 字节余量循环的每个衔接形状，并用 skipif(not is_hip()) 明确 CUDA 分支的契约边界。建议把 PR body 的 benchmark 表格与 commit 2 的隔离实验一起看，这是一个完整的"优化→发现回归→定位根因→门控→补测试"闭环。

功能与动机

PR body 指出 transfer_item_warp 在仓库中存在两份实现：HiCache 的 L2→L1 gather 版本（kernels/aot/csrc/kvcacheio/transfer.cu）正由 #30024 加宽，而 HiSparse 的 swap-in miss 拷贝版本（kernels/jit/csrc/hisparse.cuh）在 ROCm 上仍是普通 64 位标量循环、无 non-temporal 提示，同一文件里的 CUDA 分支却已使用 128 位配对 ld.global.nc.v2.b64，ROCm 路径严格窄于同文件 CUDA 路径。swap-in 位于 decode 关键路径：swap_in_selected_pages 每个 C4 层每个 decode 步在主 stream 上运行，位于 indexer 与 attention 之间，无法与计算重叠，每次 miss 都是 wavefront 直读 pinned host DRAM。

实现拆解

1. 扩展transfer_item_warp的ROCm分支（python/sglang/kernels/jit/csrc/hisparse.cuh）：新增 TransferVec4（__vector_size__(16) 的 uint32_t 向量），当源与目标均 16B 对齐且 item_size_bytes >= WARP_SIZE * 16 时执行宽路径，每个 lane 每次搬 16B；源加载使用 __builtin_nontemporal_load（流式数据只读一次、避免污染 L2/MALL），store 保持缓存语义供后续 attention 内核直接读取。宽路径结束后从 byte_pos 续走原有 64 位循环，未对齐与 sub-threshold 场景完全由 64 位循环覆盖。
2. 门控修正回归：第一版无条件加宽在 bench_hisparse.py 默认 512B item 下反而慢 14-22%，根因是 wave64 下 16B/lane 只覆盖 32/64 lane，而迭代次数与 8B 路径相同，等价于半波闲置；据此引入 item_size_bytes >= WARP_SIZE * 16 门控，保证 1024B（恰好等于门限）时整波参与、单条指令搬完一个 item。
3. 新增接缝测试（test/registered/kernels/ops/kvcache/test_hisparse.py）：test_load_cache_to_device_buffer_miss_copy_is_byte_exact 参数化覆盖 16/20/24/40B

item 在 16B 对齐与未对齐源下的 miss 拷贝，断言落点字节精确且邻槽无越界覆盖写；测试用 skipif(not is_hip()) 限定 ROCm。

4. CI 适配：第三个 commit 将测试门控到 HIP，修复其在 CUDA CI 触发 WARP_MISALIGNED_ADDRESS 导致 pytest 整体退出（exit -6）的问题，HaiShaw 用 /tag-and-rerun-ci 重跑 CI。

关键文件：

- python/sglang/kernels/jit/csrc/hisparse.cuh（模块 缓存内核；类别 source；类型 core-logic；符号 transfer_item_warp）：核心源码变更：ROCm 分支的 transfer_item_warp 新增 128 位向量 + 非临时加载路径，并用 item_size 门控避免 wave64 半波闲置回归；这是 swap-in miss 拷贝的 decode 关键路径。
- test/registered/kernels/ops/kvcache/test_hisparse.py（模块 缓存测试；类别 test；类型 test-coverage；符号 test_load_cache_to_device_buffer_miss_copy_is_byte_exact）：新增接缝测试 test_load_cache_to_device_buffer_miss_copy_is_byte_exact：此前 ROCm 用例全部使用 32B item，远低于宽路径门限，宽路径与余量循环的接缝完全没有覆盖；该测试补上了这块空白。

关键符号：transfer_item_warp, test_load_cache_to_device_buffer_miss_copy_is_byte_exact

