PR #33484 完整报告

sgl-project/sglang

perf(hisparse): fuse the DSv4 value and scale swap-in copy on ROCm

合并时间：2026-08-11 05:30

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/33484

执行摘要

- 一句话：ROCm HiSparse DSv4 swap-in 融合为单次拷贝，decode 路径提速最高约 21%
- 推荐动作：值得精读。该 PR 是典型的 ROCm 波前级性能优化案例，包含反汇编驱动的根因分析、编译期常量展开技巧、以及一个反直觉的结论（更少 pass 的 128 位拷贝反而更慢），对理解 wave64 占用率与宿主内存延迟的权衡有较高参考价值。建议关注 transfer_dsv4_item_warp 的实现细节（空 lane 处理、双层 unroll 的负载 / 存储分区）以及新测试对"未写槽位保持填充"的断言设计。若团队在 AMD 平台使用 HiSparse DSv4，此 PR 应合入且可考虑在后续 CI 中补充 ROCm 内核反汇编级别的回归检查。

功能与动机

DeepSeek-V4 C4 token 为 584 字节但并非连续内存：页面行内先有 64 个 576B value 再接 64 个 8B scale，导致 ROCm swap-in 需两次 transfer_item_warp。在 wave64 上这产生三段 wavefront pass，且 item_size_bytes 是运行时参数使编译器将 value 拷贝保留为带循环内 s_waitcnt 的滚动循环，三次 pass 在宿主内存延迟上完全串行化。PR body 明确指出 swap_in_selected_pages 位于 decode 关键路径、每个 C4 层每步执行一次，而 CUDA 分支的 transfer_item 已能合并 value 尾段与 scale，只是其 warp 逻辑硬编码为 32 lane 无法用于 ROCm。

实现拆解

1. 新增融合拷贝函数：在 python/sglang/kernels/jit/csrc/hisparse.cuh 的 ROCm #ifdef 分支中新增 transfer_dsv4_item_warp。它以 lane_id 为索引将 576B value（72 个 64 位字）与 8B scale（1 个 64 位字）视为 73 字空间，用编译期常量 kValueWords、kTotalWords、kPasses（738 字节分 64 lane 得 2 pass）驱动 #pragma unroll 的加载循环与存储循环。先循环内将所有非空槽位加载到 staged[]，再统一写入，从而让两条 global_load 先于 global_store 发出，将暴露的宿主往返从 3 次降至 1 次。
2. 替换原调用：在 load_cache_to_device_buffer_kernel 的 IsDsv4Layout ROCm 分支中，用 transfer_dsv4_item_warp(lane_id, src_value_ptr, src_scale_ptr, dst_value_ptr, dst_scale_ptr) 替换原先两次 transfer_item_warp 调用。CUDA 分支与通用（非 DSv4）路径完全不动。
3. 新增针对性测试：在 test/registered/kernels/ops/kvcache/test_hisparse.py 中新增 test_load_cache_to_device_buffer_dsv4_fused_copy_multi_miss，构造一次启动内多个 miss、源与目的 page 偏移各异（含第二页、value/scale 分界落在不同位置）的场景，从

内核输出读取落点而非假设驱逐顺序，并校验未被写入的槽位保持 0xFF 填充，防止过度拷贝越界。

4. 性能数据与方案权衡：PR body 给出 MI355X 上 batch 1/10/24/100 的实测提升，并对比了单次 128 位 pass 方案：虽然 pass 数更少但 lane 参与度低（36 个 16B 向量 + 1 个 scale），反而在 batch 1 比基线慢约 1%–3.6%，说明 64 位全 wavefront 占用优于 128 位部分占用，与 #33085 的门控效应一致。
5. 配套说明：无 API、配置、模板或文档改动；C++ 变更严格限制在 ROCm #ifdef 内。

关键文件：

- python/sglang/kernels/jit/csrc/hisparse.cuh（模块 HiSparse；类别 other；类型 core-logic；符号 transfer_dsv4_item_warp, load_cache_to_device_buffer_kernel）：核心实现文件，新增 transfer_dsv4_item_warp 并替换 ROCm DSv4 分支的两次 transfer_item_warp 调用，是本次性能优化的主要载体。
- test/registered/kernels/ops/kvcache/test_hisparse.py（模块 HiSparse；类别 test；类型 test-coverage；符号 test_load_cache_to_device_buffer_dsv4_fused_copy_multi_miss）：新增 test_load_cache_to_device_buffer_dsv4_fused_copy_multi_miss 测试，覆盖多 miss、可变源 / 目的偏移（含第二页与 value/scale 分界错位）以及未写槽位填充校验，是保障该优化正确性的关键配套。

关键符号：transfer_dsv4_item_warp, load_cache_to_device_buffer_kernel, test_load_cache_to_device_buffer_dsv4_fused_copy_multi_miss

关键源码片段

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

核心实现文件，新增 transfer_dsv4_item_warp 并替换 ROCm DSv4 分支的两次 transfer_item_warp 调用，是本次性能优化的主要载体。

```
// 本函数在 ROCm 分支内，用于将一个 DSv4 C4 token（576B value + 8B scale）
// 作为单个 73 字空间拷贝，替代原先两次 transfer_item_warp 调用。
// 关键点：kTotalWords 是编译期常量，因此两轮 #pragma unroll 循环可以在
// 编译期确定 pass 数，使所有 global_load 先发出、再统一 global_store，
// 从而把宿主内存往返从 3 次降为 1 次。
__device__ __forceinline__ void transfer_dsv4_item_warp(
    int32_t lane_id,
    const int64_t* __restrict__ src_value,
    const int64_t* __restrict__ src_scale,
    int64_t* __restrict__ dst_value,
    int64_t* __restrict__ dst_scale) {
  constexpr int32_t kValueWords = static_cast<int32_t>(device::hisparse::kValueBytes /
  sizeof(int64_t));
  constexpr int32_t kTotalWords = static_cast<int32_t>(device::hisparse::kItemBytes /
  sizeof(int64_t));
  constexpr int32_t kPasses = (kTotalWords + WARP_SIZE - 1) / WARP_SIZE;
  // 对齐假设必须在编译期成立，否则无法安全按 64 位字访问
  static_assert(device::hisparse::kValueBytes % sizeof(int64_t) == 0, "value must be whole 64-bit
  words");
```

```
  static_assert(device::hisparse::kScaleBytes % sizeof(int64_t) == 0, "scale must be whole 64-bit
  words");

  const int64_t* src_slot[kPasses];
  int64_t* dst_slot[kPasses];
  int64_t staged[kPasses];

  // 第一轮：所有 lane 把各自负责的字加载到 staged，空 lane 对应的槽位为 nullptr
#pragma unroll
  for (int32_t p = 0; p < kPasses; ++p) {
    const int32_t i = p * WARP_SIZE + lane_id;
    const bool is_value = i < kValueWords;
    const bool is_scale = !is_value && i < kTotalWords;
    src_slot[p] = is_value ? src_value + i : (is_scale ? src_scale + (i - kValueWords) : nullptr);
    dst_slot[p] = is_value ? dst_value + i : (is_scale ? dst_scale + (i - kValueWords) : nullptr);
    if (src_slot[p] != nullptr) staged[p] = *src_slot[p];
  }
  // 第二轮：统一写入，编译器可根据依赖排除同一地址的冲突，
  // 并让两条 load 先于 store 发出（反汇编验证：两个 global_load 后跟一个
  // s_waitcnt 再是 global_store）
#pragma unroll
  for (int32_t p = 0; p < kPasses; ++p) {
    if (dst_slot[p] != nullptr) *dst_slot[p] = staged[p];
  }
}
```