关键源码片段

python/sglang/kernels/jit/csrc/hisparse.cuh

核心源码变更：ROCm 分支的 transfer_item_warp 新增 128 位向量 + 非临时加载路径，并用 item_size 门控避免 wave64 半波闲置回归；这是 swap-in miss 拷贝的 decode 关键路径。

```
#ifdef USE_ROCM
// 128 位向量类型：4 个 uint32 共 16B，宽路径每次搬一个 dwordx4
using TransferVec4 = __attribute__((__vector_size__(4 * sizeof(uint32_t)))) uint32_t;

__device__ __forceinline__ void transfer_item_warp(
    int32_t lane_id,
    const void* __restrict__ src_addr,
    void* __restrict__ dst_addr,
    int64_t item_size_bytes) {
  const auto src = static_cast<const char*>(src_addr);
  auto dst = static_cast<char*>(dst_addr);

  int64_t byte_pos = 0;
  // 源与目标都 16B 对齐，且 item 大小能填满整个 wavefront 时才走宽路径（每个 lane 搬 16B）。
  // 门控是承重的：实测 MI355X（wave64）上 512B item 时 16B/lane 只覆盖 32/64 个 lane，
  // 迭代次数却与 8B 路径相同，导致 14-22% 的负优化；1024B（= WARP_SIZE *
  16）时整波参与。
  const bool aligned_16b =
      ((reinterpret_cast<uintptr_t>(src) | reinterpret_cast<uintptr_t>(dst)) & 0xF) == 0;
  const bool wide_fills_wave =
      item_size_bytes >= static_cast<int64_t>(WARP_SIZE) * 16;
```

```
  if (aligned_16b && wide_fills_wave) {
    constexpr int64_t kVecBytes = static_cast<int64_t>(sizeof(TransferVec4));
    const int64_t vec_count = item_size_bytes / kVecBytes;
    const auto src_vec = reinterpret_cast<const TransferVec4*>(src);
    auto dst_vec = reinterpret_cast<TransferVec4*>(dst);
    for (int64_t i = lane_id; i < vec_count; i += WARP_SIZE) {
      // 源是 pinned host DRAM 流式数据，只读一次，用 non-temporal 加载避免污染 L2/MALL;
      // store 故意保留缓存语义，因为下一个 attention 内核立刻要读这块 device buffer。
      dst_vec[i] = __builtin_nontemporal_load(&src_vec[i]);
    }
    byte_pos = vec_count * kVecBytes;
  }

  // 64 位路径：完整覆盖未对齐场景，并处理宽路径留下的 <=8B 余量;
  // 之后的 <8B 字节尾部循环保持原样，未在片段中重复。
  const int64_t word_count = item_size_bytes / static_cast<int64_t>(sizeof(uint64_t));
  const int64_t word_start = byte_pos / static_cast<int64_t>(sizeof(uint64_t));
  const auto src_words = reinterpret_cast<const uint64_t*>(src);
  auto dst_words = reinterpret_cast<uint64_t*>(dst);
  for (int64_t i = word_start + lane_id; i < word_count; i += WARP_SIZE) {
    dst_words[i] = src_words[i];
  }
}
#endif
```

test/registered/kernels/ops/kvcache/test_hisparse.py

新增接缝测试 test_load_cache_to_device_buffer_miss_copy_is_byte_exact：此前 ROCm 用例全部使用 32B item，远低于宽路径门限，宽路径与余量循环的接缝完全没有覆盖；该测试补上了这块空白。

```
@pytest.mark.skipif(
    not is_hip(),
    reason="CUDA transfer_item_warp assumes 16B-aligned items with no sub-8B remainder.",
)
@pytest.mark.parametrize(
    "kv_dim,miss_token",
    [
        # token 0..3 常驻，查询 >=4 才会 miss；目标槽固定为 0，因此源偏移
        #  （miss_token * item_size）决定 16B 对齐检查。
        (256, 4), # 1024B：恰好等于门限，每 lane 一个 16B 步，无余量
        (257, 4), # 1028B：宽路径 + 4B 字节尾部
        (258, 4), # 1032B：宽路径 + 一个 64 位字
        (260, 4), # 1040B：lane 0 上两轮宽迭代
        (257, 5), # 1028B 且源偏移 5140 未对齐：跳过宽路径
        (5, 4), # 20B：低于门限，走 64 位循环 + 4B 字节尾部
    ],
)
def test_load_cache_to_device_buffer_miss_copy_is_byte_exact(
    kv_dim: int, miss_token: int
```

```
) -> None:
    """miss 拷贝必须对任意 item 大小与对齐都字节精确。

    本文件其它 ROCm 用例都用 32B item，远低于 WARP_SIZE * 16 宽拷贝门限，
    完全到不了宽路径，更测不到宽路径与余量循环之间的接缝；这些尺寸
    覆盖门限两侧以及每种余量形状。
    """
    item_size_bytes = kv_dim * torch.empty((), dtype=DTYPE).element_size()
    host_cache = torch.empty(
        (HOST_CACHE_SIZE, 1, kv_dim), dtype=DTYPE, device="cpu", pin_memory=True
    )
    host_cache.copy_(torch.arange(host_cache.numel(), dtype=DTYPE).view_as(host_cache))
    device_buffer = torch.full(
        (DEVICE_CACHE_SIZE, 1, kv_dim), -1, dtype=DTYPE, device=DEVICE
    )
    # slot 0..3 放 token 0..3；slot 4 是保留的新槽位。
    device_buffer_locs = torch.tensor([[0, 1, 2, 3, 4]], dtype=torch.int32, device=DEVICE)
    device_buffer_tokens = torch.tensor([[0, 1, 2, 3, -1]], dtype=torch.int32, device=DEVICE)
    for slot in range(HOT_BUFFER_SIZE):
        device_buffer[slot].copy_(host_cache[slot].to(DEVICE))
    torch.cuda.synchronize()

    top_k_tokens = torch.tensor([[miss_token]], dtype=torch.int32, device=DEVICE)
    out = torch.full_like(top_k_tokens, -1)

    load_cache_to_device_buffer_mla(
        top_k_tokens=top_k_tokens,
        device_buffer_tokens=device_buffer_tokens,
        host_cache_locs=torch.arange(
            HOST_CACHE_SIZE, dtype=torch.int64, device=DEVICE
        ).view(1, -1),
        device_buffer_locs=device_buffer_locs,
        host_cache=host_cache,
        device_buffer=device_buffer,
        top_k_device_locs=out,
        req_pool_indices=torch.arange(1, dtype=torch.int64, device=DEVICE),
        seq_lens=torch.full((1,), 8, dtype=torch.int32, device=DEVICE),
        lru_slots=torch.arange(HOT_BUFFER_SIZE, dtype=torch.int16, device=DEVICE).view(
            1, -1
        ),
        item_size_bytes=item_size_bytes,
        num_top_k=1,
        hot_buffer_size=HOT_BUFFER_SIZE,
        page_size=1,
        block_size=256,
        num_real_reqs=torch.tensor([1], dtype=torch.int32, device=DEVICE),
    )
    torch.cuda.synchronize()
```

```
    # miss 淘汰 LRU 头（slot 0，物理 loc 0）并落到这里。
    assert torch.equal(out.cpu(), torch.tensor([[0]], dtype=torch.int32))
    assert torch.equal(device_buffer[0].cpu(), host_cache[miss_token])
    # 邻槽不能被越界拷写污染。
    for slot in range(1, HOT_BUFFER_SIZE):
        assert torch.equal(device_buffer[slot].cpu(), host_cache[slot])
```