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

新增 test_load_cache_to_device_buffer_dsv4_fused_copy_multi_miss 测试，覆盖多 miss、可变源 / 目的偏移（含第二页与 value/scale 分界错位）以及未写槽位填充校验，是保障该优化正确性的关键配套。

```
@pytest.mark.skipif(
    not is_hip(), reason="Covers the ROCm wavefront64 fused DSv4 token copy."
)
def test_load_cache_to_device_buffer_dsv4_fused_copy_multi_miss() -> None:
    """Several DSv4 misses in one launch must each land byte-exact.

    融合拷贝把 576B value 与 8B scale 当作 73 字空间行走，
    因此 value/scale 分界落在 lane 索引上而不是调用边界上。
    这里同时变化源与目的 page 偏移，包括第二页上的 token，
    确保分界不会总落在同一地址。
    """
    hot_buffer_size = 4
    num_pages = 2
    # seq_len 保持高于查询到的 token，避免新 token 被内核不经宿主拷贝直接放置
    seq_len = 16
    host_locs = list(range(seq_len))
    miss_tokens = [4, 5, 6, 7]
    # 源偏移：页中间、页 0 最后槽、页 1 第一槽、页 1 最后槽
```

```
for token, loc in zip(miss_tokens, [10, 63, 64, 127]):
    host_locs[token] = loc
# 目的偏移：页 0 第一、第二、最后槽，然后页 1
device_locs = [0, 1, 63, 64, 65]

# 用固定 seed 写入 host_cache，使每个 token 内容可预期
host_cache = torch.zeros(
    (num_pages, DSV4_PAGE_BYTES), dtype=torch.uint8, device="cpu", pin_memory=True
)
for loc in host_locs:
    _write_dsv4_token(host_cache, loc, seed=loc + 1)

# 设备缓冲填充 0xFF，用于后续检测未被拷贝的槽位是否被意外覆写
device_buffer = torch.full(
    (num_pages, DSV4_PAGE_BYTES), 0xFF, dtype=torch.uint8, device=DEVICE
)

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

load_cache_to_device_buffer_dsv4_mla(
    top_k_tokens=top_k_tokens,
    device_buffer_tokens=torch.tensor(
        [[0, 1, 2, 3, -1]], dtype=torch.int32, device=DEVICE
    ),
    host_cache_locs=torch.tensor([host_locs], dtype=torch.int64, device=DEVICE),
    device_buffer_locs=torch.tensor([device_locs], dtype=torch.int32, device=DEVICE),
    host_cache=host_cache,
    device_buffer=device_buffer,
    top_k_device_locs=out,
    req_pool_indices=torch.tensor([0], dtype=torch.int64, device=DEVICE),
    seq_lens=torch.tensor([seq_len], dtype=torch.int32, device=DEVICE),
    lru_slots=torch.arange(hot_buffer_size, dtype=torch.int16, device=DEVICE).view(1, -1),
    item_size_bytes=DSV4_ITEM_BYTES,
    num_top_k=len(miss_tokens),
    hot_buffer_size=hot_buffer_size,
    page_size=DSV4_PAGE_SIZE,
    block_size=256,
    num_real_reqs=torch.tensor([1], dtype=torch.int32, device=DEVICE),
)
torch.cuda.synchronize()

# 每个 miss 驱逐到哪个槽由 LRU 决定，因此从内核输出读取实际落点，
# 只要求落点各不相同且在合法集合内
landed = out.cpu().tolist()[0]
assert len(set(landed)) == len(landed)
assert set(landed).issubset(device_locs)

# 逐 token 校验 byte-exact：目的槽内容必须与源完全一致
```

```
    device_cpu = device_buffer.cpu()
    for token, dst_loc in zip(miss_tokens, landed):
        assert torch.equal(
            _read_dsv4_token(device_cpu, dst_loc),
            _read_dsv4_token(host_cache, host_locs[token]),
        ), f"token {token} -> device loc {dst_loc}"

    # 未被内核写过的槽位必须保持 0xFF 填充,
    # 这样任何超出 value 或 scale 边界的过度拷贝都会被抓住
    for loc in set(device_locs) - set(landed):
        assert torch.all(_read_dsv4_token(device_cpu, loc) == 0xFF)
```

评论区精华

Review 评论整体较少。HaiShaw 的最终 review 仅一句 ROCm/HIP gated 并 APPROVED，表明该改动被确认限定在 ROCm/HIP 编译门内，未触及 CUDA 路径。PR body 中作者详细记录了反汇编证据与两个备选方案的对比，核心讨论点是：为什么不用单次 128 位 pass（更少 round trip）？结论是 wave64 下 64 位全 lane 占用比 128 位部分 lane 占用更能隐藏宿主内存延迟，这一结论与同仓库 #33085 的观察一致。该决策属于实现层面的自我论证，未在评论区引发公开交锋，但体现了对 wavefront 占用率与访存延迟的深入考量。

- ROCm 路径门控确认 (design): Pr 被批准，改动被限定在 ROCm 条件编译内。
- 单次 128 位 pass 方案为何被否决 (performance): 采用两趟 64 位全 wavefront 方案，放弃单趟 128 位方案。

风险与影响

- 风险：风险点集中在新增内核路径的正确性与回归上：1）transfer_dsv4_item_warp 中 src_slot[p]/dst_slot[p] 为 nullptr 的空 lane 处理依赖 is_value/is_scale 判断，若 kValueBytes/kScaleBytes 与 8 字节对齐假设被破坏（已用 static_assert 保护）或未来 DSv4 布局变化，可能导致越界访问；2）测试仅覆盖 ROCm（skipif not is_hip()），CUDA 侧的 DSv4 paged 用例本就不走该路径，但新测试并未在 CUDA 上执行，若未来统一入口被改动可能漏检；3）性能优化依赖编译器对 #pragma unroll 与加载 / 存储顺序的调度，不同 ROCm 编译器版本可能生成不同 SASS，收益幅度存在波动风险；4）load_cache_to_device_buffer_kernel 是 decode 关键路径，任何错误都可能静默污染 KV 缓存，所幸新增测试校验了 byte-exact 与未写槽位填充，能抓住大部分越界或错位问题。
- 影响：影响范围限定在 ROCm (AMD) 平台、HiSparse DSv4 分页布局的 miss swap-in 路径，即 DeepSeek-V4 类模型在 HIP 上的 decode 关键路径。收益显著：batch 1 提升约 14%–17%，batch 10 提升约 20%–21%，且随 batch 增大收益衰减（带宽受限后 round-trip 不再主导）。对 CUDA 用户无影响，对非 DSv4 布局无影响。团队侧收益：为未来 DSv4 或类似非连续 token 布局的 ROCm 拷贝提供了可复用的实现范式，测试方法（多 miss、可变偏移、校验未写槽位）也可供后续 kernel 测试参考。
- 风险标记：ROCm 专用路径，decode 关键路径，依赖编译器展开行为，测试仅覆盖 HIP

关联脉络

- PR #33085 perf(hisparse): 128-bit non-temporal swap-in copy on ROCm: 同一模块 HiSparse 的 ROCm swap-in 性能优化，PR body 明确引用其“更宽加载反而更慢”的门控效应作为本 PR 设计取舍的依据，两者是同一性能优化系列。
- PR #28753 Fix/hisparse host backed max request length: 修复 HiSparse 长请求被设备池容量误截断，涉及同一 hisparse.cuh 与 host-backed 缓存路径，是 HiSparse 功能演进的另一分支。
- PR #33639 [Hicache][2/2]Support Mamba branching in Unified Radix Cache with HiCache: HiCache 相关功能扩展，涉及统一内存池与 HiSparse 组件协作，本次 PR 的性能优化为其在 DSv4/AMD 场景的落地提供支撑。
- PR #33974 [unified memory] Support DSPARK speculative decoding + fix two NaN root causes (page hand-out zeroing, CuTe int32 slot-stride wrap): 统一内存池与 DSv4/MLA 相关修复，涉及同一 kernel 路径的稳定性改进，与本 PR 在 DSv4 布局处理上存在关联。