评论区精华

PR 没有 review 评论，HaiShaw 直接 APPROVED；关键讨论记录在 issue 评论与提交历史中。HaiShaw 在 issue 评论里标注 /tag-and-rerun-ci 与 ROCm/HIP gated，对应第三个 commit 将新增接缝测试门控到 HIP：该测试在 CUDA CI 触发 CUDBG_EXCEPTION_WARP_MISALIGNED_ADDRESS 使 pytest 进程 exit -6 并 fast-fail 依赖任务，而 CUDA 分支契约只支持 16B 对齐且无 sub-8B 余量的 item。另一个核心讨论是门控的必要性：commit 6ffa2684 记录无条件 128 位加宽在 512B item 下是 14-22% 的回归，隔离实验证明 non-temporal hint 单独使用近乎中性（81.1 vs 81.2 us），回归完全由拷贝宽度引入，因此 item_size_bytes >= WARP_SIZE * 16 是承重门控而非保守开关。

- 宽拷贝门控：wave64 lane 利用率 vs 拷贝宽度 (performance): 引入 item_size_bytes >= WARP_SIZE * 16 门控并保留 64 位循环作为未对齐与余量路径；1024B（DeepSeek-V4 C4 size）在低 batch 获得 13-20% 收益。
- 接缝测试在 CUDA CI 崩溃，测试门控到 ROCm (testing): 用 skipif(not is_hip()) 将测试限定在 ROCm/HIP；作者说明该测试是回归保护而非演示既有 bug，原有 13 passed / 2 skipped 保持不变。
- bench_hisparse.py 只测顺序 gather 的偶然发现 (other): 对带宽结论不构成误导，但顺序模式从代码不易看出，建议后续为 benchmark 增加随机模式作为独立改进。

风险与影响

- 风险：技术风险集中在 ROCm 专属路径： 1) 门控依赖 wave64 的 lane 利用率结论，若未来 ROCm 引入 wave32 或其他波前尺寸，门控语义与收益需要重新验证（hisparse.cuh 中 WARP_SIZE 宏决定）； 2) __builtin_nontemporal_load 改变 L2/MALL 缓存行为，当前结论只在 MI355X（gfx950）上实测，其他 ROCm 显卡或混合负载需复测； 3) 新增测试只跑 HIP，CUDA 分支未来若被改动，该测试无法提供保护； 4) 性能收益边界明确：只在低 batch 且 item_size >= 1024B 时显现，高 batch 已接近约 55 GB/s 的 host-fabric 带宽上限，不应期待端到端收益。
  - 影响：影响范围仅限 AMD/ROCm + HiSparse（host-backed unified KV cache）的 decode swap-in 路径；TTFT/prefill 走 HiCache 的 kvcacheio/transfer.cu 不受影响，CUDA 与 CPU 侧行为完全不变。性能上，1024B item 在 batch 1/10 时 swap-in 拷贝延迟分别下降约 19.9%/12.8%，batch >= 24 后基本持平（0.5% 以内）。对团队而言，本 PR 提供了 wave64 宽拷贝门控的实测方法论和接缝测试范式，后续在 ROCm 上做同类向量化优化可直接复用。
  - 风险标记：ROCm 专属路径变更，decode 关键路径，wave64 门控假设，测试仅 HIP 运行

关联脉络

- PR #30024 （标题未在材料中提供）：PR body 明确说明 #30024 加宽了 kvcacheio/transfer.cu 中的 HiCache 版本 transfer_item_warp；本 PR 是同一优化在 hisparse.cuh（HiSparse swap-in）上的对等实现，且 benchmark 结论引用了 #30024 报告的约 55 GB/s host-fabric 带宽上限。
- PR #28753 Fix/hisparse host backed max request length: 历史 PR，同属 HiSparse host-backed cache 功能线，修复设备池容量误截断长请求；本 PR 在此基础上进一步优化 miss swap-in 路径的拷贝宽度，属于同一演进方向的正确性→性能序列。
- PR #31341 （标题未在材料中提供）：PR body 列为相关 PR，属于 HiSparse 功能演进线，具体内容未在本次材料中提供。
- PR #28874 （标题未在材料中提供）：PR body 列为相关 PR，属于 HiSparse 功能演进线，具体内容未在本次材料中提供。